Dans les coulisses de Microsoft Windows

Arnaud Maillard

Dans les coulisses de Microsoft Windows

Arnaud Maillard

Ce document a été mis à jour pour la dernière fois le **29 décembre 2025**. Il a à cette occasion été distribué sous le numéro de version **0.3.0.2**. Les lecteurs sont invités à vérifier qu'ils sont en possession de la dernière évolution via le site web **ntoskrnl.org**.

Cet ouvrage est publié sous licence Creative Commons (CC BY-NC-ND 4.0). Vous êtes autorisé à copier, distribuer et communiquer cette œuvre par tous moyens et sous tous formats, mais sans la modifier. Vous n'êtes pas autorisé à faire un usage commercial de cette œuvre, ni de tout ou partie du matériel la composant. Vous n'êtes pas autorisé à appliquer des conditions légales ou des mesures techniques qui restreindraient légalement autrui à utiliser l'œuvre dans les conditions décrites par la licence

1. Introduction 1
À qui s'adresse ce livre ? 1
Structure de ce livre 1
À propos de la licence 2
Outils et licences d'utilisation 2
Assistance 3
Précisions sur le glossaire 3
En ce qui concerne la lecture 3
Travaux en cours 3
2. Concepts et outils 5
Systèmes d'exploitation 5
Ce qu'est un noyau 5
Différents types de noyaux 5
Ce que propose un système d'exploitation 6
Fonctions d'un système d'exploitation 6
Exemples de systèmes d'exploitation 7
Mémoire virtuelle 7
Ordonnancement 8
Objectifs d'un ordonnanceur 8
Critères d'ordonnancement 9
Concepts et terminologie 9
Versions de Windows 9
Examen de la version courante du système d'exploitation 11
Fonctions, services et routines 13
Documentation 14
Processus, threads et autres unités fonctionnelles 14
Gestionnaire des tâches 16
Registre 17
Notation hongroise 18
Généralités 18
Nommage des identifiants 18
Histoire 19
Notation hongroise *Apps* 19
Notation hongroise *Systems* 20
Raison d'être et avantages 20
Inconvénients 21
MSDN (*Microsoft Developer Network*) 22
Sites web 22
Bibliothèque MSDN 22
Forums 23
Blogs 23
Base de connaissances 23
Magazine 24
Traduction automatique 24
Mode utilisateur et mode noyau 24
Anneaux de protection 25
Visualisation des temps utilisateur et noyau 25
Windows-1252 et Unicode 27
Déterminer si un texte est ANSI ou UNICODE 28
UNC (*Uniform Naming Convention*) 29
Système de fichiers 29
Services 30
Espace de noms 30
API Windows 31
Historique 31

Composants de l'API Windows 31
Autres implémentations 32
API Native 32
API noyau 33
Valeurs NTSTATUS 34
Chaînes Unicode 69
Objets et handles 69
Interface binaire-programme 70
Mot machine et types fondamentaux 70
Microsoft Management Console (MMC) 71
Composants logiciels enfichables 72
Modes d'accès d'une console MMC 72
Exécuter une console en mode Auteur 73
Invite de commandes (Cmd) 73
Démarrer des programmes 73
Syntaxe de la ligne de commande Cmd 73
Commandes de base 74
Interprètes de commandes 76
Fonctionnalités de Windows 77
Ajouter ou supprimer des fonctionnalités Windows 77
Windows Update 77
Pilotes de périphérique 78
Panneau de configuration 78
GUID 79
Polices et fontes 80
Services de terminal 80
SDK et WDK 80
Convention d'appel 80
Pile d'appels 81
Au coeur de Windows 81
Build libre et build controlé 82
Vérification de la version (build) exécutée 83
DebugView 84
Flux de sortie de déboggage 84
Coup d'oeil sur l'interface DebugView 84
Enregistrement du flux de sortie mode utilisateur 85
Capture du flux de sortie mode noyau 86
Contrôle à distance 86
Enregistrement et journaux 87
Options de filtrage 87
Handle 88
Tasklist 88
WinDbg 90
Directives de WinDbg 90
Débogage noyau local 91
Commandes de manipulation des threads 91
Commandes de registres 92
Commandes d'affichage de la mémoire 92
Commandes de niveau image 93
Utilitaires Sysinternals 94
Configuration et utilisation des symboles 94
Généralités concernant les symboles 94
Chemin des symboles 95
Validation des symboles 95
Chargement différé des symboles 96

Commande symopt 97
BCDEdit 97
Identificateurs bien connus 98
Gestion de l'ordinateur 99
Utilitaire Configuration du système (Msconfig) 99
Utilitaires d'administration 100
Variables d'environnement 101
Console Windows 103
Observateur d'événements 103
LiveKd 104
Driver Verifier 104
Systeminfo 106
Obtention d'informations avancées sur le système (Msinfo) 106
MASM 107
Architecture des ordinateurs 107
Multiprocesseurs et multicoeurs 107
Simultaneous Multithreading 108
Pipeline 108
Listes 109
Cartes binaire (bitmap) 109
3. Architecture du système 111
Architecture générale de Microsoft Windows 111
Fichiers système fondamentaux 111
Aperçu de l'architecture 112
Approche en couches 113
Objectifs de conception 114
Fiabilité et robustesse 114
Haute performance 116
Internationalisation 117
Extensibilité 118
Compatibilité 119
Portabilité 120
Sécurité 121
Modèle de système d'exploitation 123
Systèmes multi-utilisateur 123
Sous-systèmes d'environnement 124
Bibliothèques de sous-système 124
Informations de démarrage de sous-système 125
Environnement MS-DOS (NTVDM) 125
Sous-systèmes et création de processus 126
Composants fondamentaux du système 127
Ntdll.dll 127
Exécutif 127
Gestionnaire de processus 129
Couche d'abstraction matérielle (HAL) 129
Noyau 130
Processus système fondamentaux 131
Processus inactif du système 132
Winlogon 132
Userinit 133
Wininit 134
SMSS (*Session Manager Subsystem*) 134
Processus système initial 135
Sous-système Windows 135
Graphics Device Interface (GDI) 136

SYSTEM_INFORMATION_CLASS 137
NUMA 150
Hyperthread 150
Multitraitement 151
Multitâche 151
4. Mécanismes système 153
Windows on Windows 153
Wow32 153
Considérations sur la compatibilité 16 bits 153
Limitations et restrictions concernant Wow32 154
Wow64 154
Composants Wow64 155
Espace d'adressage des processus Wow64 155
Appels système 155
Impression 156
Performances de Wow64 156
Communication entre processus Wow64 et processus natifs 156
Pilotes en mode noyau 157
Limitations et restrictions inhérentes à Wow64 157
Redirection du système de fichiers 158
Variables d'environnement 159
Flags globaux de Windows 159
Logiciel *Global Flags* 161
Flags globaux d'image 162
Commande d'extension *!gflag* 163
Options d'exécution d'image 165
Performances 166
Compteurs de performance 166
Analyseur de performances (Perfmon) 166
Ensembles de collecteurs de données 168
Ensembles de collecteurs de données intégrés 168
Création d'un ensemble de collecteurs de données 168
Création de collecteurs depuis l'invite de commandes 169
Moniteur de fiabilité 169
Accéder au moniteur de fiabilité 170
Index de stabilité système 170
Windows System Assessment Tool (WinSAT) 171
Emplacements du registre 171
WEI (*Windows Experience Index*) 171
Emplacements de registre pour WinSAT 172
Composants de l'architecture Performances 173
PLA (*Performance Logs and Alerts*) 173
PDH (*Performance Data Helper*) 173
Fonctions PHD 174
Codes d'erreur 174
Event Tracing for Windows 177
Contrôleurs 177
Consommateurs 177
Trace des événements noyau 177
Structure d'un enregistrement 178
Windows Resource Protection 179
Vérificateur de ressources (Sfc) 179
Threads système auxiliaires 180
LPC (*Local Procedure Call*) 181
APC (*Asynchronous Procedure Calls*) 183

Niveaux de requête d'interruption (IRQL) 185
IRQL prédéfinis 186
Restrictions au niveau IRQL haut 186
Motivations du modèle IRQL 187
Kernel Transaction Manager (KTM) 187
Hotpatch 189
Débogage mode utilisateur 189
5. Mécanismes de gestion 191
Registre 191
Origines du registre 191
Composants du registre 192
Scénarios d'utilisation du Registre 192
Organisation du registre 193
Types de données du registre 193
REG_BINARY 194
REG_DWORD 194
REG_EXPAND_SZ 194
REG_FULL_RESOURCE_DESCRIPTOR 194
REG_LINK 194
REG_MULTI_SZ 194
REG_NONE 194
REG_QWORD 195
REG_RESOURCE_LIST 195
REG_SZ 195
Structure du registre 195
Clés volatiles 195
Clés racines 195
HKEY_CURRENT_USER 196
HKEY_CURRENT_USER 196
HKEY_LOCAL_MACHINE 196
HKLM\SECURITY 197
HKLM\HARDWARE 197
HKLM\SOFTWARE 197
HKLM\SYSTEM 197
HKLM\SYSTEM\CurrentControlSet 197
HKEY_PERFORMANCE_DATA 198
HKEY_CURRENT_CONFIG 198
HKEY_CLASSES_ROOT 198
Visualisation et modification du Registre 199
Editeur du Registre 199
Manipuler le registre via Windows PowerShell 199
Fichiers d'inscription 200
Utilitaire Reg (Reg.exe) 200
Registre et sécurité 201
Callbacks de registre 202
Registre et droits d'accès 202
NtQueryKey 203
NtSetInformationKey 205
Ruches 205
Services 206
Applications service 207
Base de données des services 207
Type de services 209
États d'un service 210
Démarrage des services 211

Acceptation de l'amorçage 212
SCM (*Service Control Manager*) 212
Objectifs de conception des services Windows 212
Services Windows et tâches planifiées 213
Comptes de services 213
Compte système local 214
Compte service réseau 215
Compte service local 215
Exécution de service sous un compte suppléant 215
Démarrage automatique différé 216
Démarrer et arrêter des services 216
Gérer les services 217
Service Control (Sc.exe) 217
Onglet Services du Gestionnaires des tâches 218
Services.msc 218
Erreur de démarrage 219
Sécurité des services 219
Contrôle des services 219
Erreur des services 220
Codes d'erreur d'E/S service 220
Arrêt des services 221
Processus partagés entre des services 222
Balises de service 223
Noms des services 223
Tâches planifiées 224
SchTasks 224
Protection et restauration du système 224
Créer un point de restauration 224
Restaurer les fichiers et paramètres système 225
Windows Management Instrumentation (WMI) 225
Fonctionnalités de WMI 225
Architecture WMI 226
WMIC 227
Sécurité WMI 227
PowerShell 227
Considérations sur les langages de script dans Microsoft Windows 228
Vue d'ensemble de Windows PowerShell 229
Environnements interactif et de script 229
Orientation objet 229
Types de commandes PowerShell 229
Applets de commande 230
Fonctions shell 230
Commandes natives 231
Scripts 231
WER (*Windows Error Reporting*) 232
Journaux d'événements 234
Types d'événements 235
6. Gestionnaire d'objets 236
Gestionnaire d'objets 236
Structure des objets 237
Catégories d'objet 238
Objets noyau 238
Objets exécutif 239
Autres objets 241
Objets d'interface utilisateur 241

Objets GDI 242
Objets type 243
Comptabilisation des handles ouverts pour chaque type d'objet 246
Méthodes d'objet 248
Noms des objets 249
Objets de base nommés 250
WinObj 251
Objet lien symbolique 252
Répertoires d'objet 253
Objets temporaires et objets permanents 254
Partage d'objets entre processus 255
Handles d'objet et table des handles de processus 255
Table des handles noyau 256
Démantèlement de handles 257
Rétention des objets 257
Sécurité des objets 258
OBJECT_INFORMATION_CLASS 258
Espace de noms d'objets 259
7. Synchronisation 261
Terminologie de synchronisation 261
Mécanismes de synchronisation 261
Problèmes classiques de synchronisation 263
Exclusion mutuelle 264
Synchronisation au niveau IRQL haut 264
Synchronisation au niveau IRQL bas 265
Sémaphores 265
À propos générique concernant les sémaphores 265
Structure de données 266
Interfaces Windows de sémaphore 266
Compteur et état de signal de sémaphores 266
Partage d'objets sémaphore 267
Limite 267
Synchronisation sur le plan utilisateur 268
Sections critiques 268
Généralités 268
Utilisation 268
Extension de commande !cs 269
Structures de données et mécanismes internes 270
Liste des sections critiques de processus 272
Sections critiques et événements à clé 272
Limites 273
Sections critiques et systèmes multi processeurs 273
Délai de sections critiques 274
Variables conditionnelles 274
Verrous lecture-écriture 275
Mutex 276
Mutex noyau 278
Mutex abandonné 278
Mutex rapides 279
Mutex gardés 280
Spinlocks 281
Spinlocks en file 282
Spinlocks en file dynamiques 283
Spinlocks lecture/écriture 283
Spinlocks d'interruption 283

Pushlocks 284
Événements 284
Evénements à clé 287
Evénements appariés 288
Ressources exécutif 288
Visualisation des ressources exécutif 290
Objets noyau dispatcher 291
Rundown 294
Horloges et minuteries (timers) 294
Opérations inter verrouillées 297
Région critique et région gardée 297
8. Processus et threads 299
Dans les coulisses des processus 299
Structure de données 299
Objet processus de l'exécutif (EPROCESS) 299
Visualisation du format de la structure EPROCESS 300
Objet processus du noyau (KPROCESS) 311
PEB (*Process Environment Block*) 314
Visualisation des données du bloc PEB 314
Examen du bloc PEB 316
Attributs du bloc PEB 317
Accès au PEB 324
Bloc Win32Process 324
CSR_PROCESS 325
Classification des processus 326
Interfaces 327
Variables noyau 327
Visualisation des processus actifs 328
Examen des processus via le Gestionnaire des tâches 328
Flux de CreateProcess 330
Attributs de CreateProcess 330
Hiérarchie de processus 331
Processus inactif du système 332
Processus protégés 333
Processus protégés légers (PPL) 335
Processus minimaux 336
Processus UWP 336
Processus Pico 336
Identité de processus 337
Identifiant de processus 337
Visualisation des informations d'identité d'un processus 338
Heures de processus 339
Visualisation des heures de processus 339
Image exécutée 339
Visualisation de l'image support d'un processus 340
Processus créateur 340
Visualisation des liens de parenté entre deux processus 340
Création de processus 341
Processus et synchronisation 341
Gérer les processus 341
Handles d'instances 342
Pseudo-handles 342
Notifications de niveau processus 342
Notification de création et suppression des processus 342
Terminaison de processus 343

Bureau 344
Station-fenêtre 345
Enumération de processus 346
Sécurité des processus 346
Service NtQueryInformationProcess 347
Liste des processus actifs 351
Variables d'environnement de processus 351
Répertoires de travail d'un processus 351
Dans les coulisses des threads 351
Généralités concernant les threads 351
Fonctions 352
Structure de données 352
Bloc thread de l'exécutif (ETHREAD) 352
Bloc thread du noyau (KTHREAD) 355
TEB (*Thread Environment Block*) 361
Région et bloc de contrôle du processeur 365
Variables noyau 375
Compteurs de performance des threads 375
Restrictions concernant les threads des processus protégées 376
Thread courant 376
Identifiant de thread 376
Terminaison de threads 377
Annulation de threads 377
Création de threads 378
Sécurité des threads 379
THREADINFOCLASS 379
Heures de thread 381
Traitement multithread 381
Notifications de niveau thread 382
Threads système 382
Ordonnancement des threads 383
États d'un thread 383
Fonctions Windows d'ordonnancement 384
Contexte d'exécution 384
Commutation de contexte 386
Classes de priorité 386
Suspension et reprise de thread 387
Affinité 388
Affinité système 389
Thread inactif 390
Accélérations de priorité 390
Accélération de priorité pour les tâches multimédia (MMCSS) 390
Scénarios d'ordonnancement 391
Basculement volontaire 392
Préemption 392
Fin de quantum 392
Fin 392
Généralités sur l'ordonnancement dans Windows 392
Quantum 393
Accroissement de quantum 393
Processeur idéal 394
Base de données du dispatcher 394
Systèmes multi processeur 395
Systèmes NUMA 395
Systèmes hyperthread 395

Base de données du dispatcher multi processeur 396
Algorithmes d'ordonnancement multi processeur 396
Niveaux de priorité de thread 396
Groupes de processeur 396
DLL (*Dynamic Link Library*) 397
Fonctions 398
DLL connues 399
Visualisation des DLL chargées 400
Point d'entrée DllMain 401
Communication inter processus 401
Canaux de transmission (*pipes*) 402
Tubes anonymes et tubes nommés 402
Opérations sur des tubes nommés 402
Passage de messages 402
WM_COPYDATA 407
Fibres 408
Débogage 408
Groupe de processus (Objets job) 409
Threads auxiliaires utilisateur 417
Paramètres de langue et d'Internationalisation 418
Conclusion 418
9. IPC 419
Tubes 419
Atomes globaux 419
Boites aux lettres (*Mailslots*) 420
10. Gestion de la mémoire 421
Principes de base de la mémoire 421
ROM 421
Gestionnaire de mémoire 421
Fonctionnalités clé du gestionnaire de mémoire 422
Synchronisation interne 422
Configuration du gestionnaire de mémoire 423
Paramètres de démarrage 423
Visualisation de l'utilisation de la mémoire 424
Gestion du tas 424
Sur la notion de tas 425
Tas et processus 425
Gestionnaire de tas 426
Structure du gestionnaire de tas 427
Synchronisation de tas 427
Verrouillage du tas 428
Fonctionnalités de débogage du tas 428
Visualisation des tas d'un processus 429
Types de tas 430
Tas NT 430
Tas segment 431
Options de contrôle du tas 432
ASLR (*Address Space Layout Randomization*) 433
Copy-on-write 434
Réservation et engagement de pages 435
Informations concernant l'espace d'adressage 436
Stratégies d'allocation 437
Protection de la mémoire 437
DEP (*Data Execution Prevention*) 439
DEP de niveau logiciel 440

Verrouillage de la mémoire 441
Modified Page Writer 441
Balance Set Manager 441
Examen du thread exécutant l'écrivain de pages modifiés 442
Défauts de page 442
Collision de défauts de page 443
Page système partagée 444
Pools de mémoire système 447
Pool tagging 448
Taille des pools 449
Surveillance de la consommation des pools 449
Granularité d'allocation 451
Grandes et petites pages 451
Géographie de l'espace d'adressage virtuel 452
Géographie de l'espace d'adressage utilisateur x86 452
Géographie de l'espace d'adressage système x86 453
Géographie de l'espace d'adressage virtuel utilisateur 453
Visualisation de l'espace d'adressage virtuel utilisateur 454
Géographie de l'espace d'adressage 64 bits 454
Traduction d'adresse 455
Traduction d'adresse virtuelle x86 455
Répertoire de pages 455
Traitement des défauts de page 456
Tables de pages et PTE 456
Base de données PFN 457
Dynamique des listes de pages 458
Priorité de page 458
Visualisation de la base de données PFN 459
Visualisation des threads Prêt 460
Thread de page à zéro 461
Listes look-aside 462
Pagination à la demande 462
Prefetcher 462
Working sets 463
Working set système 464
Working Set Manager 464
Working sets de processus 465
VAD (Virtual Address Descriptor) 466
Visualisation de VAD 466
Page de garde 467
Fichiers de pagination 467
Objets section 469
Piles 472
Piles utilisateur 472
Pile DPC 473
Pile d'interruption 473
Piles noyau 473
Notifications de statut de la mémoire 473
Partition mémoire 475
TLS (Thread Local Storage) 477
AWE (Address Windowing Extensions) 477
Vérification de la gestion mémoire 477
Memory Descriptor List (MDL) 478
PTE système 479
11. Portable Executable 480

Historique 480
Vue d'ensemble du format 480
Structure d'un fichier PE 480
Entête MS-DOS 480
Micro-programme mode réel 481
Entête PE 482
Types de machine 482
Caractéristiques 483
Entête optionnelle 484
DllCharacteristics 489
Sous-système 489
Répertoires 489
Entête de sections 490
Dumpbin 492
Chargeur d'image 493
Base de données des modules 494
Initialisation post traitement 496
Import address Table (IAT) 497
12. Sécurité 499
Composants du système de sécurité 499
Contrôle d'accès 500
Jetons d'accès 500
Types de jetons 506
Jetons restreints 506
Structure commune de sécurité 507
Descripteurs de sécurité 507
Visualisation d'un descripteur de sécurité 509
ACL 510
ACE 511
DACL 512
SACL 512
SID (*Security Identifier*) 512
SID (*Security Identifier*) 515
Niveaux d'intégrité 515
LSASS (*Local Security Authority Subsystem Service*) 517
Stratégies de restriction logicielle 518
Règles de stratégies et niveaux de sécurité 519
Exceptions 519
Conflits de règles 520
Paramètres de stratégies 520
Configurer les stratégies de restriction logicielle 520
Windows Defender 520
Paramètrer Windows Defender 521
Compatibilité avec ELAM 522
Analyser la station de travail avec Windows Defender 522
Définitions de virus et autres programmes malveillants 522
Service WinDefend 523
Automatisation de Windows Defender via PowerShell 523
Mise en quarantaine et autorisation des programmes 524
ELAM (Early Launch Anti-Malware) 524
Désactiver ELAM 524
Situation d'ELAM dans la séquence de démarrage 525
Fonctionnement interne de ELAM 525
SmartScreen 525
Configuration de SmartScreen 526

Contrôles de réputation SmartScreen 526
Authenticode 527
Protection par Authenticode 528
Autorités de confiance 528
Anatomie d'une signature numérique Authenticode 528
Limitations d'Authenticode 529
Kernel Patch Protection (PatchGuard) 529
Code Integrity 530
BitLocker 531
Droits utilisateur et privilèges 531
Droits utilisateur 532
Privilèges 532
Secure Boot 537
Emprunt d'identité 537
Utilisateurs et groupes 538
Comptes utilisateurs et comptes de groupes 538
Comptes utilisateurs 538
Comptes utilisateurs locaux 538
Comptes utilisateurs prédéfinis 538
Compte Administrateur 538
Compte Invité 539
Comptes utilisateurs prédéfinis système 539
Comptes d'utilisateurs de domaine 539
Comptes de groupes 539
UAC 541
Choisir le taux d'intervention du Contrôle du compte d'utilisateur 542
Exécuter un programme en tant qu'administrateur 543
Utilitaire Whoami 543
Session secondaire (RunAs) 544
Configuration des comptes 544
Net User 545
Stratégies de mots de passe 545
Stratégies de verrouillage de compte 546
Gérer les sessions des utilisateurs 546
Visualisation des autorisations affectées aux utilisateurs 546
Droits d'accès et masques d'accès 547
Droits d'accès génériques 547
Système à défaillance rapide (fastfail) 549
Mesures d'atténuation de niveau processus 550
Control Flow Guard (CFG) 552
User Interface Privilege Isolation (UIPI) 554
13. Réseau 555
BITS (Background Intelligent Transfer Service) 555
Modèle OSI 555
Commandes *net* 555
Netsh 556
Résolution de noms 557
WINS (Windows Internet Name Service) 557
Composants de mise en réseau Windows 557
Windows Filtering Platform (WFP) 558
DFS (Distributed File System) 558
Active Directory 559
TDI 559
Remote Differential Compression (RDC) 559
Winsock 559

14. Système d'E/S 561
Composants du système d'E/S 561
Gestionnaire d'entrée/sortie 562
Matériel d'entrée/sortie 562
IRP (I/O Request Packet) 563
Emplacements de pile IRP 568
Visualisation des routines de ventilation d'un pilote 569
Visualisation des IRP en instance d'un thread 570
Gestionnaire de périphériques 570
Pilotes de périphériques 571
Types de pilotes 572
Structure d'un pilote 573
Informations de pilote 574
Objet pilote 574
Objets périphérique 576
Types de périphériques 578
Requête d'E/S destinée à un pilote mono sphère 579
E/S rapides (*Fast I/O*) 580
Plug and Play 581
Gestionnaire Plug and Play 581
Installation de pilotes 582
Niveau de prise en charge Plug and Play 582
Pilotes Plug and Play 582
Pilotes non Plug and Play 583
Objet fichier 583
Énumération des périphériques 585
Affichage de l'arborescence des périphériques 586
Attributs de niveau fichier 586
FILE_INFORMATION_CLASS 588
Gestion des fichiers 591
Droits d'accès aux fichiers 591
Ports de complétion 592
Gestion des tampons IRP 592
E/S directes 594
E/S ni directes ni tamponnées 594
WDM et WDF 594
Kernel-Mode Driver Framework (KMDF) 595
Structure d'un pilote KMDF 595
Modèle de données KMDF 596
Attributs d'objet 597
Affichage des pilotes KMDF 598
Pilotes signés et non signés 598
Magasin de pilotes 599
User Mode Driver Framework (UMDF) 599
Architecture WDF 600
Comparaison entre pilotes WDM et pilotes WDF 601
Types d'E/S 601
E/S synchrones et E/S asynchrones 601
E/S scatter/gather 602
Vérification des E/S 602
Ports d'achèvement d'E/S 603
Examen du répertoire \Driver 603
Examen du répertoire \Device 603
Déchargement de pilote 604
Annulation d'E/S 604

15. Gestion de l'alimentation 606
Gestionnaire d'alimentation 606
Schémas d'alimentation 609
16. Gestion du stockage 612
Terminologie du stockage 612
Types de stockage disque 612
Disques de base 612
Etats des disques et des volumes 613
Administration des disques 613
Utilitaire Gestion des disques 614
Mise à jour des informations de l'outil Gestion des disques 614
Console Ordinateur 614
DiskPart 615
Fsutil 615
ReadyBoost 615
ReadyDrive 616
Rechercher des erreurs sur les disques 616
Nettoyage de disque 617
Cacher des lecteurs de disque 618
Changer la lettre d'un lecteur 618
Scinder un disque en plusieurs lecteurs 619
Compresser des disques 619
Compresser des répertoires ou des fichiers 619
Schéma de partition 620
Partitionnement MBR 620
Partitionnement GPT 620
Comparaison entre MBR et GPT 621
Visualisation du schéma de partition utilisé 621
Organisation sur disque de GPT 622
MBR hérité (ou protecteur) 622
Entête GPT 623
Table des partitions 624
Table des partitions secondaire 625
Entête secondaire 626
Volumes actifs, système et de démarrage 626
Organisation disque de MBR 627
Table des partitions 627
Identifiant de disque 628
17. Systèmes de fichiers 629
Terminologie relative aux systèmes de fichiers 629
Montage et démontage 629
Formats de système de fichiers Windows 629
ISO-9660 630
Joliet 630
UDF 631
FAT12, FAT16 et FAT32 631
exFAT 631
Unité d'allocation 631
Organisation disque de FAT32 632
Enregistrement de démarrage 633
FSInfo 635
Répertoire racine 636
Sauvegarde de l'environnement d'amorçage 637
NTFS 638
Noms Unicode 638

Versions de NTFS 638
Appropriation de fichier/dossier 638
Autorisations NTFS 638
Affichage des autorisations NTFS attribuées 639
Copie et déplacement de fichiers et de dossiers 640
Indexation 640
Compression NTFS 641
Compression en ligne de commandes 641
Flots de données multiples 642
Remappage dynamique des clusters défectueux 642
Fichiers clairsemés 643
Compression ZIP 643
Liens durs 643
Organisation de NTFS 644
Volumes 644
Clusters 644
Master File Table (MFT) 645
Références de fichier 647
Visualisation des données de volume 647
Attributs de fichier NTFS 648
Secteur d'amorçage NTFS 649
ID d'objet 651
Gestion des fichiers et répertoires 652
Common Log File System (CLFS) 652
Défragmentation 652
Extensions de noms de fichiers 652
Conversion d'un volume en NTFS 653
18. Démarrage et arrêt 654
BootRec 654
Dernière bonne configuration connue 654
Mode sans échec 654
Programmes utilisateur reconnaissant le mode sans échec 655
Processus de démarrage 655
Phase d'autotest d'allumage 656
Phase du chargeur de démarrage Windows 656
Initialisation du noyau 657
Gestionnaire de sessions (Smss) 658
Winlogon 658
Userinit 658
BCD 658
Modes de démarrage 661
Options d'amorçage 662
ReadyBoot 663
Modification du délai d'affichage 663
Modification temporaire de la séquence de démarrage 663
Afficher le contenu du BCD 663
Performances de démarrage 664
Causes courantes de mauvaises performances de démarrage 664
Débusquer les applications lancées au démarrage 665
Dossier Démarrage 665
Registre 665
Désactiver les programmes de démarrage via l'application Paramètres 665
Désactiver les programmes de démarrage via le gestionnaire de tâches 665
Options d'amorçage BCD 666
Arrêt 668

19. Gestionnaire de cache 670
Fonctionnalités clé du gestionnaire de cache 670
Cache système unifié et centralisé 670
Cache basé sur les blocs virtuels 670
Cohérence du cache 671
Cache et systèmes de fichiers récupérables 671
Écritures immédiates du cache 671
Désactivation de la mise en cache 671
Structures de données du cache 672
Structures de données globales du cache 672
Structures de données de niveau fichier du cache 673
Taille du cache 674
Taille virtuelle du cache 674
Taille du working set du cache 674
Lecture anticipée 674
Fast I/O 675
Interfaces avec le système de fichiers 676
Copie vers et depuis le cache 677
Mappage et épinglage 677
Cache et interfaces DMA 678
Cache avec écritures en différé 678
Lazy Writer 678
Désactivation des écritures en différé pour un fichier 678
Cache et mémoire virtuelle 678
20. Interceptions 680
Gestion des interruptions 680
Contrôleurs d'interruption x86 681
Contrôleurs d'interruption x64 682
Interruptions matérielles 682
Gestion des services système 682
Interruptions logicielles 692
DPC (Deferred Procedure Call) 693
Gestion des exceptions 694
Traitement structuré des exceptions (SEH, Structured Exception Handling) 696
21. Gestion du temps 697
Dates et heures DOS 697
22. Techniques de débogage 698
Pseudo registres 698
Débogage par affichage 698
Bug checks 698
Identification et résolution des incidents 706
Options de récupération 706
Scénarios 707
Dernier événement 707
Glossaire 708

Chapitre 1. Introduction

Ce livre invite le lecteur dans la nébuleuse des systèmes d'exploitation Microsoft Windows. Il s'adresse à un public de curieux : utilisateurs avancés, développeurs, administrateurs, professionnels expérimentés ; des profils hétéroclites, séduits par l'envers du miroir et les arcanes de Windows, qui souhaiteraient comprendre la façon dont ce système fonctionne. À travers ces pages, vous comprendrez les points clés de son architecture, les mécanismes fondamentaux qui pilotent ses fonctionnalités, ainsi que les structures et algorithmes qui régissent son fonctionnement interne. Nous espérons que vous aurez plaisir à lire cet ouvrage autant que nous avons eu à l'écrire, et vous souhaitons bon voyage dans ces coulisses de Windows.

À qui s'adresse ce livre ?

Ce livre a pour vocation d'apporter un éclairage exhaustif en ce qui concerne les systèmes d'exploitation de la gamme Microsoft Windows. Il propose ainsi un ensemble de concepts, d'outils, et de méthodes se rapportant aux dynamiques fonctionnelles (comprendre les technologies en place et les dispositifs en oeuvre) de ces systèmes, et se veut donc au fond être une opportunité offerte à toute personne intéressée à ce sujet - étant donné le caractère multidimensionnel du propos, *ces* sujets. Les profils concernés incluent tout particulièrement étudiants, enseignants et formateurs, utilisateurs avancés, concepteurs de logiciels, administrateurs et ingénieurs systèmes.

L'axe scientifique de cet ouvrage pourrait vous avoir laissé croire que son contenu était seulement pensé pour un public restreint. Si nous comprenons les raisons à l'origine de cette perspective, vraie en premier lieu, notez que de nombreux efforts ont été faits pour rendre ce livre accessible et agréable y compris à un plus large lectorat. Résultat, surtout, d'une attitude de curiosité envers les systèmes d'exploitation, c'est à cette même démarche que nous invitons le lecteur.

Structure de ce livre

Ce livre s'articule autour de dix-neuf chapitres :

- Le chapitre 1 (Concepts et outils) ouvre le bal sur une introduction à la terminologie et aux concepts clé qui seront employés tout au long de l'ouvrage. Il propose également un panorama des outils indispensables pour appréhender les subtilités et complexités du système.

- Le chapitre 2 (Architecture du système) offre une vue d'ensemble des caractéristiques structurelles de Windows. Il expose les grandes lignes de son architecture interne tout en analysant les interactions et relations entre ses principaux composants.

- Le chapitre 3 (Mécanismes système) fait un tour d'horizon des mécanismes fondamentaux qui contrôlent et pilotent les opérations essentielles du système.

- Le chapitre 4 (Mécanismes de gestion) aborde les outils et mécanismes mis à disposition pour l'administration et la configuration du système. Entrent dans cette catégorie la base de registre, les services Windows, ou encore WMI (Windows Management Instrumentation).

- Le chapitre 5 (*Gestionnaire d'objets*) traite de la manière dont Windows prend en charge le contrôle et le suivi des ressources systèmes.

- Le chapitre 6 (Synchronisation) explore les mécanismes permettant à Windows de garantir une exclusion mutuelle efficace entre différents acteurs du système, tels que les processeurs, processus et threads.

- Le chapitre 7 (*Processus et threads*) présente les notions fondamentales que sont processus et threads, et décrit les différents algorithmes qui y sont associés.

- Le chapitre 8 (Communication inter processus) explore les différents moyens qu'implémente Windows pour donner à plusieurs processus l'occasion de s'échanger des données entre eux.

- Le chapitre 9 (Gestion de la mémoire) offre une exploration détaillée des mécanismes par lesquels Windows implémente la mémoire virtuelle, assurant de la sorte une gestion optimale des espaces d'adressage alloués à chaque processus.
- Le chapitre 10 (Portable Executable) examine l'anatomie générale et fonctionnelle des modules Windows – tels que les programmes exécutables et bibliothèques – organisés suivant le format PE (Portable Executable).
- Le chapitre 11 (Sécurité) montre l'étendue de la palette de fonctionnalités qu'intègre Windows en vue de répondre aux exigences essentielles en matière de protection informatique, assurant la confidentialité, l'intégrité et la disponibilité des données et des systèmes.
- Le chapitre 12 (Réseau) passe en revue les fonctionnalités réseau qu'intègre Windows pour permettre l'échange d'informations et de services.
- Le chapitre 13 (Système d'E/S) explique de quelle façon Windows agit en tant que médiateur entre les composants matériels et logiciels (pilotes de périphérique et applications mode utilisateur) de l'ordinateur.
- Le chapitre 14 (Gestion de l'alimentation) explique de quelle façon Windows coordonne les événements liés à l'alimentation électrique, tout en orchestrant les transitions d'état nécessaires pour optimiser l'efficacité énergétique.
- Le chapitre 15 (Gestion du stockage) s'intéresse aux dispositifs classique en matière d'organisation des informations et traite de différents principes selon lesquels les fichiers sous Windows peuvent être manipulés (FAT, NTFS, etc.)
- Le chapitre 16 (Systèmes de fichiers) présente l'organisation et le fonctionnement interne de quelques systèmes de fichiers reconnus par Windows.
- Le chapitre 17 (Démarrage et arrêt) se concentre sur les méthodes qui se trouvent impliquées lors des phases de démarrage et d'arrêt du système, expliquant en détail les étapes et les interactions en jeu.
- Le chapitre 18 (Gestionnaire de cache) décrit les fonctionnalités de mise en cache que Windows met en oeuvre pour augmenter la réactivité de l'ordinateur.
- Le chapitre 19 (Interceptions) examine en profondeur les mécanismes d'interception de Windows, notamment la gestion des interruptions, des exceptions et des appels de services système, illustrant leur rôle dans le fonctionnement global du système.

À propos de la licence

Ce livre est publié sous licence CC BY-NC-ND. Concrètement, ces termes d'utilisation vous donnent le droit de copier, distribuer et communiquer le matériel par tous moyens et sous tous formats. Ils excluent par contre, sans permission expresse de l'auteur, la possibilité de modifier cette oeuvre ou de l'utiliser à des fins commerciales. (De manière générale, chacune des conditions parmi les licences Creative Commons peut être levée via autorisation du titulaire des droits.) Si ces critères peuvent au premier lieu sembler restrictifs, notez qu'il s'agit essentiellement de faire obstacle à la reproduction massive et incontrôlée de cet ouvrage. Pour plus d'information sur les licences Creative Commons, consultez le site http://creativecommons.org.

Outils et licences d'utilisation

Tout au long de ce livre, nous présentons différents logiciels que nous mettons à l'honneur à titre d'outils pédagogiques lors de toutes sortes de scénarios. Quelques-uns de ces utilitaires sont fournis avec le système d'exploitation (autrement dit intégrés à Windows, et de la sorte utilisables sans délai), tandis que d'autres sont des applications tierce partie disponibles sur Internet. Si nous nous efforçons de révéler toujours les adresses à partir desquels récupérer ces ressources, les liens web vont et viennent, leur pérennité n'est en conséquence pas garantie. De plus, une large majorité de ces logiciels implique que vous ayez lu et accepté pleinement les

termes et conditions d'utilisation qui s'y rapportent. Veillez aussi dans ce contexte à *comment* vous comptez donner lieu à cet accord. Par exemple, les outils affiliés à Microsoft/Sysinternals affichent dans une boîte de dialogue un contrat de licence utilisateur final (CLUF) lors de leur première utilisation. Vous devez dès lors non seulement accepter le CLUF, mais, si vous exécutez l'outil dans un fichier de commandes sur un système où il n'a jamais été employé (comprendre que l'image exécutable idoine n'a pas été sollicitée une seule fois à ce stade), le faire en incluant l'option appropriée sur la ligne de commande. Dans le cas contraire, une fenêtre modale surgit, empêchant par nature l'accès à toutes les autres fonctionnalités de l'application.

Assistance

Tous les efforts ont été faits rendre cet ouvrage aussi complet et précis qu'on puisse le désirer, aussi exempt d'erreurs que possible. Cependant, si vous rencontrez des fautes (d'ordre technique ou tout simplement de syntaxe), des inexactitudes ou si certains passages vous semblent confus, n'hésitez pas à les signaler en écrivant à l'auteur : *maillard.arnaud@gmail.com*.

Si une maladresse que vous souhaitez souligner relève d'un caractère strictement technique, nous vous serions reconnaissant de montrer une preuve de la solidité de vos propos, soit via indication de quelles pistes vous ont guidées (routines de programmation, attributs de structures de données, ou tout autre élément susceptible de conduire à l'identification de l'erreur), ou mieux encore, et de façon plus tangible, en joignant à vos commentaires les démonstrations (extraits de logs, captures d'écran, simulations d'une expérience dans le débogueur noyau, etc.) nécessaires à l'éclaircissement des faits.

Pour une assistance technique sur les logiciels Microsoft, consultez le site http://support.microsoft.com. Pour obtenir de l'assistance sur Microsoft Windows, allez à http://www.microsoft.com/windows. Vous pouvez également vous connecter directement à la base de connaissance Microsoft et saisir une question concernant un problème en allant sur http://support.microsoft.com/search/.

Précisions sur le glossaire

Comme la plupart des oeuvres touchant à des domaines techniques très spécifiques, ce livre se termine par un glossaire. D'ordre général un recueil de termes associés à leurs définitions, nous employons en ce qui nous concerne le glossaire de façon plus large, et profitons en l'occurrence de cet espace afin de clarifier certaines des traductions qui ont été faites - le vocabulaire entourant Microsoft Windows (et, du reste, les systèmes d'exploitation en général), né de la culture anglo-saxonne, supportant quelquefois assez mal le passage au français. Quelques exemples : objet *mutant*, routines *de diffusion*, verrous *rotatifs*, et bien d'autres.

En ce qui concerne la lecture

Pour faciliter la navigation à travers les nombreuses informations fournies dans ce livre, chaque chapitre a été conçu pour être consulté de manière autonome. Dans le même esprit, chaque section/segment tente, dans la mesure du possible, d'être en situation de suffisance vis à vis des autres.

Travaux en cours

Ce sur quoi votre attention est portée (et l'auteur l'espère, retenue) est un livre en cours de construction, dont la structure et le contenu (voire la présentation), s'ils continuent d'évoluer, n'en reste pas moins parfaitement utilisable. Un tel mode de diffusion a été choisi principalement pour deux raisons :

- **Talonner le développement de Windows** Si les mécanismes centraux dans toutes les versions de Windows se ressemblent en surface, ressorts et rouages du système peuvent différer, parfois de façon légère, quelquefois radicalement. Dans la même veine, toute sortie majeure de systèmes Microsoft apportant son lot de nouveautés, il n'est pas concevable de traiter d'innovations au même degré que des technologies plus anciennes, pour lesquelles on a eu le temps de l'analyse. De variations anodines en bouleversements plus profonds (rupture avec l'existant), nous tenions néanmoins à couvrir l'histoire et l'actualité de Windows.

- **S’assurer de la qualité de l’ouvrage** Les portes d’un livre incomplet restant toujours ouvertes, nous espérons par ce biais améliorer l’implication des lecteurs dans la définition du manuscrit final, et favoriser sa conformité à leurs attentes pour, enfin, aboutir à un ouvrage de meilleure qualité. N’hésitez donc pas à user d’un droit de regard et contacter l’auteur.

Pour les raisons susmentionnées, un numéro de version accompagne ce livre, correspondant à un état donné de son évolution.

Chapitre 2. Concepts et outils

Dans ce chapitre, nous allons introduire les principes ontologiques essentiels des systèmes d'exploitation en général, et de Microsoft Windows en particulier. Nous verrons dans un premier temps les notions de base relatives aux systèmes d'exploitation ; ce qu'ils proposent, ce qu'ils font, et pourquoi ils existent sous les formes qu'on leur connaît. L'étude se poursuivra sur l'évolution de Windows, dont le noyau et coeur opérationnel, s'il remonte aux années quatre-vingt, occupe depuis lors une place centrale dans le paysage informatique contemporain. Nous présenterons ensuite les concepts et termes fondamentaux de Microsoft Windows qui seront utilisés dans tout ce livre ; l'occasion de s'intéresser aux processus et aux threads, de découvrir les objets, voir comment utiliser les interfaces de programmation applicative (API), découvrir la base de registre, etc. Nous présenterons également les outils rendant possible l'exploration des coulisses du système, tels le débogueur noyau, la console Performances et d'autres utilitaires clé, tous la source de précieux enseignements sur les mécanismes internes de Windows.

Veillez à connaitre tout ce qui est décrit dans ce chapitre pour pouvoir comprendre le reste de cet ouvrage.

Systèmes d'exploitation

Windows étant une incarnation d'un système d'exploitation parmi d'autres - du reste ancrée dans le temps, par le biais des *versions* du logiciel, il nous a paru opportun avant de consacrer toute la lumière au thème central de cet ouvrage, d'accorder une place, quoique modeste, sur les processus par lesquels de tels logiciels étaient conçus, implantés et gérés. À titre informatif, notez que les notions passées en revue le sont dans une optique générale, déconnectée de tout système particulier - elles restent bien évidement valides dans le contexte de Microsoft Windows.

Ce qu'est un noyau

Pièce maîtresse dans le fonctionnement d'une grande majorité de systèmes d'exploitation, le noyau gère les ressources informatiques de base, incluant le matériel, le logiciel et les données, et ressemble à cet égard à un chef d'orchestre, dont le rôle est la coordination harmonieuse des éléments en place.

En tant que partie du système d'exploitation (il est lui-même un logiciel), le noyau fournit des mécanismes d'abstraction du matériel, notamment de la mémoire, du (ou des) processeur(s), et des échanges d'informations entre logiciels et périphériques matériels.

Chaque noyau de système d'exploitation a un rôle et un usage. Ainsi, certains noyaux sont conçus dans l'optique de pouvoir fonctionner sur différentes plateformes matérielles, quelques-uns pour être particulièrement robustes, d'autres pour avoir des performances élevées, d'autres encore pour combiner, moyennant certains compromis, les trois. En parallèle, un noyau n'a d'existence que sur une architecture machine donnée, laquelle est la spécification fonctionnelle d'un processeur (jeu d'instructions, ensembles de registres visibles, organisation de la mémoire, etc.). C'est au final les capacités de l'architecture sous-jacente qui déterminent ce que peut faire le noyau, à lui de construire par dessus elle un environnement concret.

L'un des aspects les plus importants des noyaux de système d'exploitation est la réalisation du concept d'abstraction, qui vise à représenter de manière commode (et surtout commune), la diversité des approches et des matériels. Cette façon de faire est avantageuse en plusieurs points : elle isole le matériel des applications (une application ne s'adresse au matériel qu'indirectement et sous l'aval du noyau) et les applications du matériel (les applications ne sont pas au fait du fonctionnement interne de la machine).

Différents types de noyaux

Un aspect incroyable des noyaux de système d'exploitation est la diversité des approches pour les concevoir. Il existe ainsi toutes sortes de noyaux, plus ou moins spécialisés. Certains sont spécifiques à une architecture, d'autres sont généralistes ; certains sont destinés aux ordinateurs personnels, d'autres aux serveurs, aux terminaux nomades ou aux consoles de jeux. L'ensemble de ces noyaux peut être divisé en deux démarches

opposées d'architectures logicielles, les noyaux monolithiques et les micronoyaux, chacune définissant la manière d'aborder les fondamentaux du système.

- *Noyaux monolithiques non modulaires* Dans un noyau monolithique non modulaire, les fonctions et extensions (pilotes) du système forment un seul bloc binaire généré à la compilation. De par la simplicité de leur concept, les noyaux monolithiques ont été les premiers à être développés et mis en œuvre.

- *Noyaux monolithiques modulaires* Dans un noyau monolithique modulaire, seules les fonctions fondamentales du noyau sont gardées dans un bloc monolithique. Les pilotes de périphériques sont séparés sous la forme de module qui pourront être chargés à la demande.

- *Systèmes à micro-noyaux* Les systèmes à micronoyaux cherchent à minimiser les fonctionnalités dépendantes du noyau en plaçant la plus grande partie des services du système d'exploitation à l'extérieur de ce noyau, c'est-à-dire dans l'espace utilisateur. Ces fonctionnalités sont alors fournies par des processus indépendants et autonome, ce dans l'optique de limiter - en théorie – l'impact des dysfonctionnements potentiels.

Ce que propose un système d'exploitation

Un système d'exploitation (OS, *Operating System*) est une collection de programmes qui tient le rôle d'intermédiaire entre l'utilisateur d'un ordinateur et ses programmes d'une part, et le matériel de l'ordinateur d'autre part.

Malgré les différences de point de vue, de forme, de taille et de type, un système informatique peut être divisé grossièrement en quatre composants : le matériel, le système d'exploitation, les programmes applicatifs et les utilisateurs. Les programmes applicatifs - tels que les traitements de texte, les jeux vidéo, les tableurs et les navigateurs internet - définissent les mécanismes adéquats pour résoudre les problèmes informatiques des utilisateurs. Le matériel, composé des processeurs qui exécutent les instructions, de la mémoire centrale qui contient les données et les instructions à exécuter, de la mémoire secondaire qui sauvegarde les informations, et des périphériques d'entrées/sorties (clavier, souris, écran, etc.) pour introduire ou récupérer des informations, fournit les ressources informatiques de base. Le système d'exploitation contrôle et coordonne l'utilisation de ces ressources parmi les divers programmes applicatifs pour les divers utilisateurs. Il reçoit à ce titre de la part des programmes des demandes d'utilisation des capacités de l'ordinateur - capacité de stockage des mémoires (mémoire de masse et mémoire volatile), capacité de traitement de l'unité centrale (*central processing unit* ou processeur), capacité d'utilisation des périphériques connectés à la machine (dispositifs d'entrée/sortie).

Fonctions d'un système d'exploitation

Les systèmes d'exploitation modernes sont constitués de centaines de milliers, voire de millions de lignes de code. Ils ont comme rôle primordial la gestion des ressources informatiques de base (processeur, mémoire et périphériques) pour divers utilisateurs.

- *Gestion du processeur* Le système d'exploitation contrôle et coordonne l'utilisation du processeur parmi les applications et les utilisateurs, ce sur la base d'un algorithme d'ordonnancement permettant à toute tâche de s'exécuter à un moment ou un autre.

- *Gestion de la mémoire vive* Le système d'exploitation est chargé de gérer l'espace mémoire alloué à chaque application, et partant, chaque utilisateur. Comme la mémoire vive est généralement trop petite pour contenir toutes les données et tous les programmes, le système d'exploitation peut créer une zone mémoire auxiliaire sur le disque dur, et réaliser ce faisant le principe de mémoire virtuelle, qui permet de faire fonctionner des applications nécessitant plus de mémoire qu'il en existe de physiquement disponible sur le système. En contrepartie, cette mémoire est plus lente.

- *Gestion des processus* Le système d'exploitation est chargé du déroulement des applications, gérant à ce titre et de façon optimale les ressources nécessaires à leur bon fonctionnement. Il organise et rend visible en

parallèle nombre de dispositifs permettant aux programmeurs de concevoir des logiciels, et aux utilisateurs d'interagir avec ces derniers, par exemple fin à une application ne répondant plus correctement.

- *Gestion des droits* Le système d'exploitation veille à la sécurité des programmes et à la confidentialité des données. Il empêche les applications de lui nuire, de compromettre d'autres applications, et garantit en sus que les ressources ne sont utilisées que par les programmes et utilisateurs possédant les droits adéquats.
- *Gestion des fichiers* Le système d'exploitation permet l'enregistrement sur support de stockage (disque dur, SSD, CD-ROM, clé USB, disquette, etc.) de collections d'informations numériques, ce qui laisse la possibilité de traiter et de conserver des quantités importantes de données, ainsi que de les partager entre plusieurs programmes informatiques.
- *Gestion des entrées-sorties* Le système d'exploitation contrôle les périphériques associés à l'ordinateur. Il régule l'accès des programmes aux ressources matérielles par l'intermédiaire des pilotes, et fait la gestion des flux de données en entrée comme en sortie.

Exemples de systèmes d'exploitation

Dans le secteur informatique, les systèmes d'exploitation les plus répandus sont Windows, dont Windows 10 est la déclinaison la plus récente ; Mac OS, pour les ordinateurs d'Apple, Linux et Unix. Pour les terminaux légers, on trouve Android, iOS et Windows Phone.

Mémoire virtuelle

Windows implémente un système de mémoire virtuelle qui donne à chaque processus l'illusion de disposer d'un très large espace d'adressage, et d'en être de surcroît le seul détenteur. En effet, pour un processus donné, tout se passe comme s'il possédait les adresses 0 à *x*, sachant que c'est dans cette configuration le système d'exploitation qui, par l'intermédiaire du matériel, se charge de faire une conversion des adresses virtuelles en adresses physiques. Par exemple, si l'on considère une adresse 32 bits, chaque processus se pense dépositaire de la mémoire depuis l'adresse 0 jusqu'à l'adresse 2^32-1, à quoi correspond un espace d'adressage maximal de 4 Go. C'est en définitive le rôle de la mémoire virtuelle de faire correspondre besoins réels en mémoire, adresses mémoire virtuelles utilisées par les processus et mémoire physique réelle.

La mémoire virtuelle fournit une vue logique de la mémoire éventuellement décorrélée de sa disposition physique. Applications utilisateur et code système référencent des adresses virtuelles. Lors de l'exécution, le gestionnaire de mémoire, épaulé à cette fin par l'unité matérielle de gestion mémoire (MMU, *Memory Management Unit*), traduit ou *mappe* les adresses virtuelles en adresses physiques, lesquelles contiennent effectivement les données.

Telle que la mettent en oeuvre les systèmes et les équipements actuels, la mémoire virtuelle est le plus souvent basée sur un mécanisme de pagination, cela afin d'en augmenter la taille théorique maximale.

Les bases fondamentales sur lesquelles s'appuie la pagination se présentent sous les formes que voici : la mémoire centrale est découpée en cadres de taille identique (par exemple 4 Ko). L'espace d'adressage virtuel est quant à lui divisé en unité appelées *pages*. Chaque segment de la mémoire en cours d'exécution réside alors dans un certain nombre de pages. Une page est un ensemble d'octets qui peut résider dans un cadre. Au fur et à mesure qu'un processus demande de la mémoire, une page lui est allouée. Le nombre de pages alloués à l'ensemble des processus est souvent supérieur au nombre de cadres de la mémoire physique. C'est en l'occurence le disque dur qui permet de stocker les pages allouées en sus du nombre de cadres, cela par le biais d'un fichier d'échange.

La plupart des machines ayant beaucoup moins de mémoire physique que la mémoire virtuelle totale utilisée par les processus en cours d'exécution, le gestionnaire de mémoire transfère ou *pagine* une partie du contenu de la mémoire vers le disque. Cela signifie qu'à tout moment, un certain nombre de pages de mémoire allouées aux processus résident en mémoire centrale, le reste dans la mémoire secondaire (autrement dit le fichier d'échange). Quand un thread accède à une adresse virtuelle qui a été paginée vers le disque, le gestionnaire de mémoire recharge les données depuis le disque vers la mémoire centrale. Ce processus est transparent pour les applications.

L'approche employée pour choisir les pages à remiser dans le fichier d'échange s'appuie le plus souvent sur le nombre d'accès à une page (ce qui en préfigure la nécessité) et/ou sa dernière date d'accès. L'objectif de cette stratégie est de minimiser les défauts de page qui, s'ils font partie du fonctionnement normal de la mémoire virtuelle, n'en restent pas moins un handicap pour la rapidité d'exécution de l'ensemble du système.

Ainsi que nous l'avons déjà mentionné, la transformation adresse virtuelle/adresse physique est du ressort de l'unité matérielle de gestion mémoire (MMU). Si cette dernière se présentait autrefois sous la forme d'une puce spécialisée prenant place entre le processeur et la mémoire, les processeurs actuels l'intègrent nativement, cela pour des raisons de performance.

La taille de l'espace d'adressage virtuel varie selon la plateforme matérielle et la version de Windows qui l'anime. Généralement, sur les systèmes x86 32 bits, l'espace d'adressage virtuel total a un maximum théorique de 4 Go, dont 2 sont dévolus aux processus du mode noyau, et 2 autres aux processus du mode utilisateur. Dans l'éventualité où la machine dispose de suffisamment de mémoire physique (1 Go minimum), cette configuration peut évoluer de façon à donner aux processus exécutant des programmes spécialement calibrés pour la possibilité d'utiliser 3 Go d'adressage privé, laissant ainsi 1 Go au système d'exploitation.

Si une large majorité des processus se satisfont pleinement des quantités de mémoire dont nous avons parlé jusqu'à présent, quelques-uns manifestent à cet égard de plus gros besoin, par exemple les applications du genre serveur de base de données ou logiciel d'édition vidéo. Pour résoudre ce problème sur les systèmes 32 bits, dont les limites architecturales peuvent en ce qui concerne ces cas se révéler un véritable frein, Windows fournit un mécanisme appelé AWE (Address Windowing Extension) qui permet à une application 32 bits de manipuler une capacité de mémoire supérieure aux 2 ou 3 Go disponibles par le biais des dispositifs d'adressage standards.

Les systèmes 64 bits ouvrent la voie à un espace d'adressage particulièrement volumineux, à savoir 16 Eo (2^64). Dans la pratique, toutefois, les versions actuelles de Windows 64 bits ne vont pas au delà de 8192 Go de mémoire adressable.

Indépendamment de la taille jusqu'à laquelle peut aller l'espace d'adressage, Windows alloue (sauf configuration particulière) la moitié de cet espace (en l'occurence la moitié inférieure) aux processus du mode utilisateur (par exemple, les applications), leur procurant de cette manière un volume de stockage privé individuel relativement confortable, et utilise l'autre moitié (la moitié supérieure, donc) pour les processus du mode noyau (par exemple, le système d'exploitation et les pilotes). Alors que les mappages de la moitié inférieure évoluent pour refléter l'espace d'adressage virtuel du processus qui est en cours d'exécution, les mappages de la moitié supérieure sont immuables, et de la sorte visibles depuis tous les processus.

Ordonnancement

La sélection dans le temps des processus pouvant accéder aux ressources de l'ordinateur est un problème dit d'ordonnancement. Nous présentons dans cette section le cas général, les besoins, et décrirons différents algorithmes mettant en œuvre la politique générale d'ordonnancement dans divers systèmes d'exploitation.

Objectifs d'un ordonnanceur

Si la politique générale du système d'exploitation peut en intégrer d'autres, les objectifs les plus communs en matière d'ordonnancement visent en ce sens une utilisation harmonieuse des ressources entre tous les processus parmi l'ensemble des utilisateurs. Cette optique a généralement trait à plusieurs facteurs :

- *Maximiser l'utilisation du processeur* le système doit faire en sorte que tout processus passe le moins de temps possible dans un état d'ordonnancement incompatible avec la possibilité d'être élu comme prochain (comprendre prochain processus à exécuter). Il doit pour ce faire être capable d'identifier les relations de dépendances entre les divers processus et, par exemple, prioriser ceux détenteurs d'une ressource afin que les attendeurs de la dite ressource puisse devenir des unités ordonnancables directement.

- *Présenter un temps de réponse acceptable* L'expérience utilisateur au niveau des délais, avec laquelle sera en majorité jugé le système, ne doit souffrir d'aucune quelconque pénalité venue de décisions

d'ordonnancement. Nombre de systèmes modernes atteignent cet objectif via un système de priorité assignant une plus haute aux threads faisant fonctionner des éléments graphiques.

- *Respecter l'équité entre les processus* Selon le critère d'ordonnancement utilisé, les processus gérés par l'ordonnancement doivent généralement être traités à égalité. S'il existe des priorités, le privilège se limitera à elles.

- *S'assurer de l'absence de famine* Les décisions issues de l'ordonnancement doivent autant que possible éviter de mettre les travaux gérés en famine, où un ou plus processus pourrait se voir refuser l'accès à une ressource pendant un temps indéterminé. L'algorithme doit ne pas créer ces situations, et encore savoir les éviter et comment les prévenir si elles ont lieu.

Critères d'ordonnancement

Un objectif majeur de toute stratégie d'ordonnancement est l'optimalité, établie via des algorithmes qui sont compromis entre sûreté et vivacité, et doivent en ce sens identifier quel processus conduira aux meilleures performances pour l'ensemble du système. Entrent pour ce faire en compte divers critères à l'importance relative variable. La liste qui suit passe en revue les critères les fréquemment utilisées.

- *Utilisation du processeur* Pourcentage de temps pendant lequel un processeur exécute un processus. L'importance de ce critère varie généralement en fonction du degré de partage du système.

- *Rendement (throughput)* Nombre de processus exécutés par unité de temps.

- *Temps de service (turnaround time)* est le temps qui s'écoule entre le moment où un processus devient prêt à s'exécuter et le moment où il finit de s'exécuter (temps d'accès mémoire + temps d'attente dans la file des processus éligibles + temps d'exécution dans l'unité centrale + temps d'attente + temps d'exécution dans les périphériques d'entrée/sortie). On utilise en général un temps moyen de service calculé pour tous les processus mis en jeu pendant une période d'observation donnée. Il est inversement proportionnel au rendement.

- *Temps de réponse (response time)* Temps qui s'écoule entre la soumission d'une requête et la première réponse obtenue. On utilise en général un temps moyen de réponse calculé pour tous les processus mis en jeu pendant une période d'observation donnée. Le temps de réponse est utile dans l'évaluation des systèmes de type transactionnel où il est intéressant de mesurer le temps qui s'écoule entre le moment où l'utilisateur formule une requête, et celui où la réponse lui parvient.

- *Equité* Degré auquel tous les processus reçoivent une chance égale de s'exécuter.

- *Priorités* Attribue un traitement préférentiel aux processus dont le niveau de priorité est supérieur.

- *Temps d'attente (waiting time)* temps passé dans la file des processus éligibles. On utilise en général un temps moyen d'attente calculé pour tous les processus mis en jeu pendant une période d'observation donnée. Mesurer la performance par le temps de rotation présente un inconvénient : Le temps de production du processus accroît le temps de service ; Le temps d'attente représente donc une mesure plus précise de la performance.

Notez que ces critères étant plus ou moins mutuellement exclusifs, les comparaisons des différents algorithmes se fait donc non sur toutes mais sur une sélection de ces mesures.

Concepts et terminologie

Versions de Windows

Bien que ce livre se concentre principalement sur les dernières versions de Windows (à tout le moins celles postérieures à Windows Vista, lequel amorce la plupart des processus de transition vers ce que l'on connaît

aujourd'hui), il est utile de revenir sur l'histoire des produits de la famille Microsoft Windows. Bon nombre d'aspects des systèmes de cette gamme ont en effet un héritage commun, à savoir Windows NT (NT pour *New technology*), conçu et développé par Microsoft à la fin des années 80, et dont on peut jusque dans les versions les plus récentes de Windows, percevoir l'écho.

Chaque version de Microsoft Windows est connue par le biais d'un nom, d'un numéro de version et d'un nom de code. Le nom de produit fait office de désignation générique (et surtout commode) diffusée auprès du grand public afin d'assurer au logiciel (voire à toute une gamme) une identité unique sur les marchés. Quelques exemples à cet égard : XP, Vista, 7, 10. Le numéro de version se rapporte à une numérotation interne mettant en évidence la version du noyau animant le système d'exploitation. Généralement, une nouvelle famille de système s'ouvre avec une évolution majeure du noyau, laquelle constitue sous cette perspective une réponse tangible aux besoins actuels du secteur. Enfin, un nom de code accompagne chaque produit avant la publication officielle, le temps pour Microsoft d'en annoncer un définitif.

Tableau 2.1. Versions de Windows

Nom de produit	Numéro de version interne	Nom de code	Année de sortie
Windows NT 3.1	3.1	n/a	Juillet 1993
Windows NT 3.5	3.5	Daytona	Septembre 1994
Windows NT 3.51	3.51	n/a	Mai 1995
Windows NT 4.0	4.0	SUR (Shell Update Release), Cairo	Juillet 1996
Windows 2000	5.0	n/a	Décembre 1999
Windows XP	5.1	Whistler	Aout 2001
Windows Server 2003	5.2	Whistler Server	Mars 2003
Windows Vista	6.0 (Build 6000)	Longhorn	Janvier 2007
Windows Server 2008	6.0 (Build 6001)	Longhorn Server	Mars 2008
Windows 7	6.1	Blackcomb, Vienna	Octobre 2009
Windows Server 2008 R2	6.1	-	Octobre 2009
Windows 8	6.2	Jupiter	Janvier 2012
Windows Server 2012	6.2	Windows Server 8	Aout 2012
Windows 8.1	6.3	Blue	Octobre 2013
Windows Server 2012 R2	6.3	Windows Server 8	Octobre 2013
Windows 10	10.0	Threshold, Redstone	Juillet 2015
Windows Server 2016	10.0	Redstone Server	Septembre 2016

Il existe six versions client majeures de Windows 10 : Windows 10 Famille, orientée pour un usage domestique, Windows 10 Professionnel, pour les petites et moyennes entreprises, Windows 10 Entreprise, pour les grands comptes (licences en volume), Windows 10 Education, pour les grandes écoles et universités, Windows 10 Professionnel Éducation, qui intègre des paramètres par défaut spécifiquement adaptés aux structures éducatives.

Il existe principalement trois moutures de Windows Server 2019 : Centre de données, Standard et Essentiel.

Si versions client et versions serveur de Windows partagent les mêmes fondations, y compris l'image du noyau, les bibliothèques HAL, les pilotes de périphériques et les DLL, elles diffèrent dans bon nombre de fonctionnalités, par exemple le nombre de processeurs autorisés, la quantité de mémoire physique prise en compte ou le nombre de connexions réseau concurrentes envisageables. En outre, certains composants des éditions serveur ne font pas partie de leurs homologues client, tels les services d'annuaire.

De manière générale, les systèmes serveur sont optimisés au niveau du débit en tant que serveurs d'application à hautes performances. La version client, même si elle peut présenter des fonctionnalités serveur, est conçu pour minimiser le temps de réponse du bureau interactif, lequel influe fortement sur la perception de de l'utilisateur. Ainsi, le type du produit conditionne un certain nombre de décisions d'allocation de ressources prises au démarrage du système, par exemple la taille et le nombre des tas (ou pools) du système d'exploitation, le nombre

de threads système auxiliaires internes, ou encore la taille du cache des données système. En outre, certaines décisions stratégiques concernant l'ordonnancement ne sont pas les mêmes entre éditions serveur et client, par exemple la façon dont le gestionnaire de mémoire équilibre les demandes d'allocation entre le système d'exploitation et les processus, la longueur par défaut de la tranche de temps, ou quantum, allouée aux threads (avec une valeur de quantum plus élevée sur les systèmes serveur, les applications ont plus de de chances de traiter une requête dans le temps imparti). Sauf mention explicite du contraire, tout dans cet ouvrage s'applique à la fois aux versions client et aux version serveur.

Fonction	Service
GetProductInfo	RtlGetProductInfo
GetVersionEx	RtlGetVersion
	RtlGetNtVersionNumbers
	RtlGetNtProductType
VerifyVersionInfo	RtlVerifyVersionInfo
IsWindowsServer	

Element	Lieu	Autres references
OSBuildNumber	PEB	RtlGetNtVersionNumbers
OSMajorVersion	PEB	RtlGetNtVersionNumbers
OSMinorVersion	PEB	RtlGetNtVersionNumbers
NtBuildNumber	KUSER_SHARED_DATA	
NtMajorVersion	KUSER_SHARED_DATA	
NtMinorVersion	KUSER_SHARED_DATA	
NtProductType	KUSER_SHARED_DATA	RtlGetNtProductType
SuiteMask	KUSER_SHARED_DATA	RtlGetSuiteMask

Clé
HKLM\System\CurrentControlSet\Control\ProductOptions\ProductType
HKLM\Software\Microsoft\Windows NT\CurrentVersion\CurrentMajorVersionNumber
HKLM\Software\Microsoft\Windows NT\CurrentVersion\CurrentMinorVersionNumber
HKLM\Software\Microsoft\Windows NT\CurrentVersion\CurrentBuild
HKLM\Software\Microsoft\Windows NT\CurrentVersion\CurrentBuildNumber

Structure	Autres références
OSVERSIONINFOEX	VerifyVersionInfo

Examen de la version courante du système d'exploitation

Windows intègre nativement plusieurs applications et commandes par l'intermédiaire desquelles utilisateurs et administrateurs peuvent facilement se rendre compte de la version du système d'exploitation utilisé sur la station de travail. (Pour une vue programmatique du sujet, voir chapitre Processus et threads.)

Si vous ne savez pas précisément quelle version de Microsoft Windows votre machine exécute, depuis une invite de commande ou le contrôle Exécuter, saisissez la commande winver puis validez. Une fenêtre devrait alors apparaître, affichant la version du système, par exemple 7, 8 ou 10, ainsi que d'autres détails.

Pour voir les détails de version, procédez comme suit : (1) cliquez sur Démarrer puis cliquez sur Paramètres, (2) dans la fenêtre des paramètres, cliquez sur Système, (3) à partir des onglets qui se trouvent sur la gauche, cliquez sur Informations système. Vous verrez à ce moment quelle version de Windows vous utilisez, votre numéro de version (par exemple 1511) et le type de système (32-bit ou 64-bit).

La commande ver constitue une méthode parmi les plus pratiques dans le but d'examiner les informations de version de Windows. Elle ne requiert aucun argument et produit un et un seul type de résultat, ce qui se prête particulièrement bien à un usage dans des scripts d'administration. Les précisions apportées par

ladite commande incluent le nom, les numéros majeurs et mineur, et le numéro de fabrication (build) du système d'exploitation. Pour voir concrètement ces données, saisissez simplement ver depuis une fenêtre d'invite de commandes de commande. Vous devriez à ce moment voir quelque chose de similaire à ce qui suit. Par comparaison avec les autres utilitaires que nous avons vus jusqu'ici, c'est sans doute WMI qui permet d'afficher le plus d'informations sur le système. Consultez dans ce cas les propriétés Caption, CSDVersion, OSArchitecture, et Version de la classe OS.

Une grande majorité des informations en lien avec la version de Windows employée sur le poste sont regroupés sous la clé HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion.

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion
    SystemRoot    REG_SZ    C:\WINDOWS
    BuildBranch    REG_SZ    th2_release
    CurrentBuild    REG_SZ    10586
    CurrentMajorVersionNumber    REG_DWORD    0xa
    CurrentMinorVersionNumber    REG_DWORD    0x0
    CurrentType    REG_SZ    Multiprocessor Free
    CurrentVersion    REG_SZ    6.3
    EditionID    REG_SZ    Core
    InstallationType    REG_SZ    Client
    InstallDate    REG_DWORD    0x566e53eb
    ProductName    REG_SZ    Windows 10 Home
    ReleaseId    REG_SZ    1511
    SoftwareType    REG_SZ    System
    UBR    REG_DWORD    0x24d
    PathName    REG_SZ    C:\WINDOWS
    Customizations    REG_SZ    ModernApps
    BuildLabEx    REG_SZ    10586.589.amd64fre.th2_release.160906-1759
    BuildLab    REG_SZ    10586.th2_release.160906-1759
    ProductId    REG_SZ    00326-10000-00000-AA115
    CurrentBuildNumber    REG_SZ    10586
    BuildGUID    REG_SZ    ffffffff-ffff-ffff-ffff-ffffffffffff
    RegisteredOrganization    REG_SZ    Hewlett-Packard Company
    RegisteredOwner    REG_SZ    arnaud
    InstallTime    REG_QWORD    0x1d13630855020eb
    .
    .
    .
```

La liste qui suit donne la signification des valeurs les plus importantes de cette clé.

Tableau 2.2. Valeurs sous CurrentVersion

Valeur	Description
BuildLab	Numéro de version complet de Windows
CurrentBuildNumber	Numéro de version de Windows
CurrentMajorVersionNumber	Numéro de version primaire
CurrentMinorVersionNumber	Numéro de version secondaire
CurrentVersion	Numéro de version du noyau
DigitalProductId	Numéro d'identification de Windows
InstallDate	Date à laquelle Windows a été installé (donnée en temps Unix, soit le nombre de secondes écoulées depuis le 1er janvier 1970).
InstallTime	Date à laquelle Windows a été installé (donnée, au contraire de la valeur InstallDate, en temps Windows).
PathName	Répertoire d'installation du système.
ProductId	ID de produit
ProductName	Nom du système d'exploitation tel qu'il a été distribué de façon commerciale
RegisteredOwner	Nom du propriétaire enregistré de la licence Windows, tel que spécifié lors du processus d'installation du système.

ReleaseId ID	ID de publication, Cette valeur, visible seulement à partir de Windows 10 et à quoi semble correspondre une date de sortie prévue, est généralement un code à 4 chiffres, dont les deux premiers donnent une année et les deux autres un mois.

Fonctions, services et routines

Comme pour certains termes du langage ordinaire, quelques-uns propres à l'écosystème Windows voient leur signification varier en fonction du contexte. Ainsi, hors d'un cadre précisément situé, un *service* peut désigner potentiellement une routine incorporée du système d'exploitation, un pilote de périphériques, ou un processus serveur. Face à ce problème, et afin que nous nous entendions dès le départ sur leur sens, la liste suivante chercher à préciser la définition de plusieurs termes utilisés dans l'ouvrage. (Notez que ces éclaircissements sont suggérés ici uniquement afin de parvenir à une bonne entente sur les concepts communs. S'ils ont tous un caractère simple et pratique, aucun ne devrait être considéré comme universel.)

- *Fonctions d'API Windows* Sous-routines appellables depuis le mode utilisateur et incorporées au sein de diverses bibliothèques de liens dynamiques (DLL). Ces fonctions font partie intégrante du système d'exploitation Windows et permettent d'effectuer des tâches lorsqu'il s'avère difficile d'écrire des procédures équivalentes. Par exemple, Windows propose les fonctions nommées *CreateProcess* et *ExitProcess*, affiliées au module Kernel32.dll et avec lesquelles, respectivement, donner naissance et mettre fin à un processus. La plupart des fonctions de l'API Windows bénéficient d'une documentation complète incluant les objectifs, les paramètres d'appel et quelques remarques générales d'utilisation.

- *Services système natifs* Sous-routines chargées de faire la liaison entre des fonctions du mode utilisateur et d'autres fonctions du mode noyau. L'invocation de telles routines tient à la fois de l'appel de procédure et du traitement d'interruption, dans la mesure où cela se traduit par un changement du mode d'exécution du processeur (bascule du mode mode utilisateur vers le mode noyau, et inversement à l'issue de l'exécution), et par un branchement à l'adresse d'une fonction implémentée au niveau de l'espace système. Ainsi, *NtCreateProcess* est le service système que la fonction Windows *CreateProcess* appelle pour créer un processus. Ces pièces de logiciel ayant un comportement susceptible d'être modifié à chaque itération de Windows, il est recommandé aux concepteurs de logiciels d'éviter leur utilisation, même si de récents efforts à ce niveau tendent à montrer la volonté de Microsoft de rendre publique la documentation idoine.

- *Fonctions (ou routines, ou interfaces) noyau* Sous-routines appelables depuis le mode noyau qui mettent en oeuvre l'interface de programmation pour pilotes de périphériques et autres composants bas niveau. Par exemple, *IoCreateDevice* est la routine que les pilotes de périphérique utilisent pour créer ou ouvrir un objet périphérique représentant un équipement physique ou logique, *ExAllocatePoolWithTag* celle qu'ils appellent pour allouer de la mémoire à partir des tas système de Windows.

- *Routines de support noyau* Sous-routines endogènes internes au système d'exploitation. Par exemple, *KeReadStateMutant*, via laquelle déterminer l'état (signalé ou non) d'un objet mutant (nom interne du mutex, à quelques variations techniques près), ou *KeSetIdealProcessorThread*, que le système (et normalement lui seul) emploie pour définir le processeur idéal d'un thread.

- *Services* Windows Processus démarrés par le Gestionnaire de contrôle des services (SCM, *Service Control Manager*) effectuant diverses tâches pour le compte des utilisateurs de l'ordinateur ou du domaine. (La base de Registre définit les pilotes de périphérique Windows comme étant des services, ce que nous ne ferons pas dans cet ouvrage.) De telles applications fonctionnent en arrière-plan et n'interagissent généralement pas avec l'utilisateur. Les exemples de services Windows incluent le Planificateur de tâches, le gestionnaire automatique de réseaux sans fil, etc.

- *DLL (Dynamic Link Library)* Collection de fonctions liées entre elles sous la forme d'un fichier binaire pouvant être chargé dynamiquement, ledit fichier étant relié à l'application lorsque celle-ci en a besoin. Les composants et applications Windows en mode utilisateur font un usage intensif des DLL, d'une part pour les interfaces de programmation que ces bibliothèques mettent en oeuvre, mais également pour de nombreuses extensions : boîtes de dialogue, widgets, polices de caractères, etc. Citons, par exemple, Kernel32.dll, qui

héberge une bonne partie des fonctionnalités mises à disposition par l'API Windows, ou User32.dll, qui contient des fonctions associées à l'interface utilisateur. L'avantage des DLL par rapport aux bibliothèques statiques, dont le code est incorporé à chaque programme, est que les applications peuvent se partager les DLL, Windows faisant en sorte que le code d'une DLL ne soit chargé qu'une seule fois en mémoire même s'il est référencé par plusieurs applications.

Documentation

C'est en qualité de société éditrice que Microsoft s'emploie à documenter la plupart des services embarqués par son système d'exploitation. À ce seul niveau, la principale différence entre les fonctions publiques et celles privées est palpable avec l'engagement de Microsoft à fournir un écosystème dont les conditions d'utilisation sont pérennes, où la présence de documentation d'une fonction garantit à cette dernière une existence dans les versions futures de Windows. Cette discipline d'ingénierie, asseyant la confiance des développeurs en la portabilité de ce qu'ils conçoivent, est sans doute l'une des raisons du succès de Windows auprès des informaticiens créateurs de logiciels.

Documentation logicielle. La documentation logicielle est un texte écrit dont le but est d'expliquer comment le logiciel fonctionne, et/ou comment on doit l'employer. Si le terme peut avoir des significations différentes pour des personnes de différents profils, nous l'utilisons dans cet ouvrage dans le contexte le plus répandu. Par conséquent, nous disons d'une fonction qu'elle est documentée dans la mesure où un support adéquat est disponible auprès la société éditrice, ici en l'occurrence Microsoft. Toute autre source est disqualifiée. De plus, Windows étant un logiciel à source fermée, les détails d'implémentation ne sont, en principe, pas visible.

Microsoft rend visible la documentation des structures de données et des interfaces Windows sur le site web Microsoft Developer Network (MSDN). Il est vivement recommandé de s'en tenir au contenu de ce média, dont l'ensemble des éléments décrits bénéficie d'un support à long terme. Les API non documentées peuvent disparaître, changer de nom, voire de comportement, y compris dans les évolutions mineures de la même version du système (service pack). Hormis de rares situations, vous ne devriez considérer ces fonctions autrement que pour un usage pédagogique. Pour autant, cela ne veut pas dire qu'elles ne devraient pas susciter l'intérêt, du fait notamment, pour la plupart, de contenir les détails d'implémentation internes de Windows. Ce livre tend même à prouver que la connaissance de certains de ces détails est souvent important lors des phases de réalisation (programmation), d'analyse (retro ingénierie), ou de résolution des dysfonctionnement (débogage) de vos propres programmes.

En plus d'héberger les informations de l'API Windows, le site MSDN contient également un nombre important d'écrits sur les technologies du système d'exploitation. Il accueille par exemple une catégorie spéciale de documents, appelée Base de connaissances (KB, Knowledge Base), laquelle regroupe des articles publiés par l'équipe de support client Microsoft pour proposer des solutions de contournement à des problèmes connus.

Processus, threads et autres unités fonctionnelles

Les unités de base dans les systèmes à temps partagé modernes sont les programmes, les *processus*, les *threads* et les *fibres*. A cela s'ajoute, mais concerne Windows seulement, la notion de *job*.

Au plus haut niveau d'abstraction, un processus est constitué d'un programme en exécution ainsi que de son contexte. Chaque processus est de la sorte plus que le code du programme dont il résulte, et contient également l'activité courante, représentée par le contenu des registres du processeur, les données, et l'environnement tels que piles, pointeur de pile, valeur des variables internes, emplacement de lecture dans un fichier, etc.

Un *processus* est concrètement une zone mémoire contenant des ressources, parmi lesquelles :

- Un *espace d'adressage virtuel privé*, qui est un ensemble d'adresses mémoire virtuelles utilisables par le processus
- Un programme exécutable, hébergeant le code machine exécutable et le jeu de données nécessaires à ces opérations, qui est mappé dans l'espace d'adressage virtuel du processus

- Une liste de handles ouverts sur différentes ressources système, telles que fichiers, ports de communication, primitives de synchronisation, etc.
- Un contexte de sécurité, qui identifie l'utilisateur, les groupes de sécurité et les privilèges associés au processus
- Des informations d'identification, qui permettent d'identifier sans ambiguïté un processus sur le système, et de le désigner par son nom ou au moyen d'un numéro unique (PID, process identifier)
- Au moins un thread

Sous un jour plus large que celui structurel, un processus est une instance d'un programme en cours d'exécution. Tout système d'exploitation moderne étant multitâche, capable à ce titre d'entrelacer l'exécution de plusieurs processus en même temps (et surtout de le faire de façon harmonieuse), il est possible d'exécuter un nombre virtuellement infini de processus sur un même processeur.

Un processus peut créer un ou plusieurs processus qui, à leur tour, peuvent créer d'autre processus, le tout donnant naissance à structure arborescente. À l'exception, du reste particulièrement notable, du premier, initié par le système d'exploitation, tous les processus descendent directement ou indirectement d'un autre. Par analogie aux liens intrafamiliaux humains, on utilise quand il est besoin de les distinguer les notions de processus parent (ou père) et de processus enfant (ou fils).

Plusieurs processus peuvent exécuter parallèlement le même programme - ce que se traduit sous la forme d'un corollaire par le fait qu'un même programme peut être sollicité à maintes reprises en donnant lieu chaque fois à un processus différent.

Un *programme* est une suite d'instructions enregistrées au niveau de la mémoire auxiliaire, usuellement un disque dur. On désigne par ce biais généralement le fichier en langage machine obtenu après compilation (parlant auquel cas aussi de programme exécutable ou de binaire), mais aussi potentiellement le fichier source écrit dans un langage donné, par exemple un programme C, C++, Java. Si programmes et processus se ressemblent en surface, ils sont fondamentalement différents. Un programme est une entité statique qui *décrit* un traitement (une suite d'instructions), alors qu'un processus est une entité dynamique qui *réalise* un traitement. Un programme existe donc en dehors de toute utilisation. A l'inverse, un processus doit son existence à un programme, dont il est le direct représentant de la prise en charge par le système d'exploitation.

Une *application* est constituée d'un ou plusieurs processus coopérants. Typiquement, un éditeur de texte (*Bloc-notes* par exemple), un navigateur web (*Edge* ou *Firefox*), un lecteur multimédia (*VLC*), un jeu vidéo (*Dark Souls*), sont des applications. Vous pouvez voir toutes les applications et tous les processus en cours par le biais du *gestionnaire des taches* (*task manager*). Il est courant d'avoir une vingtaine de processus en même temps, même si vous avez ouvert un petit nombre d'applications.

Un processus possède un ou plusieurs *unités d'exécution* appelée(s) *threads*. Un *thread* est l'entité de base dont Windows coordonne l'exécution. Au plus haut niveau d'abstraction, un thread est similaire à un processus dans la mesure où tout deux représentent l'exécution d'un ensemble d'instructions du langage machine d'un processeur. Toutefois, là où chaque processus possède sa propre mémoire virtuelle, les threads d'un même processus se partagent l'espace d'adressage virtuel du processus auxquels ils appartiennent.

Sous un jour essentiellement pratique, la notion de thread n'a aucune signification matérielle ou physique au niveau du processeur, mais se concrétise par un ensemble de données dont il se dégage principalement deux grands thèmes : le contexte d'exécution et l'état. Un contexte d'exécution se compose d'un sous-ensemble des registres du processeur dont la sauvegarde est nécessaire permettre la suspension et la reprise ultérieure d'un thread donnée. À chaque thread est également associé un état. Les différents états possibles, de même que leur signification respective, sont propres à chaque système d'exploitation, mais incluent généralement les options Prêt pour l'exécution (le thread pourrait occuper le processeur si un autre ne le faisait pas déjà), En cours d'exécution (le thread s'exécute), Bloqué (le thread attend sur la disponibilité d'une ressource matérielle ou logique) et Terminé (le thread a terminé son exécution et ne devrait par tarder à être démantelé par le système).

Un thread ne peut appartenir qu'à un processus. Quand un processus commence, le système d'exploitation lui associe automatiquement un thread, appelé auquel cas thread principal ou thread primaire (*main thread* ou *primary thread* en anglais). Dans un processus Windows, un thread peut créer d'autres threads. Il n'y a pas de valeur maximale pour le nombre de threads par processus, une limitation naturelle s'établissant toutefois via la dépendance collective des threads via à vis des ressources mémoire disponibles.

Windows étend l'approche préemptive à base de priorités régissant l'ordonnancement des threads en y ajoutant la notion de *fibre*. Les fibres permettent à une application d'effectuer un traitement micro structurel de ses propres flux logistiques (multitâche coopératif). Dans ce scénario, chaque fibre doit pour commencer son exécution être manuellement sélectionnée, et pour se terminer passer le relais à une autre.

Un *job* permet à des groupes de processus d'être gérés et manipulées comme une entité unique. Les attributs et opérations typiques qu'un objet job peut effectuer ont des répercussions sur l'ensemble des processus associés au job. Par certains aspects, une telle approche vise à compenser l'absence dans les systèmes Windows d'une arborescence structurée de processus.

Gestionnaire des tâches

L'utilitaire prédéfini Gestionnaire des tâches (*Task Manager*) affiche des informations détaillées concernant les programmes, les processus et les services en cours d'exécution sur l'ordinateur. Il fournit des informations sur l'état du système en temps réel et couvre ainsi plusieurs aspects de sa performance.

Il peut également être utilisé pour définir la priorité des processus, de les arrêter, de basculer d'un programme à un autre, d'accéder aux fonctions de démarrage, de mise en veille et d'extinction du système, et de contrôler les informations de performances.

L'application Gestionnaire des tâches occupe les systèmes d'exploitation développés par Microsoft depuis Windows NT 4.0. Les versions précédentes incluaient une application nommée Task List dotée de beaucoup moins de fonctionnalités.

Vous pouvez démarrer le gestionnaire de plusieurs façons : (1) Appuyez sur le CTRL+ALT+SUPPR et cliquez sur le bouton Gestionnaire des tâches, (2) Ouvrez une fenêtre Executer (Windows+R) et tapez le nom du module sous-jacent au gestionnaire, à savoir taskmgr.exe. La vue initiale du gestionnaire montre seulement les processus auxquels sont associées des fenêtres visibles de premier niveau. Pour voir l'ensemble des processus, cliquez sur le bouton Plus de détails (More details). Dans ce mode de visualisation sont présentés les noms génériques des processus (par exemple, « Microsoft Word », ou « Hôte de la fenêtre de la console »), groupés en catégories : applications, processus d'arrière-plan et processus du système d'exploitation. Pour afficher les noms des images dont sont une instance ces processus, cliquez sur l'onglet Details. Pour afficher des informations supplémentaires à celles déjà présentes, faites un clic droit sur le bandeau supérieur, cliquez sur l'entrée Sélectionner les colonnes du menu contextuel qui vient d'apparaitre, puis sélectionnez les colonnes à afficher.

Le Gestionnaire des tâches se présente sous la forme d'une boite de dialogue à plusieurs onglets, chacun dévolu à une thématique.

- L'onglet Processus affiche une vue simplifiée des processus en cours d'exécution, triés par catégories : Applications (qui disposent d'une fenêtre active), Processus en arrière-plan (applications et services), Processus Windows (processus critiques et services hébergés par SvcHost).

- L'onglet performance offre une vue d'ensemble des données d'utilisation et de diagnostic se rapportant à divers équipements et fonctionnalités de l'ordinateur, par exemple l'unité centrale, les disques ou le réseau. Les informations montrées à ce niveau de l'interface peuvent se montrer extrêmement utile dès qu'il s'agit de détecter des anomalies, notamment une utilisation des ressources systèmes anormale, par exemple un nombre inhabituellement important d'E/S disque.

- L'onglet Détails liste les processus en cours d'exécution ainsi qu'un certain nombre de compteurs relatifs à la consommation de ressources.

- L'onglet Services liste les services présents sur le système ainsi que leur état démarré ou arrêté. Le lien Ouvrir les services en bas de la fenêtre procure un moyen d'accès rapide à l'outil d'administration Services, qui permet entre autres choses de modifier la configuration de démarrage des services.

- L'onglet Démarrage liste les applications démarrés automatiquement lors de la procédure d'amorçage de Windows.

Par défaut, le Gestionnaire des tâches met à jour ses données toutes les deux secondes. Pour augmenter la fréquence à deux fois par seconde, choisissez Affichage, Fréquence d'actualisation et changez la fréquence de Normale à Haute. Pour réduire la fréquence de mise à jour à une fois toutes les quatre secondes, optez pour le paramètre Basse.

Registre

Il est à peu près entendu si vous lisez ce livre que vous ayez déjà entendu parler, voire même mis les mains sous le capot, du Registre, lequel tient lieu de dépôt central pour moult paramètres relatifs à Windows et aux logiciels de l'ordinateur, incluant les données d'amorçage et de configuration du système et les préférences des utilisateurs et de chacune de leurs applications. En lien avec le fonctionnement global de la machine, le Registre est de ce fait un incontournable pour qui s'intéresse aux mécanismes internes de Windows.

Parmi la myriade d'informations stockées dans le registre, un certain nombre concerne l'état courant du système (par exemple les pilotes qui sont chargés, les ressources qu'ils utilisent, etc.) et quelques-unes font passerelle vers les compteurs de performance de Windows. (S'ils ne pas directement directement intégrés dans le Registre, on accède à ces compteurs via des fonctions de registre.)

Apparue avec la version 3 de Windows, le registre se présente alors comme une méthode alternative aux fichiers INI utilisés dans les systèmes prédécesseurs pour l'enregistrement des paramètres de configuration (voir encadré plus loin). Sous cette forme, par ailleurs rudimentaire, le registre sert exclusivement à associer une extension de fichier à un logiciel permettant l'édition des données impliquées par ce format. En 1993, avec la première version de NT, le registre est étendu de sorte à inclure un ensemble de clés hiérarchiques et de valeurs. Windows 95, en 1995, est la première version de Windows orientée complètement autour de ce dispositif, de même que toutes les versions qui suivent, incluant Windows 7, 8 et 10.

Le registre est organisé selon une structure hiérarchique de sous-arborescences contenant des clés, des sous-clés et des entrées. Un ensemble de clés, de sous-clés et de valeurs qui figure en haut de la hiérarchie du Registre est appelé une ruche. Il existe plusieurs ruches distinctes pour les informations relatives à l'ordinateur local, les profils d'utilisateurs, l'installation des logiciels et la sécurité. Les informations de la ruche système étant nécessaires au démarrage du système, le gestionnaire de registre, ou plus précisément le gestionnaire de configuration, est implémenté comme un composant de l'exécutif.

Dans la plupart des scénarios, administrateurs et utilisateurs interagissent avec le registre par le biais d'applications intermédiaires, par exemple l'éditeur graphique du Registre fourni avec Windows (regedit.exe) ou l'outil en ligne de commande reg.exe. De plus, quelques utilitaires d'administration standard permettent de voir ou de modifier bon nombre des paramètres de configuration stockés dans le registre. Dans de nombreux cas, les changements apportés par l'utilisateur ou l'administrateur au registre se font en réalité au travers d'un programme d'installation (application, correctifs), modifiant le registre de façon transparente, et n'autorisant pas une interaction directe avec des clés et des valeurs particulières.

Comme nous l'avons déjà évoqué, le registre joue un rôle très important dans le fonctionnement du système d'exploitation, et contient à ce titre une kyrielle d'informations en ce qui en concerne le comportement, les performances et les données internes. (Notez dès à présent que si vous décidez de modifier des paramètres du registre, faites-le avec précaution, une manipulation maladroite pouvant dégrader les performances ou, pire encore, empêcher le démarrage du système.) Tout au long de ce livre, vous trouverez des références à diverses clés du registre relatives à tel ou tel composant.

Pour plus de détails sur le registre et sa structure interne, reportez-vous au chapitre Mécanismes de gestion.

Notation hongroise

La notation hongroise est une norme d'écriture utilisée en programmation informatique afin de donner un éclairage complémentaire sur les variables et les fonctions d'un programme. Elle renvoie de la sorte au premier lieu, comme par ailleurs n'importe quelle autre convention de nommage, à la lisibilité du code source, et a fortiori sa compréhension ainsi que sa maintenance. Ce standard est notamment utilisé par Microsoft pour les API Windows, ainsi que les dérivés programmatiques qui s'ensuivent.

Généralités

L'idée fondatrice sur laquelle repose la notation hongroise est de désigner toute donnée (variables, constantes, fonctions, procédures, etc.) d'après un schéma qui en accentue la nature ou la fonction. (Ainsi que vous le verrez plus loin, la distinction entre les deux n'est pas toujours facile à établir avec certitude.) Elle instaure à cet égard un corpus méthodologique relativement complet, animé par un ensemble de raisonnements, de catégorisations, de classifications, bref de mesures, chargées de régir la mise en forme des noms dans un programme, avec comme ambition centrale d'en améliorer l'ancrage interprétatif.

Une des caractéristiques importantes, sinon la plus essentielle, de la notation hongroise est l'introduction de quelques symboles, tous chargés d'une sémantique particulière, chacun porteur d'un sens propre quant aux propriétés ou à l'usage d'un élément de programmation. Dans un tel contexte, le principe directeur consiste à faire précéder chaque identifiant logiciel (noms des éléments du programme) d'un préfixe reflétant son type, ainsi que potentiellement ses propriétés et sa localité (à savoir si la variable est de nature globale ou locale). Ce préfixe est composé d'une plusieurs lettres minuscules, chacune portée par un aspect descriptif : i pour un entier (int), dw pour un double mot (double word), etc. Les mots qui composent le nom de la variable seront en minuscules avec leur première lettre en majuscule.

On distingue en principe deux types de notation hongroise, chacun orienté par un axe de compréhension bien défini : la notation hongroise Apps, d'un côté, vise à mettre en avant l'usage des éléments qu'elle englobe ; la notation hongroise Systems, d'un autre côté, a pour but d'en souligner le type. Notez que si ces noms semblent cerner plusieurs choses distinctes, la notation hongroise Apps est essentiellement un calque des intentions originelles de l'auteur, l'étiquette étant ici affaire de conventions. Pour complexifier encore davantage la situation, un discours sur la notation hongroise (sans mention du contexte ou de la forme visée) peut faire référence à la notation Apps, à la notation Systems, ou aux deux.

Sans être le seul standard du genre, et sans non plus être exempte de défauts, la notation hongroise est sans doute l'une de celles que vous rencontrerez le plus souvent lors de la lecture de programmes conçus pour Windows.

Nommage des identifiants

Une question à laquelle est confronté tout concepteur de logiciel en face de la nécessité d'un nouvel identifiant porte sur le choix d'un nom pertinent et utile. Dans l'idéal, trouver des intitulés reflétant la signification des contenus qu'ils enveloppent est souvent recommandé, et d'ordinaire une bonne habitude à perpétuer. Ce point entraine en général à considérer les facteurs suivants :

- La valeur mnémonique du nom, de sorte que la personne l'ayant choisi puisse s'en souvenir avec aisance.
- Le potentiel suggestif, de manière à ce que d'autres puissent tirer de la seule lecture de ce nom des observations marquantes, sinon des informations intéressantes.
- La cohérence, de façon à ce que ce que les noms ayant des caractéristiques communes puissent se ressembler les uns entre les autres.
- La rapidité de décision, de sorte à limiter le temps de réflexion induit pour la mise en forme d'un intitulé, ainsi que celui nécessaire pour la saisie au clavier.

Les multiples conventions de nommage existantes doivent satisfaire peu ou prou les quelques impératifs que nous venons d'énoncer. Dans l'ensemble, harmoniser un intitulé vis-à-vis de telle ou telle nomenclature peut être une tâche délicate, voire fastidieuse. Le maintien de la cohérence peut être particulièrement difficile.

Histoire

Conçue pour être indépendante du langage de programmation utilisé, la notation hongroise a trouvé sa première utilisation à grande échelle dans BCPL (Basic Combined Programming Language). N'intégrant qu'un seul type de données - le type word, dont la taille correspond à celle du mot machine gérée à l'échelle du processeur, ce langage ne fournit par conséquent aucun indice quant à l'interprétation des valeurs constitutives d'un programme. La notation hongroise vise à remédier à cette lacune, cela en fournissant aux concepteurs de logiciels une connaissance explicite, instinctive au premier abord, du domaine d'utilisation prévu pour chaque variable.

La notation hongroise a été pensée par Charles Simonyi, un informaticien hongrois émigré aux États-Unis en 1968. Après ses études à l'Université de Berkeley, il est embauché au Xerox PARC à Palo Alto en Californie. Recruté par Microsoft en 1981, il crée et prend la direction d'un groupe de travail dont la première réalisation serait un logiciel de traitement de texte WYSIWYG. (Ledit logiciel, sorti en 1983 sous le nom de Microsoft Word, deviendra une des mannes financières les plus importantes pour la firme de Redmond.) Se basant sur ses propres expériences, qui le font mettre les fautes de typage parmi les premières sources de dysfonctionnement des programmes, Simonyi jette les bases d'une convention qui promeut l'emploi systématique de noms de variables préfixés de leur type. Il envisage par ce biais de diminuer les erreurs sur ces variables dues à des manipulations impropres. Son poste chez Microsoft lui a permis de tester ce modèle, l'imposant comme standard de travail pour son équipe.

La désignation accompagnant la méthode hongroise est une référence directe au pays d'origine de Simonyi, né à Budapest, capitale et plus grande ville de la Hongrie. Sur un autre plan plus factuel, la formation d'associations conformes à la notation hongroise emprunte pour une grande part à la grammaire descriptive classique du hongrois, qui fait une utilisation intensive de suffixes pour conférer à partir d'un alphabet sémantique élémentaire toutes sortes de sens.

Après avoir été popularisée au sein de la division Application de Microsoft (d'où l'appellation Apps dont elle sera affublée par la suite), la notation hongroise atteint les équipes de développement à l'origine de Windows, futur système d'exploitation phare de la firme. Les concepteurs en reprennent les principes fondateurs, mais soulignent que certaines pratiques liées à la méthode hongroise tendent à dévoiler plus facilement des informations sur la partie logique d'une variable (ce à quoi elle sert) que sur ses aspects physiques (les valeurs qu'elle peut stocker, et surtout sous quelle forme). Ils imaginent de ce fait un dialecte de la notation hongroise, étiquetée Systems, plus proche de la machine et des traitements et des opérations qui s'y déroulent.

En guise d'anecdote, et dans un registre plus décalé, il faut aussi mentionner que pour quantité de personnes, la notation hongroise est remarquable avant tout par le fait de séries plus ou moins longues et de complexité variable de consonnes, qui sont en tout état de cause imprononçables sorties de quelques contextes linguistiques particuliers. Notez encore, toujours au registre du détail, que le hongrois fait partie de la famille des langues ouraliennes (du nom de l'Oural, leur lieu supposé d'origine), et contrairement aux langues slaves, plutôt riche en voyelles.

Notation hongroise *Apps*

Dévolue surtout à rendre compte de cas d'utilisation, la notation hongroise *Apps* invente un plan de nommage tourné vers la sémantique, orienté par l'utilité fonctionnelle individuelle des données de programme. Les symboles envisagés de cette manière le sont sur la base d'un ensemble de préfixes commandé par la force de liens logiques ou formels, plutôt que par la forme ou par la nature. Ainsi, à titre d'exemple, le préfixe i peut éventuellement correspondre à un indice, cb à une taille en octets (count of bytes), et les préfixes rw et col faire référence, respectivement, à une valeur de ligne et une valeur de colonne. Quelques noms élaborés à partir de ce principe : lName, pour une variable hébergeant un entier long (long integer); usName, pour une variable appelée

à contenir une chaine de caractères dont on peut avoir un doute sur la sécurité (unsafe string), bName pour indiquer un type booléen (bool), et ainsi de suite.

L'orientation principale sur laquelle la notation hongroise Apps assoie ses modèles est de nature à donner une idée des aboutissants potentiels d'une variable ou d'une fonction, sans chercher à en révéler les tenants exacts. Un développeur attentif au versant Apps de la convention optera par conséquent de préférence pour des symboles qui correspondent à cette optique, par exemple strName à la place de szName, ou cName au lieu de dwName. La forme choisie dans le premier cas parvient à révéler l'existence d'une chaine de caractères, sans cependant rien dire de l'implémentation sous-jacente. Lors du second cas de figure, la symbolique préférée l'est du fait d'indiquer que la variable agit en tant que compteur, qu'il s'agisse d'un entier numérique étant un détail de relativement peu d'importance. En définitive, cet accent mis sur la sémantique permet de véhiculer toutes sortes d'informations utiles.

L'intérêt évident de la notation hongroise Apps est d'aller au-devant de l'utilisation accidentelle d'une donnée dans le mauvais contexte. Il est par exemple presque certain, sans même connaître les détails afférents, que l'expression rwXxx + cbXxx, ajoutant un numéro de ligne à une taille, est un défaut de conception dans le code.

Un constat qui ne peut manquer d'être établi, et d'interpeller le regard moderne, en ce qui concerne les constructions avancées dans la notation hongroise Apps est que toutes ne sont pas de nature sémantique. Certains préfixes semblent en effet avoir trait à la qualité intrinsèque d'une donnée brute, tels que sz pour ce qui est des chaines de caractères, ou b pour une valeur au niveau de l'octet. Sur la question du bien-fondé de ces éléments, il faut se rappeler que les langages de programmation sur lesquelles ils ont pris corps ne disposaient pas à l'époque de systèmes de types. Les concepts que les langages modernes, et par extension les concepteurs de logiciels tiennent pour acquis aujourd'hui n'existaient pas.

La notation hongroise Apps est nommée de la sorte en référence à l'enracinement de ce standard au sein de la division Applications de Microsoft. Son emploi à guidé des logiciels comme Word, Excel, et bien d'autres.

Notation hongroise *Systems*

Mettant en exergue des informations techniques plutôt que des pistes d'utilisation, la notation hongroise Systems consiste à insuffler à même le nom d'une donnée le type avec lequel elle est en interaction, celui dont elle tient s'il s'agit d'une variable, celui qu'elle retourne dans le cas d'une fonction.

La notation hongroise Systems tire son nom des conditions dans lesquelles elle a été pensée.

Raison d'être et avantages

L'avantage premier auquel conduit l'emploi de la notation hongroise est que son déploiement systématique introduit *de facto* une normalisation au niveau des règles programmatiques. Si l'appréciation de tel ou tel protocole est laissée à la discrétion et au goût de chacun, il est généralement de bon ton, dans un domaine aussi vaste que l'informatique, qui mérite un grand nombre de mises au point et de réflexions sur le sujet, d'appliquer des standards. Sans trop entrer dans les détails à ce stade, la notation hongroise procure les mêmes bénéfices, pour ce qui est de la discipline, que ceux inhérents à tout type de normalisation.

Un second bienfait dont découle l'application de la notation hongroise est une meilleure lisibilité du code source, considérée dans une optique communément partagée comme une des composantes primordiales de la qualité logicielle. En règle générale, les partisans de la notation hongroise apprécient le surcroît d'information immédiate que leur apporte l'ajout d'informations à chaque nom.

Du fait qu'elle les regroupe selon une schématisation bien formée, la notation hongroise permet de mécaniser en grande grande partie la génération des noms. Cela peut se montrer utile, par exemple, lors du recours à des outils de documentation automatique, qui peuvent aider à dépister éliminer d'éventuelles erreurs.

Parmi tous les atouts qui ont participé au succès de la notation hongroise, l'un en particulier mérite une mention spéciale, en ce sens que le standard en question est soutenu par une firme dont le poids, sur la scène mondiale

de l'informatique, est considérable. Les pratiques technologiques qu'elle véhicule tendent donc, de ce fait, à se propager plus loin et plus rapidement que toutes autres.

Une fois maitrisée, la notation hongroise permet une lecture très précise du code. Les erreurs de type ou d'utilisation de pointeurs notamment sont beaucoup plus faciles à repérer.

Inconvénients

Si elle présente effectivement plusieurs avantages, la notation hongroise n'en reste pas moins inadaptée dans quelques cas. Une critique couramment soulevée à son encontre est qu'elle impacte négativement la lisibilité du code. Il faut sur ce plan se rappeler que presque tous les préfixes imaginés conformément à la norme le sont à partir d'une ou de plusieurs consonnes. Sans même évoquer de questions esthétiques, cette forme rebute la plupart des concepteurs, qui y voient (au moins pendant leur période d'acclimatation) une perte d'accessibilité globale des éléments concernés.

Au niveau programmatique, plusieurs facteurs contribuent à mettre en doute l'utilité de la notation hongroise. Entre autres :

- Les langages modernes disposent d'un système de types particulièrement élaboré que les compilateurs font respecter. De ce point de vue, toute forme de codification, y compris la notation hongroise, est considérée comme un obstacle. Ce point est d'autant plus mis dans en relief dans une perspective purement objet, où l'emploi de noms qui puissent transmettre des informations liées au tapage va à l'encontre de l'objectif fixé, à savoir rendre fonctionnellement équivalent toutes sortes de types. Il est sur cet aspect même possible de considérer l'emploi de la notation hongroise comme un biais évaluatif favorisant les opérations réalisables sur un objet - dont il faut pour l'occasion saisir pleinement le type - plutôt que se préoccuper de ce qu'une instance d'objet peut faire.

Les environnements de développement modernes disposent pour la plupart d'une aide intégrée visant à contextualiser l'emploi des variables et des fonctions. Certains offrent en plus de cela un marquage automatique des opérations qui essaient de manipuler des types incompatibles, voire même suggèrent quels sont ceux attendus. Ces auxiliaires tendent à rendre la notation hongroise largement inutile.

- Souvent, connaitre l'usage potentiel d'une donnée permet d'en déduire - même approximativement - le type. A l'inverse, discerner clairement le type d'une donnée ne permet pas d'en pressentir la fonction.

- Lorsque les noms choisis pour identifier des objets sont suffisamment descriptifs, l'ajout d'informations de type peut être redondant, voire superflu. Il est par exemple très peu probable que la variable ProcessName fasse référence à autre chose qu'une chaine de caractère.

- Une tendance générale du développement logiciel est de s'orienter vers des fonctions courtes, dont le rôle peut être facilement identifié. Dans un tel contexte, la déclaration d'une variable ne devrait jamais se situer très loin des opérations qui en font usage.

- Modifier le type d'une variable signifie, dans un projet pour lequel on a choisi d'appliquer la notation hongroise, la renommer, et ce partout où elle est utilisée. Un

- Étroitement liée au langage C, la notation hongroise véhicule des usages qui ont du sens d'abord et avant tout pour ce langage. De ce fait, plus important encore et en dépit de la généralité de ses termes, elle n'est pas directement transposable à d'autres langages de programmation.

Sur le fond, peut-être l'aurez déjà remarqué à la lecture de la liste qui précède, bon nombre des arguments avancés contre la notation hongroise ciblent en réalité le volet Systems de celle-ci, la catégorie Apps étant relativement épargnée.

A noter que les pratiques contemporaines en matière de programmation informatique semblent avoir sur la nature même des diverses conventions de nommage, par extension de la notation hongroise, un regard plutôt hostile. Une illustration particulièrement marquante de ce désaveu tient au fait que Microsoft, qui en fut

pourtant le premier promoteur, affirme désormais le caractère obsolète du standard. (Pour information, cette manière d'envisager les choses peut notamment être perçue dans les publications de Microsoft Press relatives au cadre logiciel .NET.) Tristement, les raisons énoncées ne sont que très peu significatives. La position de la firme à ce sujet consiste à mettre au-devant de la scène la technologie IntelliSense, qui consiste en un agrégat de fonctionnalités visant à faciliter l'écriture et la gestion du code, et fournit en l'occurrence une aide (particulièrement bien vue, cela dit) concernant le suivi des données. L'existence de telles solutions de support rendrait l'emploi d'une notation préfixée redondante et peu judicieuse. Si elle est défendable sur le plan de la démarche, cette position reste néanmoins peu avisée pour plusieurs raisons : (1) parce que cette position présume qu'un seul environnement intégré impose le rythme à un pan entier du génie logiciel ; (2) parce que cette position présume que plus personne ne lit de code imprimé ; (3) parce que les technologies de type IntelliSense existent depuis longtemps et, sous d'autres noms et des circonstances différentes, n'ont jamais été vues comme moyen de mettre fin à telle ou telle approche.

Pour finir sur une anecdote amusante, un aspect sur lequel reviennent les détracteurs les plus opiniâtres de la notation hongroise est que même au sein de Microsoft Windows, le maintien de la conformité des variables à ladite norme ne paraît pas, à une certaine échelle, avoir dépassé le stade des versions 16 bits du système. Pour comprendre le propos, il est nécessaire de revenir sur les conditions où s'opérait le traitement des messages dans Windows, ainsi que sur comment elles ont évolué. Ainsi, dans le contexte de Windows 16 bits, les deux types primitifs qui véhiculaient des informations de message étaient encodés l'un sur 16 bits, ce que reflétait le nom wParam (word), l'autre sur 32 bits, ce que soulignait le nom lParam (long). Lors du passage à une architecture 32 bits, aucune mesure ne fut prise afin de mettre en phase ces noms à leur nouvel environnement - la forme wParam aurait dû évolué vers dwParam. Pour certains, cela ne fait que mettre en relief le manque d'evolutivité du modèle. Pour d'autres, cela tient essentiellement à la dimension sémantique du terme word, laquelle renverrait ici au mot machine, dont la taille est fonction de la plateforme matérielle.

MSDN (*Microsoft Developer Network*)

Microsoft Developer Network (MSDN) est la partie de Microsoft responsable de la gestion de la relation de la firme avec les développeurs qui conçoivent des appareils, services et applications pour Windows. Les médias résultants sont de nature diverse : sites web, bulletins d'information, livres blancs, conférences, articles de presse, entretiens, journaux Web, disques physiques… et s'additionnent pour donner naissance à un catalogue à ce jour sans égal en matière de programmation. Pour accéder à la page principale du site MSDN, consultez http://msdn.microsoft.com/fr-fr/default.aspx.

Sites web

Structurée pour l'essentiel autour des moyens de communication offerts par le réseau Internet, l'infrastructure MSDN se présente au premier lieu comme un ensemble de sites web qui accueillent des informations et des discussions utiles au sujet des produits et systèmes Microsoft. Les documents ainsi constitués, qui se chiffrent en millions de pages, sont de la main soit de l'entreprise soit de la communauté de développeurs au sens large. De façon générale, l'accent est mis davantage sur la correspondance et le dialogue plutôt que sur la diffusion unilatérale de contenus.

Bibliothèque MSDN

Ressource primordiale pour les développeurs ciblant les outils, produits et technologies Microsoft, la bibliothèque MSDN contient des informations de programmation technique, des échantillons de code, de la documentation, des articles techniques et des guides de référence. Elle regroupe par exemple l'intégralité des API Windows, du plus bas niveau (écriture de pilotes de périphériques) au plus haut (conception de modules pour les produits Microsoft comme Visual Studio).

La bibliothèque MSDN peut être consultée aux adresses suivantes : http://msdn.microsoft.com/fr-fr/library/ pour la version française et http://msdn.microsoft.com/en-us/library/ pour la version anglaise. Si vous maitrisez l'anglais, nous vous conseiller de consulter cette version, car elle est beaucoup plus complète. De plus, vous êtes certain d'y trouver une documentation parfaitement à jour. Comme il assez est facile de se perdre

dans l'arborescence des contenus hébergés par la bibliothèque, Microsoft propose une page web où il est possible de procéder à des recherches. Pour effectuer une recherche, rendez-vous à l'adresse suivante : https://social.msdn.microsoft.com/Search/. La recherche peut s'effectuer soit en anglais soit en français. Cette base de données s'interroge par mot clé ; on peut également faire des recherches avec des opérateurs booléens et restreindre l'espace de recherche à différentes bases de données.

Forums

Les forums MSDN constituent un espace propice aux échanges et aux discussions en ce qui concerne un large éventail de sujets liés au développement de logiciels. Animés par une communauté très active, ils offrent la possibilité de poser des questions, partager des informations ou échanger des idées avec d'autres utilisateurs et des experts partout dans le monde.

Alimentés en permanence de questions et de réponses techniques, les forums MSDN sont l'endroit idéal pour obtenir de l'aide, mais également en apporter. Si vous ne trouvez pas de réponse dans un forum, vous pouvez poser une nouvelle question, être averti lors de la survenue de nouvelles réponses et noter la réponse adéquate. L'interface graphique utilisateur des forums est conçu pour les rendre plus facile à utiliser que les groupes de discussion.

Pour entamez la navigation sur les forums MSDN, consultez le site https://social.msdn.microsoft.com/Forums/.

Blogs

Les blogs MSDN couvrent un éventail considérable de sujets relatifs aux technologies Microsoft. Certains de ces blogs sont consacrés entièrement à un produit majeur, par exemple Windows, Visual Studio ou l'environnement PowerShell, tandis que d'autres privilégient la diffusion de connaissances techniques spécifiques aux métiers de l'informatique.

Voici une sélection de liens que nous vous conseillons :

- *The Old New Thing* - http://blogs.msdn.microsoft.com/oldnewthing/
- *Ntdebugging* - http://blogs.msdn.com/ntdebugging/

Un mot sur la conduite à tenir au sein de la communauté de développeurs. Si les auteurs respectifs offrent en général volontiers leur aide, les journaux MSDN sont ouverts à quiconque est à la recherche d'informations sur un sujet technique. Il est ici important de rappeler que les règles qui régissent les forums s'appliquent de la même façon et dans les mêmes proportions aux journaux. Par conséquent, si vous étiez amené à poser une question, veillez à le faire à un endroit approprié.

Base de connaissances

La base de connaissances Microsoft renferme une mine d'informations pratiques et de données sur tous les produits et technologies Microsoft. Alimentée en permanence par des milliers de professionnels de l'assistance, elle est mise à jour, développée et améliorée régulièrement afin de mettre les informations technologiques les plus récentes au vu et au su de tous.

Organisée à la façon d'un vaste dépôt de documents en ligne, chaque article de la base de connaissances Microsoft traite d'un sujet différent et vous y trouverez, presque à coup sûr, les informations que vous recherchez, ou à tout le moins des pistes pour le faire. Le recours à cette base présente toutefois un inconvénient non négligeable, à savoir le fait d'être a priori dépourvue de tout principe évident d'organisation. Cela signifie que si vous ne ciblez pas correctement votre recherche, votre requête risque fort d'être noyée dans le bruit documentaire.

Chaque article de la base de connaissances est identifié par un numéro à six chiffres. À cela s'ajoute un titre décrivant le thème couvert, des informations éventuelles axées sur le produit et la version de produit, ainsi qu'un

résumé synthétique présentant brièvement les sujets abordés. Au-delà de ça, chaque article est étiqueté au moyen de mots-clés en relation avec les thématiques dans lesquelles son contenu prend place.

La Base de connaissances Microsoft est disponible sur le réseau MSDN à l'adresse suivante : http://support.microsoft.com.

Magazine

MSDN Magazine fournit des explications approfondies sur l'implémentation des technologies et des outils Microsoft actuels. Il est diffusé sous forme de publication mensuelle en format numérique et imprimé. Pour consulter le magazine en ligne, rendez-vous sur la page https://msdn.microsoft.com/fr-fr/magazine/.

Traduction automatique

Un grand nombre des pages qui entérinent l'engagement de Microsoft sur le secteur du support et l'assistance bénéficient de services de traduction automatisés. Microsoft propose cette traduction pour offrir aux personnes ne maîtrisant pas l'anglais l'accès au contenu relatif aux produits, services et technologies de la firme. Les documents traduits par ce biais peuvent cependant se révéler de moins bonne facture que ceux ayant profité d'une traduction humaine professionnelle.

Mode utilisateur et mode noyau

Pour empêcher les applications d'accéder à et/ou de modifier les données vitales du système d'exploitation, Windows s'appuie sur deux modes d'exécution distincts du processeur : *mode utilisateur* et *mode noyau*, l'un et l'autre servant à définir une politique globale en matière matière d'accès, incluant l'interface avec la mémoire et l'exécution des instructions machine. Employée avant tout par Windows afin de parvenir à modèle de contrôle d'accès efficace, cette démarcation entérine au premier stade une scission nette entre le coeur du système, jugé sûr, et ses satellites, considérés comme moins digne de confiance au niveau de la sécurité et de la stabilité.

Le code des applications, ainsi que des sous-systèmes sur lesquels s'appuient les applications de l'utilisateur, est exécuté en mode utilisateur. Les processus en mode utilisateur ne bénéficient pas d'un accès direct au matériel ; ils sont limités à une zone mémoire affectée (excluant la mémoire système et la mémoire d'autres processus) et à un sous-ensemble des instructions du processeur (excluant celles ayant à voir avec la configuration du système informatique). A contrario, le code du système d'exploitation (par exemple les services système et les pilotes) est exécuté en mode noyau, lequel donne accès à l'ensemble de la mémoire et à toutes les instructions du processeur.

Bien que chaque processus Windows ait son propre espace mémoire, le code système et le code des pilotes de périphérique exécuté en mode noyau se partagent un même espace d'adressage, et disposent ainsi des mêmes accès sans restriction à la mémoire système. Autrement dit, tout code noyau a l'accès complet a la mémoire de l'espace système, avec ce que cela suppose d'attention à entretenir lors le chargement de pilotes tierce partie, capables à l'occasion de contourner le modèle de sécurité Windows, voire accidentellement ou volontairement faire s'effondrer le système.

La partition entre mode utilisateur et mode noyau constitue l'un des éléments de base du contrôle d'accès. Les applications exécutées en mode utilisateur ne peuvent ainsi, intentionnellement ou accidentellement, accéder à des données ne leur appartenant pas, ni solliciter diverses instructions machine sensibles en matière de sécurité. En interne, les modes de fonctionnement des processeurs sont implémentés via un mécanisme d'interceptions (*trap*). Quand une application outrepasse ses droits, par exemple quand une opération issue de son code machine exécutable accède à de la mémoire protégée, s'ensuit le déclenchement d'une interruption matérielle, laquelle est interceptée par le noyau qui met généralement fin à l'application fautive.

Les applications utilisateur passent du mode utilisateur au mode noyau généralement pour solliciter un *service système*. Quand une application requiert le concours d'une routine exécutée en mode noyau, elle le fait à l'aide

d'une instruction processeur spéciale, qui force le processeur à basculer en mode noyau et transfère le contrôle d'une manière sécurisée vers des points d'entrée prédéfinis dans un anneau de plus bas niveau. Le système d'exploitation intercepte cette instruction, remarque la demande vers un service système, valide les arguments que le thread a passé à la fonction système, puis exécute la fonction interne. Lorsque celle-ci se termine, le système d'exploitation repasse le processeur en mode utilisateur et restitue au thread son contexte original, dès lors en mesure de continuer son exécution en mode utilisateur.

Les développeurs peuvent intégrer la partie du système d'exploitation exécutée en mode noyau via la conception de pilotes de périphériques. En réalité, nombre des fonctionnalités attribuées au noyau Windows, telles le système de fichiers, le système de fenêtrage et de graphisme ou la pile réseau TCP/IP, sont conçus de cette façon, conséquemment mis en oeuvre sur la base d'un ou de plusieurs pilotes indépendants.

Anneaux de protection

Les modes utilisateur et noyau que nous venons de voir sont une illustration concrète de comment Windows s'appuie sur les anneaux de protection des architectures x86 et ses extensions. Chaque anneau défini dans un tel schéma l'est de telle sorte à correspondre à un niveau de privilège et de sécurité empêchant les dommages, intentionnels ou non, émanant de code de moindre privilège. Les anneaux sont arrangés dans une hiérarchie allant du plus privilégié (celui qui est le plus sécurisé, habituellement le numéro zéro dit Ring 0) au moins privilégié (le moins sécurisé, habituellement l'anneau le plus élevé).

La notion d'anneaux destinés à sécuriser le système d'exploitation provient à l'origine de Multics, un prédécesseur fortement sécurisé de la famille actuelle des systèmes UNIX, qui s'est par ailleurs brillamment distingué par le grand nombre d'idées novatrices qu'il apportait. Il fut en outre le premier système d'exploitation à intégrer la notion de sécurité informatique (liée au multiutilisateur) dès sa conception.

L'utilisation efficace de l'architecture en anneau implique une coopération étroite entre le matériel (le processeur) et le logiciel d'exploitation. De nombreuses architectures modernes de processeurs intègrent une telle forme de protection, bien que les systèmes d'exploitation ne l'exploitent pas toujours entièrement. Windows, par exemple, quelque soit le nombre d'anneaux que la plateforme d'accueil lui aura conférés, n'en emploie que deux (niveau zéro pour le code noyau, trois en ce qui concerne le code utilisateur). Il déjoue ainsi les pièges d'un modèle de sécurité trop astreignant pour former la base d'un système d'usage général (l'un de ses objectifs de conception), et reste de cette manière compatible avec les quelques architectures matérielles n'incluant que deux types d'anneaux.

Les processeurs de la famille x86 disposent au minimum de quatre anneaux de privilèges. Un autre éventuellement présent correspond au mode hyperviseur sur les machines pourvues d'extensions matérielles de virtualisation (Intel VT et AMD Pacifica, par exemple). Considérant que le système d'exploitation est dans cette configuration un programme sous l'égide d' un outil tiers, ce mode de fonctionnement est quelquefois vu comme un potentiel niveau -1.

Visualisation des temps utilisateur et noyau

L'utilitaire Performances permet de suivre la répartition du temps processeur entre les applications (processus utilisateur) et le système (processus noyau). Voici la procédure :

1. Démarrez l'utilitaire Performances, par exemple en saisissant la commande perfmon.msc dans la boite de dialogue Exécuter.

2. Cliquez sur le noeud Analyseur de performances.

3. Cliquez sur le bouton Ajouter (+) de la barre d'outils.

4. L'objet Processeur étant sélectionné (il l'est automatiquement), cliquez sur le compteur % Temps privilégié puis, en maintenant la touche CTRL enfoncée, cliquez sur le compteur % Temps utilisateur.

5. Cliquez sur Ajouter puis faites Ok.

6. Déplacez rapidement la souris dans un sens puis dans un autre. Selon l'amplitude (minorée par la surface d'affichage) et la vitesse donnée à votre geste, vous devriez constater des variations de plus ou moins grande importance au sein de la courbe % Temps privilégié, à quoi correspond le temps consacré à recevoir et à traiter les interruptions souris, ainsi qu'à gérer les opérations qui s'ensuivent dans la portion mode noyau du sous-système Windows.

Le Gestionnaire des tâches constitue également un moyen pratique de voir rapidement cette activité. Allez dans l'onglet Performances puis sélectionnez l'objet Processeur. Positionnez ensuite le curseur de la souris sur le graphique, faites un clic droit et activez l'option Afficher les temps du noyau. Cela a pour effet de superposer une seconde courbe qui représente le temps de processeur utilisé par le noyau.

Pour voir combien un processus consomme de temps utilisateur et de temps noyau, procédez de la manière qui suit.

1. Démarrez l'utilitaire Performances. Si une instance de ce processus est déjà en marche, nous vous conseillons de supprimer les compteurs éventuellement présents, ou sinon de masquer l'affichage des informations qui en résultent.

2. Cliquez sur le noeud Analyseur de performances, et ensuite sur le bouton Ajouter (+) de la barre d'outils.

3. Parmi ceux disponibles, choisissez les compteurs % Temps privilégié et % Temps utilisateur. À titre informatif, le compteur % Temps Processeur, qui donne le temps CPU total utilisé par un processus, est un cumul des compteurs % Temps privilégié et % Temps utilisateur du même objet.

4. Sélectionnez le nom du processus duquel vous souhaitez voir les temps d'exécution dans la zone Instances de l'objet sélectionné.

5. Cliquez sur Ajouter, puis sur Ok.

Du fait que quand le contexte l'exige, Windows effectue une transition de mode, il est tout à fait commun de voir un thread utilisateur passer une partie de son temps d'exécution en mode utilisateur et une autre partie en mode noyau. En outre, les opérations liées à la gestion des fenêtres et au dessin étant en grande majorité mises en oeuvre coté noyau, les applications fortement orientées graphismes passent plus de temps en mode noyau qu'en mode utilisateur.

La liste qui suit énumère les compteurs de performance qui concernent le mode processeur, et partant, se prêtent particulièrement bien à un examen du temps que passe chacune des unités fonctionnelles du système (processeurs, processus et threads) en mode noyau et en mode utilisateur.

- *Processeur : % Temps privilégié* Pourcentage de temps qu'un processeur (ou éventuellement tous les processeurs) a passé en mode noyau.

- *Processeur : % Temps utilisateur* Pourcentage de temps qu'un processeur (ou éventuellement tous les processeurs) a passé en mode utilisateur.

- *Processus : % Temps privilégié* Pourcentage de temps que les threads d'un processus ont passé en mode noyau.

- *Processus : % Temps utilisateur* Pourcentage de temps que les threads d'un processus ont passé en mode noyau.

- *Thread : % Temps privilégié* Pourcentage de temps qu'un thread a passé en mode noyau. En interne, ce compteur (exprimé en pourcentage) dérive d'un calcul fondé sur la valeur de l'attribut KernelTime dans le bloc KTHREAD d'un thread.

- *Thread : % Temps utilisateur* Pourcentage de temps qu'un thread a passé en mode utilisateur.

Tableau 2.3. Temps utilisateur et temps noyau

Élément	Type	Lieu	Autres références
TotalUserTime	LARGE_INTEGER	JOBOBJECT_BASIC_-ACCOUNTING_INFORMATION	
TotalKernelTime	LARGE_INTEGER	JOBOBJECT_BASIC_-ACCOUNTING_INFORMATION	
ThisPeriodTotalUserTime	LARGE_INTEGER	JOBOBJECT_BASIC_-ACCOUNTING_INFORMATION	
KernelTime	LARGE_INTEGER	KERNEL_USER_TIMES	NtQueryInformationProcess (ProcessInformationClass = ProcessTimes), NtQueryInformationThread (ThreadInformationClass = ThreadTimes)
UserTime	LARGE_INTEGER	KERNEL_USER_TIMES	NtQueryInformationProcess (ProcessInformationClass = ProcessTimes), NtQueryInformationThread (ThreadInformationClass = ThreadTimes)
KernelTime		KPRCB	
UserTime		KPRCB	

Windows-1252 et Unicode

Windows-1252 et Unicode sont deux normes informatiques visant à permettre le codage de texte écrit. Le premier (Windows-1252, appelés par confusion ANSI) concerne le codage des caractères de l'alphabet latin, là où Unicode s'entend à décrire de manière unifiée n'importe quel caractère de n'importe quel système d'écriture. Windows s'exportant partout dans le monde, et se devant d'assurer la pérennité des données textuelles, les versions récentes de Windows utilisent le standard Unicode, Windows-1252 subsistant dans les composants hérités du système. Nous le présentons néanmoins pour marquer son existence et son importance passée dans le système d'exploitation.

Windows-1252 ou CP1252 est un jeu de caractères, utilisé historiquement par défaut sur le système d'exploitation Microsoft Windows en anglais et dans les principales langues d'Europe de l'Ouest, dont le français. Il constitue une extension de la norme ISO/CEI 8859-1 (souvent appelée Latin-1 ou Europe occidentale), mais se distingue de cette dernière par l'utilisation de caractères imprimables, plutôt que des caractères de contrôle, dans les codes 128 à 159. Ces derniers ajoutent un certain nombre de caractères, telles les guillemets anglais, les points de suspension, les tirets cadratin et demi-cadratin.

Windows-1252 est parfois appelé ANSI, du nom de l'organisme chargé de superviser le développement de normes pour les produits, les services, les procédés, les systèmes et les employés des États-Unis, l'Institut de normalisation américaine (ANSI, American National Standards Institute). En réalité, Windows-1252 n'a jamais été un standard de l'ANSI. Le nom est donc abusif, faute en incombe à l'utilisation dans la communauté Windows du terme *page de code* ANSI (ACP, *ANSI code page*) pour faire référence à Windows-1252. Bien que très populaire, Windows-1252 n'est jamais devenu une norme ANSI, mais le jargon s'y rapportant est néanmoins resté.

Sous l'influence des problèmes d'interopérabilité (les jeux de caractère classiques possèdent des caractéristiques très différentes les uns des autres) et de la mondialisation des échanges (ils ne peuvent au mieux prendre en charge que quelques langues), et bien que le codage Windows-1252 reste très utilisé, ce codage subit la concurrence et le développement du standard Unicode.

Unicode est un mécanisme universel de codage de caractères. Il définit une manière cohérente de coder des textes multilingues et présente un moyen commode de représenter la plupart des alphabets connus dans le monde. Mis en œuvre dans une grande majorité des systèmes d'exploitation modernes, Unicode est au centre de tout logiciel à caractère international.

Le standard Unicode est constitué d'un répertoire de plus de 109 000 caractères couvrant 93 écritures, d'un ensemble de tableaux de codes pour référence visuelle, d'une méthode de codage et de plusieurs codages de caractères standard, d'une énumération des propriétés de caractère (lettres majuscules, minuscules, symboles, ponctuation, etc.), et d'un certain nombre d'éléments liés, tels que des règles de normalisation, de décomposition, de tri, de rendu et de directionalité (pour l'affichage correct de texte contenant à la fois des caractères d'écritures droite à gauche, comme l'arabe et l'hébraïque, et de gauche à droite).

Les données Unicode peuvent être codées sous trois formes principales : une forme codée sur 32 bits (UTF-32), une forme sur 16 bits (UTF-16), et autre forme de 8 bits (UTF-8) conçue pour faciliter son utilisation sur les systèmes ASCII préexistants. Le standard Unicode est identique à la norme internationale ISO 10646 en ce qui concerne l'affectation des caractères (leur numéro) et leurs noms (Unicode est généralement mieux connu que la norme ISO-10646 qui en est un sur-ensemble).

Windows enregistre la plupart des données textuelles internes à l'échelle de caractères Unicode 16 bits, ce que ne fait pas forcément toutes les applications - beaucoup gèrent encore des chaînes de caractère ANSI à 8 bits. Aussi, pour assurer la comptabilité, les fonctions Windows acceptant des chaînes comme paramètres ont deux points d'entrée : une version Unicode (large, 16 bits) et une version ANSI (étroit, 8 bits). Lorsqu'une application sollicite la version étroite d'une fonction Windows, les chaînes en entrée sont converties en Unicode avant d'être traités par le système et les chaînes en sortie sont converties en ANSI avant d'être retournées à l'application.

Dans les versions de Windows antérieures a Windows 2000, les éditions Asie et Moyen Orient étaient des déclinaisons à part entière de la version commerciale américaine standard, et contenaient des fonctions supplémentaires pour les saisies et affichages complexes (par exemple, pour gérer les textes bidirectionnels). Du moment où Unicode fut intimement mêlée à Windows, à partir de Windows 2000 donc, cela permis à Microsoft de ne plus prendre en charge des versions distinctes pour tels ou tels langages, mais de reposer à la place sur une même installation gérant plusieurs langues (via l'ajout de packs linguistiques). Les applications peuvent aussi utiliser les fonctions Windows permettant à un même binaire de gérer de multiples langues.

L'architecture de Windows est ainsi faite que les applications conçus sous la perspective Unicode sont plus performantes que les autres. Les fonctions du noyau Windows s'appuyant exclusivement sur Unicode, les chaînes échangées entre le système d'exploitation et les applications usant d'un autre système de représentation des caractères nécessitent un processus de traduction continu qui, s'il n'a pas forcément un impact décisif, n'en reste pas moins présent.

Pour plus d'informations sur Unicode, visitez le site www.unicode.org.

Déterminer si un texte est ANSI ou UNICODE

L'adaptation à différents jeux de caractères peut conduire les applications manipulatrices de fichiers à s'assurer du positionnement Unicode de telles ou telles données. La fonction IsTextUnicode, rendue visible par Advapi32.dll, sert précisément cet objectif.

Le premier paramètre, pvBuffer, spécifie l'adresse du tampon mémoire duquel vérifier la conformité envers Unicode. L'utilisation d'un pointeur non-typé reflète le manque d'informations concrètes sur la nature des données que renferme ledit tampon.

Le second paramètre, cb, indique le nombre d'octets qui seront lues depuis le tampon. Encore une fois, comme les informations disponibles à ce stade ne sont pas suffisantes pour faire référence à un nombre de caractères spécifique, la valeur exprimée ici l'est en nombre d'octets. Elle peut en outre représenter une taille inférieure à celle du du tampon, cela dit au détriment de l'exactitude des conclusions que IsTextUnicode est appelée à fournir.

Le troisième paramètre, pResult, contient l'adresse d'un nombre entier dont la valeur, sous forme d'un masque binaire, sert à l'appelant (comprendre en entrée) à clarifier les objectifs et la portée de l'audit attendu, et pour ce qui concerne la fonction IsTextUnicode elle-même (comprendre en sortie) à relayer les les résultats effectivement obtenus par cet audit en fonction de la démarche suivie.

UNC (*Uniform Naming Convention*)

La convention de nommage uniforme (UNC, *Uniform Naming Convention*) est un ensemble de règles destinées à identifier à l'aide d'une adresse unique n'importe qu'elle ressource disponible aux utilisateurs réseau, telle que les répertoires, les fichiers, les canaux de transmission nommés (*named pipes*) et les imprimantes. Dans les systèmes d'exploitation Windows, et éventuellement d'autres systèmes d'exploitation, UNC peut remplacer dans ses fonctions le système de nommage local (par exemple, C:\partage).

Les dénominations UNC doivent se conformer à la syntaxe \\NOM_SERVEUR\NOM_PARTAGE, dans laquelle NOM_SERVEUR correspond au nom d'un serveur sur le réseau et NOM_PARTAGE au nom de la ressource partagée. Appliquées à des répertoires ou des fichiers, elles peuvent également faire mention du chemin d'accès du répertoire sous le nom de partage, conformément à la syntaxe \\NOM_SERVEUR\NOM_PARTAGE\REPERTOIRE \NOM_FICHIER.

Les dénominations UNC ne requièrent pas d'une ressource qu'elle soit soit *stricto sensu* un partage réseau, mais désigne par convention comme tel toute ressource accessible par son intermédiaire.

Chaque élément constitutif d'une qualification UNC, à l'exception du dernier, est appelé composant de chemin d'accès (*pathname component*) ou composant de chemin (*path component*). Un chemin UNC valide doit en contenir au moins deux, avec le premier considéré comme le *premier composant de chemin*, le second comme *deuxième composant de chemin*, et ainsi de suite. Le dernier élément du chemin est aussi appelé *composant feuille* (*leaf component*), le situant, dans une terminologie empruntée aux arbres binaire, au plus bas échelon de la hiérarchie.

Système de fichiers

Les périphériques de stockage principaux des systèmes informatiques contemporains, tels que disque dur, CD-ROM et clé USB, doivent au préalable de toute utilisation être configurés par l'intermédiaire d'un *système de fichiers*, lequel définit un ensemble de conditions et de principes selon lesquels organiser et manipuler des fichiers. Chaque système d'exploitation possède son système de fichier privilégié, même s'il peut en utiliser d'autres. Le tableau suivant donne quelques noms.

Tableau 2.4. Systèmes de fichiers de quelques systèmes d'exploitation

Système	Système de fichier
MS-DOS	FAT, aussi appelé FAT-16 (File Allocation Table 16 bits)
Windows 95/98	FAT32, extension du système FAT-16 (File Allocation Table 32 bits)
Windows NT et supérieur	NTFS (New Technology File System)
OS/2	HPFS (High Performance File System)
Linux	Ext4 (Fourth extended file system)

En interne une suite statique d'octets, un fichier est à plus haut niveau d'abstraction une collection d'informations numériques réunies sous un même nom, manipulées comme une unité, et dont c'est le système de fichiers qui caractérisent les propriétés définitoires. A ce titre, fichiers et systèmes de fichiers conditionnent l'accès à toutes les ressources logicielles du système d'exploitation, qui sont à l'exception de la ROM de démarrage, nécessairement situées au niveau d'un ou plusieurs dispositifs de stockage.

Le système de fichiers, ou système de gestion des fichiers, assure plusieurs fonctions :

- *Manipulation des fichiers* Le système de gestion de fichiers définit pour manipuler fichiers et répertoires un panel d'opérations les concernant, à savoir créer et détruire des fichiers et des répertoires, insérer, supprimer et modifier les données d'un fichier.

- *Allocation sur mémoires secondaires* Le système de fichiers gère l'allocation de l'espace disque aux fichiers et l'espace libre sur le disque dur. Il alloue à chaque fichier, dont la taille est dynamique, une certaine quantité de mémoire secondaire de taille fixe (blocs).

- *Localisation des fichiers* Le système de fichiers offre à l'utilisateur une vue abstraite sur ses données, lui permettant de les localiser à partir d'un ensemble d'informations descriptives (nom, adresse) réunies dans un chemin d'accès. Le chemin d'accès d'un fichier ou d'un répertoire est une chaîne de caractères décrivant la position de ce fichier ou répertoire dans le système de fichiers.

- *Sécurité des fichiers* Le système de fichiers détermine les différentes options en matière de protection et de contrôle des fichiers. Il permet de la sorte le partage des fichiers par différents programmes d'applications tout en assurant la sécurité et la confidentialité des données parmi les divers utilisateurs.

Services

Nous avons vu dans ce chapitre que le terme "services" sous Windows renvoyait potentiellement à une routine du système d'exploitation, un processus serveur ou un pilote de périphérique. Cette section va traiter des services qui sont des processus en mode utilisateur. Dans ce contexte, un service (ou service Windows) désigne un type de programme qui comprend un ou un ensemble de processus s'exécutant en arrière-plan plutôt que sous le contrôle direct d'un utilisateur. En principe, les services n'interagissent pas avec l'utilisateur connecté.

Les services ressemblent aux "démons" UNIX, en ce sens d'être démarrés souvent lors du chargement du système d'exploitation, et de servir en général à répondre à des requêtes du réseau, à l'activité du matériel ou à d'autres programmes en exécutant certaines tâches. Les services peuvent donc être configurés pour démarrer automatiquement, sans création préalable d'une session interactive. En variante, ils peuvent être lancés manuellement par l'utilisateur (par exemple, via l'outil d'administration Services) ou par un événement ayant besoin du service (événement qui sollicitera pour l'occasion la fonction Windows *StartService*).

Un service doit se conformer aux règles d'interface et aux protocoles du composant Gestionnaire de contrôle de service (SCM, Service Control Manager), chargé de démarrer, arrêter et gérer les processus de service. Les programmes de service sont en fait des images Windows dont le code est écrit de telle manière à pouvoir répondre aux messages et sollicitations du SCM, incluant des actions du genre inscription du démarrage du service, réponse aux requêtes d'état, suspension ou arrêt du service.

Les services sont rattachés à trois comptes d'utilisateur : le compte Système, le compte Service réseau et le compte Service local. Parce que les services sont associés à leurs propres comptes utilisateur dédiés, ils peuvent fonctionner sans qu'un utilisateur soit connecté au système d'exploitation. Pour plus de détails sur le contexte dans lequel les services sont exécutés, voir au chapitre x la section Comptes de service.

Un certain nombre de composants et fonctionnalités clé de Windows existent sous forme de services, par exemple le spouleur d'impression, le journal des événements, le planificateur de tâches, plus divers composants relatifs au réseau.

Espace de noms

Au premier lieu un moyen commode de lever une ambiguïté sur des termes qui pourraient sans cela être homonymes, le concept d'espace de noms fait dans la perspective des systèmes d'exploitation référence à un lieu abstrait conçu pour l'accueil et l'organisation de ressources de toute nature (mais le plus souvent apparentés), donnant de la sorte aux applications la possibilité d'identifier rapidement un élément parmi d'autres dans la hiérarchie ainsi formée.

Une illustration pour le moins parlante de ce que peut être un espace de noms est le système de fichiers, où les dossiers font dans ce contexte figure d'espace de noms pour les fichiers qu'ils hébergent. Par exemple, le fichier foo.txt peut exister dans plus d'un répertoire, mais deux copies de foo.txt ne peuvent pas coexister dans le même répertoire. De plus, pour faire référence audit fichier depuis l'extérieur du dossier qui le contient, il est nécessaire d'indiquer son chemin d'accès complet, tel que C:\Windows\foo.txt ou C:\Users\foo.txt.

Potentiellement, toute ressource formée à partir d'un sous-ensemble provenant d'un ensemble plus vaste peut être catégorisé selon le principe d'un espace de noms. Cela inclut les fichiers et les dossiers, dont nous avons

déjà parlé, mais également les collections de symboles dans un langage de programmation spécifique, les primitives de synchronisation (par exemple mutex, sémaphores, événements, etc.) et les régions de la mémoire partagée utilisées par les processus en cours, et bien d'autres.

API Windows

L'API (*Application Programming Interface*) Windows est l'interface programmatique de la famille des systèmes d'exploitation Microsoft Windows. Elle est conçue pour les langages de programmation C et C++ et est la manière privilégiée pour une application d'interagir avec le système d'exploitation.

Constituée de milliers de sous-routines appelables depuis le mode utilisateur - autant de points d'entrée vers les services moyen et bas niveau du système -, la couche logicielle Windows est probablement celle présentant le plus d'intérêt pour le plus grand nombre de développeurs, incluant autant ceux nouveaux venus dans l'écosystème Windows que ceux habitués des frameworks et des composants d'interface à l'abstraction plus marquée. Dans ce livre, où il nous arrivera de traiter moins d'elle que de certaines consœurs de plus bas niveau, c'est parce que nous envisageons l'API Windows seulement pour ses liens avec les autres composants fondamentaux du système ; et ce livre n'étant pas un manuel de programmation, jamais en dehors.

L'API Windows couvre un large éventail de fonctionnalités, allant des services de base comme la manipulation des processus et des threads (création et démantèlement), la communication entre programmes ou l'exploitation des réseaux informatiques, à d'autres plus avancés tel la cryptographie, l'application de la sécurité ou l'interaction avec des dispositifs matériels tiers. (C'est par exemple via la fonction Windows *DeviceIoControl* que le code mode utilisateur envoie une demande à un pilote de périphérique donné, laquelle amène le matériel lié au pilote à effectuer l'opération idoine.)

Les fonctions, structures de données et énumérations de l'API Windows sont pour la plupart documentées, décrites à cet effet dans les divers programmes Microsoft d'assistance aux développeurs, Platform SDK et MSDN notamment. Pour plus de détails, voyez msdn.microsoft.com. Vous trouverez sinon sur le net nombre de tutoriaux.

Historique

À ses débuts, comme le projet était destiné à l'origine comme remplaçant de OS/2 version 2, Windows NT offrait l'interface de programmation 32 bits de OS/2 Presentation Manager. Cependant, Windows 3.0 et le succès qu'on lui connait montrèrent la situation sous un éclairage neuf, et donnèrent un tour nouveau à la carrière de Windows NT, pressenti dès lors pour remplacer Windows au lieu de OS/2.

L'API Windows apportait nombre de fonctionnalités inconnues de Windows 3.1, mais Microsoft décida de rendre la nouvelle API compatible avec les noms de fonction et l'emploi des types de données Windows 16 bits. Pour mettre en exergue son adaptation aux processeurs 32 bits, tels que le Intel 80386 et ses successeurs, et la distinguer de la précédente interface inclue dans les éditions 16 bits (Windows 3.1 et ses prédécesseurs), Microsoft donna à la nouvelle API le nom Win32, renommant au passage l'ancienne en Win16.

A titre informatif, notez que si la conservation de la sémantique entre Win32 et Win16 était pour l'époque un bon moyen de faciliter le portage des applications Windows 16 bits vers Windows NT, elle fut et reste source de confusion. Si vous découvrez l'API Windows pour la première fois et que vous vous demandez pourquoi quelques noms de fonctions et d'interfaces semblent incohérents, la raison majeure vient de vous être expliquée.

Win64 est la version de l'API Windows pour les plateformes 64-bits. Sauf mention du contraire, les architectures de processeur 32 bits et 64 bits coexistant dans les matériels modernes, sauf mention explicite du contraire, Win32 et Win64 forment ensemble l'API Windows.

Composants de l'API Windows

Les fonctionnalités fournies par l'API Windows peuvent être rangées entre plusieurs catégories :

- *Services de base* Les services de base donnent accès aux ressources fondamentales du système informatique. Cela inclue par exemple les processus et les threads, le système de fichiers, les pilotes et les périphériques qu'ils desservent. Ils donnent également accès à certaines primitives clé du système d'exploitation, comme la gestion des conditions exceptionnelles pendant l'exécution d'un programme (système de gestion d'exceptions) ou la possibilité de gérer comme un tout un groupe de processus (jobs).

- *Services avancés* Là où les services de base se concentrent sur des entités ou des groupes (fichiers, répertoires et autres), les services avancés agissent sur le système dans son ensemble. Sont concernées des opérations spécifiques comme l'extinction de l'ordinateur, la création, le démarrage et l'arrêt des services, ou la manipulation dans les comptes d'utilisateurs.

- *Services graphiques* Les services graphiques offrent un tremplin vers les ressources logicielles faisant le transport de contenus graphique vers les divers périphériques de sortie appropriés, tels moniteurs et imprimantes.

- *Services d'interface utilisateur* Les services d'interface utilisateur permet d'afficher et de gérer les contrôles de base comme les boutons et barres de défilement, de recevoir les informations du clavier et de la souris et des fonctionnalités associées comme l'environnement graphique.

- *Services réseau* Les services réseau ouvrent l'accès aux diverses possibilités du système d'exploitation en matière de gestion de réseau.

Autres implémentations

Bien que l'API Windows soit soumise au Code de la Propriété Intellectuelle et aux droits d'auteur en particulier, quelques initiatives existent pour en proposer une version sur d'autres plateformes. C'est le cas par exemple de Wine qui émule une API compatible avec Win32 pour les systèmes d'exploitation à base UNIX. Un autre exemple est le système ReactOS.

- *Wine* Wine est l'acronyme récursif anglophone de "Wine Is Not an Emulator". Ce logiciel est une implémentation libre de l'interface de programmation Windows bâtie sur X et UNIX, c'est-à-dire qu'il permet d'utiliser sur des systèmes d'exploitation non-Microsoft, par exemple Linux ou Mac OS X, des programmes conçus pour fonctionner sous Windows. Il est constitué d'un chargeur de programmes ainsi que d'une bibliothèque d'émulation qui permet l'exécution d'applications Windows sous Unix. Le chargeur de programme va exécuter le binaire d'une application alors que la bibliothèque va servir d'interface entre les appels de fonctions Windows et le système Unix (et l'environnement graphique X).

- *ReactOS* ReactOS est un projet de système d'exploitation en développement se voulant compatible avec les programmes et pilotes Microsoft Windows. L'objectif du projet, tel que cité par lui-même, est de fournir un système d'exploitation gratuit et entièrement libre basé sur l'architecture de Microsoft Windows, et devenir une alternative au système d'exploitation dominant le marché actuel. Oeuvrant de concert avec le projet WINE, ReactOS peut donc bénéficier des progrès de Wine dans l'implémentation de l'API Windows. Ces travaux concernent principalement les bibliothèques logicielles, dont la plupart peuvent être échangées entre ReactOS et Wine.

API Native

Si la conception de programmes Windows sous-entend généralement l'utilisation de l'API du même nom, cette dernière n'est en réalité pas tellement proche du coeur du système d'exploitation, lequel se base en l'occurrence sur un autre sous-ensemble d'interfaces, la véritable API système de Windows, dite Native (remarquez la lettre majuscule).

L'API Native est utilisée principalement lors de l'amorçage du système, stade où les autres composants de Windows ne sont pas encore disponibles (ils ne sont sont pas encore chargés en mémoire), puis tout au long du cycle de vue du système par les sous-systèmes, les DLL de sous-système et autres images natives. En plus de

cela, l'API Native implémente le code de diffusion de service système pour les services système en mode noyau - autrement dit les appels système.

Quelques processus fondamentaux du système d'exploitation, dont le sous-système d'environnement (Csrss.exe), sont implémentés en utilisant l'API Native. Pour l'utilisateur, l'exemple le plus visible d'une application native est le programme autochk, lequel planifie l'exécution du vérificateur de système de fichiers lors du prochain redémarrage du système.

La plupart des capacités des interfaces de l'API Native sont accessibles via les bibliothèques de l'API Windows principale. Ainsi, NtCreateProcess est le service système interne que la fonction Windows CreateProcess appelle pour créer un nouveau processus. Néanmoins, toutes les fonctions de l'API Native ne sont pas exposées, certaines sont en effet réservées à usage interne du système d'exploitation, et permettent des choses impossibles à réaliser via l'utilisation des API classiques, comme, par exemple, obtenir la liste des handles ouverts d'un processus ou définir les paramètres étendus d'un fichier.

Seule une mince portion du contenu de l'API Native est documentée - le kit de développement pour pilotes laisse place à une petite quantité de description et la Base de connaissances Microsoft à quelques timides évocations. Notez également que la dénomination « API native » n'a rien d'officiel. Il s'agit cependant d'une sorte de consensus dans le monde de la programmation Windows.

Les interfaces et structures de données de l'API Native sont implantés au niveau du module de support système Ntdll.dll, sur lequel nous reviendrons au prochain chapitre.

API noyau

Les différents pensionnaires de l'espace système sous Windows, y compris le noyau et les composants de l'exécutif, définissent relativement aux technologies prises en charge (ordonnancement, processus, mémoire, entrées-sorties, et ainsi de suite) un vaste ensemble de sous-routines, appelables par nature exclusivement à partir du mode noyau, et dont l'ensemble sert en l'occurrence de façade par laquelle le système d'exploitation offre des services à tous les logiciels, incluant les pilotes et autres codes noyau, mais aussi indirectement les applications.

Une des grandes forces de l'API noyau de Windows, par ailleurs la plus essentielle de toutes au regard des aspects fonctionnels, est indéniablement sa stabilité. Cela permet ainsi aux développeurs de concevoir du code qui est portable entre différentes versions de noyaux.

Par contraste avec les interfaces définies au niveau de l'API Windows, disséminées entre plusieurs bibliothèques (Kernel32.dll, User32.dll, etc.) et dont les noms n'évoquent pas les caractéristiques internes, les fonctions rendues visibles sur le plan noyau le sont par l'intermédiaire d'une gamme réduite de support, à savoir Ntoskrnl.exe (ou équivalent) et Hal.dll, et ont leur noms régis par une convention fortement influencée par le caractère modulaire du système d'exploitation.

Les noms des routines système suivent une convention de nommage mettant en valeur l'appartenance de chaque symbole à tel ou tel composant. Par exemple, le préfixe *Io* représente les fonctions du gestionnaire d'E/S, *Ke* les interfaces du noyau, *Ex* les routines de support de l'exécutif, et ainsi de suite. Le tableau qui vient énumère les préfixes les plus couramment employés en la matière.

Tableau 2.5. Préfixes pour interfaces en mode noyau

Préfixe	Composant
Alpc	Système (A)LPC
Bvga	Boot Video (VGA)
Cc	Gestionnaire de cache
Cm	Gestionnaire de configuration
Em	Gestionnaire d'errata
Etw	Event Tracing for Windows

Préfixe	Composant
Ex	Routines de l'exécutif
ExpWnf	Windows Notification Framework
FsRtl	Bibliothèque d'exécution du pilote de système de fichiers
Hal	Couche d'abstraction matérielle
Io	Gestionnaire d'E/S
Ke	Débogueur noyau
Ke	Noyau
Ldr	Chargeur de modules
Lpc	Appel de procédure locale
Mm	Gestionnaire mémoire
Nt	Services système
Ob	Gestionnaire d'objets
Pcw	Performance Counter for Windows
Pf	Prefetcher logique
Po	Gestionnaire d'alimentation
Pp	Gestionnaire PnP
Ps	Gestionnaire de processus
Rtl	Bibliothèque d'exécution
Sdb	Base de données Shim
Se	Sécurité
Tm	Gestionnaire de transaction
Vf	Driver Verifier
Wdi	Windows Diagnostic Infrastructure
Wer	Windows Error Reporting
Whea	Windows Hardware Error Architecture
Wmi	Windows Management Instrumentation
Zw	Appel système

En marge du schéma précédemment décrit, les principaux composants de l'exécutif utilisent une variation du préfixe pour désigner les fonctions internes : soit la première lettre du préfixe suivie d'un *i* (comme *interne*), soit le préfixe complet suivi de la lettre *p* (comme *privé*). Ainsi, *Ki* représente les fonctions internes du noyau et *Psp* les fonctions internes de support du gestionnaire de processus.

Valeurs NTSTATUS

Une large majorité des routines exécutés en mode noyau, de sorte à rendre compte de la réussite ou de l'échec d'une opération, emploient le type prédéfini `NTSTATUS`, concrètement un entier 32 bits signé. En règle générale, une valeur égale à 0 signifie que l'action souhaitée a pris fin avec succès, tandis qu'une valeur différente du chiffe susdit met en évidence une erreur.

Les valeurs NTSTATUS sont divisées en quatre grandes catégories : les valeurs de réussite (intervalle 0 - 0x3FFFFFFF), les valeurs d'information (0x40000000 – 0x7FFFFFFF), les avertissements (0x80000000 – 0xBFFFFFFF) et les valeurs d'erreur (0xC0000000 - 0xFFFFFFFF). Plusieurs macros sont définies relativement à ces aspects.

- **NT_SUCCESS** S'assure qu'une valeur donnée s'apparente à un état ou revêt un caractère informationnel.
- **NT_INFORMATION** S'assure qu'une valeur donnée fait référence à un résultat d'opération, qui plutôt que de dénoter une réussite ou un échec proprement dit, apporte des informations supplémentaires sur la manière dont l'action envisagée s'est accomplie jusque-là.

- **NT_WARNING** S'assure qu'une valeur donnée résulte d'un incident qui, sans extrême gravité apparente, nécessite toutefois une certaine attention ou action.

- **NT_ERROR** S'assure qu'une valeur donnée corresponde à une erreur sévère.

Dans l'éventualité où un code NTSTATUS doit être relayée à un composant mode utilisateur, la valeur concernée est pour l'occasion convertie en un autre format, notée `ERROR_xxx` au niveau de l'API Windows (voir fonction GetLastError).

Status	Valeur
STATUS_SUCCESS	0x0
STATUS_WAIT_1	0x1
STATUS_WAIT_2	0x2
STATUS_WAIT_3	0x3
STATUS_WAIT_63	0x3F
STATUS_ABANDONED	0x80
STATUS_ABANDONED_WAIT_0	0x80
STATUS_ABANDONED_WAIT_63	0xBF
STATUS_USER_APC	0xC0
STATUS_KERNEL_APC	0x100
STATUS_ALERTED	0x101
STATUS_TIMEOUT	0x102
STATUS_PENDING	0x103
STATUS_REPARSE	0x104
STATUS_MORE_ENTRIES	0x105
STATUS_NOT_ALL_ASSIGNED	0x106
STATUS_SOME_NOT_MAPPED	0x107
STATUS_OPLOCK_BREAK_IN_PROGRESS	0x108
STATUS_VOLUME_MOUNTED	0x109
STATUS_RXACT_COMMITTED	0x10A
STATUS_NOTIFY_CLEANUP	0x10B
STATUS_NOTIFY_ENUM_DIR	0x10C
STATUS_NO_QUOTAS_FOR_ACCOUNT	0x10D
STATUS_PAGE_FAULT_TRANSITION	0x110
STATUS_PAGE_FAULT_DEMAND_ZERO	0x111
STATUS_PAGE_FAULT_COPY_ON_WRITE	0x112
STATUS_PAGE_FAULT_GUARD_PAGE	0x113
STATUS_PAGE_FAULT_PAGING_FILE	0x114
STATUS_CACHE_PAGE_LOCKED	0x115
STATUS_CRASH_DUMP	0x116
STATUS_BUFFER_ALL_ZEROS	0x117
STATUS_REPARSE_OBJECT	0x118
STATUS_TRANSLATION_COMPLETE	0x120
STATUS_NOTHING_TO_TERMINATE	0x122
STATUS_PROCESS_NOT_IN_JOB	0x123
STATUS_PROCESS_IN_JOB	0x124
STATUS_VOLSNAP_HIBERNATE_READY	0x125
STATUS_PROCESS_CLONED	0x129
STATUS_FILE_LOCKED_WITH_WRITERS	0x12B
STATUS_VALID_IMAGE_HASH	0x12C

Status	Valeur
STATUS_RING_PREVIOUSLY_EMPTY	0x210
STATUS_RING_PREVIOUSLY_FULL	0x211
STATUS_RING_NEWLY_EMPTY	0x213
STATUS_OPLOCK_HANDLE_CLOSED	0x216
STATUS_WAIT_FOR_OPLOCK	0x367
DBG_EXCEPTION_HANDLED	0x10001
DBG_CONTINUE	0x10002
STATUS_FLT_IO_COMPLETE	0x1C0001
STATUS_DIS_ATTRIBUTE_BUILT	0x3C0001
STATUS_OBJECT_NAME_EXISTS	0x40000000
STATUS_THREAD_WAS_SUSPENDED	0x40000001
STATUS_WORKING_SET_LIMIT_RANGE	0x40000002
STATUS_IMAGE_NOT_AT_BASE	0x40000003
STATUS_RXACT_STATE_CREATED	0x40000004
STATUS_SEGMENT_NOTIFICATION	0x40000005
STATUS_LOCAL_USER_SESSION_KEY	0x40000006
STATUS_BAD_CURRENT_DIRECTORY	0x40000007
STATUS_SERIAL_MORE_WRITES	0x40000008
STATUS_REGISTRY_RECOVERED	0x40000009
STATUS_FT_WRITE_RECOVERY	0x4000000B
STATUS_SERIAL_COUNTER_TIMEOUT	0x4000000C
STATUS_NULL_LM_PASSWORD	0x4000000D
STATUS_RECEIVE_PARTIAL	0x4000000F
STATUS_RECEIVE_EXPEDITED	0x40000010
STATUS_EVENT_DONE	0x40000012
STATUS_EVENT_PENDING	0x40000013
STATUS_CHECKING_FILE_SYSTEM	0x40000014
STATUS_FATAL_APP_EXIT	0x40000015
STATUS_PREDEFINED_HANDLE	0x40000016
STATUS_WAS_UNLOCKED	0x40000017
STATUS_SERVICE_NOTIFICATION	0x40000018
STATUS_WAS_LOCKED	0x40000019
STATUS_ALREADY_WIN32	0x4000001B
STATUS_WX86_UNSIMULATE	0x4000001C
STATUS_WX86_CONTINUE	0x4000001D
STATUS_WX86_SINGLE_STEP	0x4000001E
STATUS_WX86_BREAKPOINT	0x4000001F
STATUS_WX86_EXCEPTION_CONTINUE	0x40000020
STATUS_WX86_EXCEPTION_CHAIN	0x40000022
STATUS_NO_YIELD_PERFORMED	0x40000024
STATUS_TIMER_RESUME_IGNORED	0x40000025
STATUS_ARBITRATION_UNHANDLED	0x40000026
STATUS_CARDBUS_NOT_SUPPORTED	0x40000027
STATUS_WX86_CREATEWX86TIB	0x40000028
STATUS_MP_PROCESSOR_MISMATCH	0x40000029
STATUS_HIBERNATED	0x4000002A
STATUS_RESUME_HIBERNATION	0x4000002B

Status	Valeur
STATUS_MESSAGE_RETRIEVED	0x4000002E
STATUS_ABANDON_HIBERFILE	0x40000033
STATUS_BIZRULES_NOT_ENABLED	0x40000034
STATUS_FT_READ_FROM_COPY	0x40000035
STATUS_IMAGE_AT_DIFFERENT_BASE	0x40000036
DBG_REPLY_LATER	0x40010001
DBG_UNABLE_TO_PROVIDE_HANDLE	0x40010002
DBG_TERMINATE_THREAD	0x40010003
DBG_TERMINATE_PROCESS	0x40010004
DBG_CONTROL_C	0x40010005
DBG_PRINTEXCEPTION_C	0x40010006
DBG_RIPEXCEPTION	0x40010007
DBG_CONTROL_BREAK	0x40010008
DBG_COMMAND_EXCEPTION	0x40010009
STATUS_GUARD_PAGE_VIOLATION	0x80000001
STATUS_DATATYPE_MISALIGNMENT	0x80000002
STATUS_BREAKPOINT	0x80000003
STATUS_SINGLE_STEP	0x80000004
STATUS_BUFFER_OVERFLOW	0x80000005
STATUS_NO_MORE_FILES	0x80000006
STATUS_WAKE_SYSTEM_DEBUGGER	0x80000007
STATUS_HANDLES_CLOSED	0x8000000A
STATUS_NO_INHERITANCE	0x8000000B
STATUS_GUID_SUBSTITUTION_MADE	0x8000000C
STATUS_PARTIAL_COPY	0x8000000D
STATUS_DEVICE_PAPER_EMPTY	0x8000000E
STATUS_DEVICE_POWERED_OFF	0x8000000F
STATUS_DEVICE_OFF_LINE	0x80000010
STATUS_DEVICE_BUSY	0x80000011
STATUS_NO_MORE_EAS	0x80000012
STATUS_INVALID_EA_NAME	0x80000013
STATUS_EA_LIST_INCONSISTENT	0x80000014
STATUS_INVALID_EA_FLAG	0x80000015
STATUS_VERIFY_REQUIRED	0x80000016
STATUS_EXTRANEOUS_INFORMATION	0x80000017
STATUS_RXACT_COMMIT_NECESSARY	0x80000018
STATUS_NO_MORE_ENTRIES	0x8000001A
STATUS_FILEMARK_DETECTED	0x8000001B
STATUS_MEDIA_CHANGED	0x8000001C
STATUS_BUS_RESET	0x8000001D
STATUS_END_OF_MEDIA	0x8000001E
STATUS_BEGINNING_OF_MEDIA	0x8000001F
STATUS_MEDIA_CHECK	0x80000020
STATUS_SETMARK_DETECTED	0x80000021
STATUS_NO_DATA_DETECTED	0x80000022
STATUS_SERVER_HAS_OPEN_HANDLES	0x80000024
STATUS_ALREADY_DISCONNECTED	0x80000025

Status	Valeur
STATUS_LONGJUMP	0x80000026
STATUS_PLUGPLAY_QUERY_VETOED	0x80000028
STATUS_UNWIND_CONSOLIDATE	0x80000029
STATUS_REGISTRY_HIVE_RECOVERED	0x8000002A
STATUS_DLL_MIGHT_BE_INSECURE	0x8000002B
STATUS_STOPPED_ON_SYMLINK	0x8000002D
STATUS_NO_ACE_CONDITION	0x8000002F
DBG_EXCEPTION_NOT_HANDLED	0x80010001
STATUS_CLUSTER_NODE_ALREADY_UP	0x80130001
STATUS_FLT_BUFFER_TOO_SMALL	0x801C0001
STATUS_FVE_PARTIAL_METADATA	0x80210001
STATUS_FVE_TRANSIENT_STATE	0x80210002
STATUS_UNSUCCESSFUL	0xC0000001
STATUS_NOT_IMPLEMENTED	0xC0000002
STATUS_INVALID_INFO_CLASS	0xC0000003
STATUS_INFO_LENGTH_MISMATCH	0xC0000004
STATUS_ACCESS_VIOLATION	0xC0000005
STATUS_IN_PAGE_ERROR	0xC0000006
STATUS_PAGEFILE_QUOTA	0xC0000007
STATUS_INVALID_HANDLE	0xC0000008
STATUS_BAD_INITIAL_STACK	0xC0000009
STATUS_BAD_INITIAL_PC	0xC000000A
STATUS_INVALID_CID	0xC000000B
STATUS_TIMER_NOT_CANCELED	0xC000000C
STATUS_INVALID_PARAMETER	0xC000000D
STATUS_NO_SUCH_DEVICE	0xC000000E
STATUS_NO_SUCH_FILE	0xC000000F
STATUS_INVALID_DEVICE_REQUEST	0xC0000010
STATUS_END_OF_FILE	0xC0000011
STATUS_WRONG_VOLUME	0xC0000012
STATUS_NO_MEDIA_IN_DEVICE	0xC0000013
STATUS_UNRECOGNIZED_MEDIA	0xC0000014
STATUS_NONEXISTENT_SECTOR	0xC0000015
STATUS_MORE_PROCESSING_REQUIRED	0xC0000016
STATUS_NO_MEMORY	0xC0000017
STATUS_CONFLICTING_ADDRESSES	0xC0000018
STATUS_NOT_MAPPED_VIEW	0xC0000019
STATUS_UNABLE_TO_FREE_VM	0xC000001A
STATUS_UNABLE_TO_DELETE_SECTION	0xC000001B
STATUS_INVALID_SYSTEM_SERVICE	0xC000001C
STATUS_ILLEGAL_INSTRUCTION	0xC000001D
STATUS_INVALID_LOCK_SEQUENCE	0xC000001E
STATUS_INVALID_VIEW_SIZE	0xC000001F
STATUS_INVALID_FILE_FOR_SECTION	0xC0000020
STATUS_ALREADY_COMMITTED	0xC0000021
STATUS_ACCESS_DENIED	0xC0000022
STATUS_BUFFER_TOO_SMALL	0xC0000023

Status	Valeur
STATUS_OBJECT_TYPE_MISMATCH	0xC0000024
STATUS_NONCONTINUABLE_EXCEPTION	0xC0000025
STATUS_INVALID_DISPOSITION	0xC0000026
STATUS_UNWIND	0xC0000027
STATUS_BAD_STACK	0xC0000028
STATUS_INVALID_UNWIND_TARGET	0xC0000029
STATUS_NOT_LOCKED	0xC000002A
STATUS_PARITY_ERROR	0xC000002B
STATUS_UNABLE_TO_DECOMMIT_VM	0xC000002C
STATUS_NOT_COMMITTED	0xC000002D
STATUS_INVALID_PORT_ATTRIBUTES	0xC000002E
STATUS_PORT_MESSAGE_TOO_LONG	0xC000002F
STATUS_INVALID_PARAMETER_MIX	0xC0000030
STATUS_INVALID_QUOTA_LOWER	0xC0000031
STATUS_DISK_CORRUPT_ERROR	0xC0000032
STATUS_OBJECT_NAME_INVALID	0xC0000033
STATUS_OBJECT_NAME_NOT_FOUND	0xC0000034
STATUS_OBJECT_NAME_COLLISION	0xC0000035
STATUS_PORT_DISCONNECTED	0xC0000037
STATUS_DEVICE_ALREADY_ATTACHED	0xC0000038
STATUS_OBJECT_PATH_INVALID	0xC0000039
STATUS_OBJECT_PATH_NOT_FOUND	0xC000003A
STATUS_OBJECT_PATH_SYNTAX_BAD	0xC000003B
STATUS_DATA_OVERRUN	0xC000003C
STATUS_DATA_LATE_ERROR	0xC000003D
STATUS_DATA_ERROR	0xC000003E
STATUS_CRC_ERROR	0xC000003F
STATUS_SECTION_TOO_BIG	0xC0000040
STATUS_PORT_CONNECTION_REFUSED	0xC0000041
STATUS_INVALID_PORT_HANDLE	0xC0000042
STATUS_SHARING_VIOLATION	0xC0000043
STATUS_QUOTA_EXCEEDED	0xC0000044
STATUS_INVALID_PAGE_PROTECTION	0xC0000045
STATUS_MUTANT_NOT_OWNED	0xC0000046
STATUS_SEMAPHORE_LIMIT_EXCEEDED	0xC0000047
STATUS_PORT_ALREADY_SET	0xC0000048
STATUS_SECTION_NOT_IMAGE	0xC0000049
STATUS_SUSPEND_COUNT_EXCEEDED	0xC000004A
STATUS_THREAD_IS_TERMINATING	0xC000004B
STATUS_BAD_WORKING_SET_LIMIT	0xC000004C
STATUS_INCOMPATIBLE_FILE_MAP	0xC000004D
STATUS_SECTION_PROTECTION	0xC000004E
STATUS_EAS_NOT_SUPPORTED	0xC000004F
STATUS_EA_TOO_LARGE	0xC0000050
STATUS_NONEXISTENT_EA_ENTRY	0xC0000051
STATUS_NO_EAS_ON_FILE	0xC0000052
STATUS_EA_CORRUPT_ERROR	0xC0000053

Status	Valeur
STATUS_FILE_LOCK_CONFLICT	0xC0000054
STATUS_LOCK_NOT_GRANTED	0xC0000055
STATUS_DELETE_PENDING	0xC0000056
STATUS_CTL_FILE_NOT_SUPPORTED	0xC0000057
STATUS_UNKNOWN_REVISION	0xC0000058
STATUS_REVISION_MISMATCH	0xC0000059
STATUS_INVALID_OWNER	0xC000005A
STATUS_INVALID_PRIMARY_GROUP	0xC000005B
STATUS_NO_IMPERSONATION_TOKEN	0xC000005C
STATUS_CANT_DISABLE_MANDATORY	0xC000005D
STATUS_NO_LOGON_SERVERS	0xC000005E
STATUS_NO_SUCH_LOGON_SESSION	0xC000005F
STATUS_NO_SUCH_PRIVILEGE	0xC0000060
STATUS_PRIVILEGE_NOT_HELD	0xC0000061
STATUS_INVALID_ACCOUNT_NAME	0xC0000062
STATUS_USER_EXISTS	0xC0000063
STATUS_NO_SUCH_USER	0xC0000064
STATUS_GROUP_EXISTS	0xC0000065
STATUS_NO_SUCH_GROUP	0xC0000066
STATUS_MEMBER_IN_GROUP	0xC0000067
STATUS_MEMBER_NOT_IN_GROUP	0xC0000068
STATUS_LAST_ADMIN	0xC0000069
STATUS_WRONG_PASSWORD	0xC000006A
STATUS_ILL_FORMED_PASSWORD	0xC000006B
STATUS_PASSWORD_RESTRICTION	0xC000006C
STATUS_LOGON_FAILURE	0xC000006D
STATUS_ACCOUNT_RESTRICTION	0xC000006E
STATUS_INVALID_LOGON_HOURS	0xC000006F
STATUS_INVALID_WORKSTATION	0xC0000070
STATUS_PASSWORD_EXPIRED	0xC0000071
STATUS_ACCOUNT_DISABLED	0xC0000072
STATUS_NONE_MAPPED	0xC0000073
STATUS_TOO_MANY_LUIDS_REQUESTED	0xC0000074
STATUS_LUIDS_EXHAUSTED	0xC0000075
STATUS_INVALID_SUB_AUTHORITY	0xC0000076
STATUS_INVALID_ACL	0xC0000077
STATUS_INVALID_SID	0xC0000078
STATUS_INVALID_SECURITY_DESCR	0xC0000079
STATUS_PROCEDURE_NOT_FOUND	0xC000007A
STATUS_INVALID_IMAGE_FORMAT	0xC000007B
STATUS_NO_TOKEN	0xC000007C
STATUS_BAD_INHERITANCE_ACL	0xC000007D
STATUS_RANGE_NOT_LOCKED	0xC000007E
STATUS_DISK_FULL	0xC000007F
STATUS_SERVER_DISABLED	0xC0000080
STATUS_SERVER_NOT_DISABLED	0xC0000081
STATUS_TOO_MANY_GUIDS_REQUESTED	0xC0000082

Status	Valeur
STATUS_GUIDS_EXHAUSTED	0xC0000083
STATUS_INVALID_ID_AUTHORITY	0xC0000084
STATUS_AGENTS_EXHAUSTED	0xC0000085
STATUS_INVALID_VOLUME_LABEL	0xC0000086
STATUS_SECTION_NOT_EXTENDED	0xC0000087
STATUS_NOT_MAPPED_DATA	0xC0000088
STATUS_RESOURCE_DATA_NOT_FOUND	0xC0000089
STATUS_RESOURCE_TYPE_NOT_FOUND	0xC000008A
STATUS_RESOURCE_NAME_NOT_FOUND	0xC000008B
STATUS_ARRAY_BOUNDS_EXCEEDED	0xC000008C
STATUS_FLOAT_DENORMAL_OPERAND	0xC000008D
STATUS_FLOAT_DIVIDE_BY_ZERO	0xC000008E
STATUS_FLOAT_INEXACT_RESULT	0xC000008F
STATUS_FLOAT_INVALID_OPERATION	0xC0000090
STATUS_FLOAT_OVERFLOW	0xC0000091
STATUS_FLOAT_STACK_CHECK	0xC0000092
STATUS_FLOAT_UNDERFLOW	0xC0000093
STATUS_INTEGER_DIVIDE_BY_ZERO	0xC0000094
STATUS_INTEGER_OVERFLOW	0xC0000095
STATUS_PRIVILEGED_INSTRUCTION	0xC0000096
STATUS_TOO_MANY_PAGING_FILES	0xC0000097
STATUS_FILE_INVALID	0xC0000098
STATUS_ALLOTTED_SPACE_EXCEEDED	0xC0000099
STATUS_INSUFFICIENT_RESOURCES	0xC000009A
STATUS_DFS_EXIT_PATH_FOUND	0xC000009B
STATUS_DEVICE_DATA_ERROR	0xC000009C
STATUS_DEVICE_NOT_CONNECTED	0xC000009D
STATUS_DEVICE_POWER_FAILURE	0xC000009E
STATUS_FREE_VM_NOT_AT_BASE	0xC000009F
STATUS_MEMORY_NOT_ALLOCATED	0xC00000A0
STATUS_WORKING_SET_QUOTA	0xC00000A1
STATUS_MEDIA_WRITE_PROTECTED	0xC00000A2
STATUS_DEVICE_NOT_READY	0xC00000A3
STATUS_INVALID_GROUP_ATTRIBUTES	0xC00000A4
STATUS_BAD_IMPERSONATION_LEVEL	0xC00000A5
STATUS_CANT_OPEN_ANONYMOUS	0xC00000A6
STATUS_BAD_VALIDATION_CLASS	0xC00000A7
STATUS_BAD_TOKEN_TYPE	0xC00000A8
STATUS_BAD_MASTER_BOOT_RECORD	0xC00000A9
STATUS_INSTRUCTION_MISALIGNMENT	0xC00000AA
STATUS_INSTANCE_NOT_AVAILABLE	0xC00000AB
STATUS_PIPE_NOT_AVAILABLE	0xC00000AC
STATUS_INVALID_PIPE_STATE	0xC00000AD
STATUS_PIPE_BUSY	0xC00000AE
STATUS_ILLEGAL_FUNCTION	0xC00000AF
STATUS_PIPE_DISCONNECTED	0xC00000B0
STATUS_PIPE_CLOSING	0xC00000B1

Status	Valeur
STATUS_PIPE_CONNECTED	0xC00000B2
STATUS_PIPE_LISTENING	0xC00000B3
STATUS_INVALID_READ_MODE	0xC00000B4
STATUS_IO_TIMEOUT	0xC00000B5
STATUS_FILE_FORCED_CLOSED	0xC00000B6
STATUS_PROFILING_NOT_STARTED	0xC00000B7
STATUS_PROFILING_NOT_STOPPED	0xC00000B8
STATUS_COULD_NOT_INTERPRET	0xC00000B9
STATUS_FILE_IS_A_DIRECTORY	0xC00000BA
STATUS_NOT_SUPPORTED	0xC00000BB
STATUS_REMOTE_NOT_LISTENING	0xC00000BC
STATUS_DUPLICATE_NAME	0xC00000BD
STATUS_BAD_NETWORK_PATH	0xC00000BE
STATUS_NETWORK_BUSY	0xC00000BF
STATUS_DEVICE_DOES_NOT_EXIST	0xC00000C0
STATUS_TOO_MANY_COMMANDS	0xC00000C1
STATUS_ADAPTER_HARDWARE_ERROR	0xC00000C2
STATUS_INVALID_NETWORK_RESPONSE	0xC00000C3
STATUS_UNEXPECTED_NETWORK_ERROR	0xC00000C4
STATUS_BAD_REMOTE_ADAPTER	0xC00000C5
STATUS_PRINT_QUEUE_FULL	0xC00000C6
STATUS_NO_SPOOL_SPACE	0xC00000C7
STATUS_PRINT_CANCELLED	0xC00000C8
STATUS_NETWORK_NAME_DELETED	0xC00000C9
STATUS_NETWORK_ACCESS_DENIED	0xC00000CA
STATUS_BAD_DEVICE_TYPE	0xC00000CB
STATUS_BAD_NETWORK_NAME	0xC00000CC
STATUS_TOO_MANY_NAMES	0xC00000CD
STATUS_TOO_MANY_SESSIONS	0xC00000CE
STATUS_SHARING_PAUSED	0xC00000CF
STATUS_REQUEST_NOT_ACCEPTED	0xC00000D0
STATUS_REDIRECTOR_PAUSED	0xC00000D1
STATUS_NET_WRITE_FAULT	0xC00000D2
STATUS_PROFILING_AT_LIMIT	0xC00000D3
STATUS_NOT_SAME_DEVICE	0xC00000D4
STATUS_FILE_RENAMED	0xC00000D5
STATUS_VIRTUAL_CIRCUIT_CLOSED	0xC00000D6
STATUS_NO_SECURITY_ON_OBJECT	0xC00000D7
STATUS_CANT_WAIT	0xC00000D8
STATUS_PIPE_EMPTY	0xC00000D9
STATUS_CANT_ACCESS_DOMAIN_INFO	0xC00000DA
STATUS_CANT_TERMINATE_SELF	0xC00000DB
STATUS_INVALID_SERVER_STATE	0xC00000DC
STATUS_INVALID_DOMAIN_STATE	0xC00000DD
STATUS_INVALID_DOMAIN_ROLE	0xC00000DE
STATUS_NO_SUCH_DOMAIN	0xC00000DF
STATUS_DOMAIN_EXISTS	0xC00000E0

Status	Valeur
STATUS_DOMAIN_LIMIT_EXCEEDED	0xC00000E1
STATUS_OPLOCK_NOT_GRANTED	0xC00000E2
STATUS_INVALID_OPLOCK_PROTOCOL	0xC00000E3
STATUS_INTERNAL_DB_CORRUPTION	0xC00000E4
STATUS_INTERNAL_ERROR	0xC00000E5
STATUS_GENERIC_NOT_MAPPED	0xC00000E6
STATUS_BAD_DESCRIPTOR_FORMAT	0xC00000E7
STATUS_INVALID_USER_BUFFER	0xC00000E8
STATUS_UNEXPECTED_IO_ERROR	0xC00000E9
STATUS_UNEXPECTED_MM_CREATE_ERR	0xC00000EA
STATUS_UNEXPECTED_MM_MAP_ERROR	0xC00000EB
STATUS_UNEXPECTED_MM_EXTEND_ERR	0xC00000EC
STATUS_NOT_LOGON_PROCESS	0xC00000ED
STATUS_LOGON_SESSION_EXISTS	0xC00000EE
STATUS_INVALID_PARAMETER_1	0xC00000EF
STATUS_INVALID_PARAMETER_2	0xC00000F0
STATUS_INVALID_PARAMETER_3	0xC00000F1
STATUS_INVALID_PARAMETER_4	0xC00000F2
STATUS_INVALID_PARAMETER_5	0xC00000F3
STATUS_INVALID_PARAMETER_6	0xC00000F4
STATUS_INVALID_PARAMETER_7	0xC00000F5
STATUS_INVALID_PARAMETER_8	0xC00000F6
STATUS_INVALID_PARAMETER_9	0xC00000F7
STATUS_INVALID_PARAMETER_10	0xC00000F8
STATUS_INVALID_PARAMETER_11	0xC00000F9
STATUS_INVALID_PARAMETER_12	0xC00000FA
STATUS_REDIRECTOR_NOT_STARTED	0xC00000FB
STATUS_REDIRECTOR_STARTED	0xC00000FC
STATUS_STACK_OVERFLOW	0xC00000FD
STATUS_NO_SUCH_PACKAGE	0xC00000FE
STATUS_BAD_FUNCTION_TABLE	0xC00000FF
STATUS_VARIABLE_NOT_FOUND	0xC0000100
STATUS_DIRECTORY_NOT_EMPTY	0xC0000101
STATUS_FILE_CORRUPT_ERROR	0xC0000102
STATUS_NOT_A_DIRECTORY	0xC0000103
STATUS_BAD_LOGON_SESSION_STATE	0xC0000104
STATUS_LOGON_SESSION_COLLISION	0xC0000105
STATUS_NAME_TOO_LONG	0xC0000106
STATUS_FILES_OPEN	0xC0000107
STATUS_CONNECTION_IN_USE	0xC0000108
STATUS_MESSAGE_NOT_FOUND	0xC0000109
STATUS_PROCESS_IS_TERMINATING	0xC000010A
STATUS_INVALID_LOGON_TYPE	0xC000010B
STATUS_NO_GUID_TRANSLATION	0xC000010C
STATUS_CANNOT_IMPERSONATE	0xC000010D
STATUS_IMAGE_ALREADY_LOADED	0xC000010E
STATUS_ABIOS_NOT_PRESENT	0xC000010F

Status	Valeur
STATUS_ABIOS_LID_NOT_EXIST	0xC0000110
STATUS_ABIOS_LID_ALREADY_OWNED	0xC0000111
STATUS_ABIOS_NOT_LID_OWNER	0xC0000112
STATUS_ABIOS_INVALID_COMMAND	0xC0000113
STATUS_ABIOS_INVALID_LID	0xC0000114
STATUS_ABIOS_INVALID_SELECTOR	0xC0000116
STATUS_NO_LDT	0xC0000117
STATUS_INVALID_LDT_SIZE	0xC0000118
STATUS_INVALID_LDT_OFFSET	0xC0000119
STATUS_INVALID_LDT_DESCRIPTOR	0xC000011A
STATUS_INVALID_IMAGE_NE_FORMAT	0xC000011B
STATUS_RXACT_INVALID_STATE	0xC000011C
STATUS_RXACT_COMMIT_FAILURE	0xC000011D
STATUS_MAPPED_FILE_SIZE_ZERO	0xC000011E
STATUS_TOO_MANY_OPENED_FILES	0xC000011F
STATUS_CANCELLED	0xC0000120
STATUS_CANNOT_DELETE	0xC0000121
STATUS_INVALID_COMPUTER_NAME	0xC0000122
STATUS_FILE_DELETED	0xC0000123
STATUS_SPECIAL_ACCOUNT	0xC0000124
STATUS_SPECIAL_GROUP	0xC0000125
STATUS_SPECIAL_USER	0xC0000126
STATUS_MEMBERS_PRIMARY_GROUP	0xC0000127
STATUS_FILE_CLOSED	0xC0000128
STATUS_TOO_MANY_THREADS	0xC0000129
STATUS_THREAD_NOT_IN_PROCESS	0xC000012A
STATUS_TOKEN_ALREADY_IN_USE	0xC000012B
STATUS_PAGEFILE_QUOTA_EXCEEDED	0xC000012C
STATUS_COMMITMENT_LIMIT	0xC000012D
STATUS_INVALID_IMAGE_LE_FORMAT	0xC000012E
STATUS_INVALID_IMAGE_NOT_MZ	0xC000012F
STATUS_INVALID_IMAGE_PROTECT	0xC0000130
STATUS_INVALID_IMAGE_WIN_16	0xC0000131
STATUS_LOGON_SERVER_CONFLICT	0xC0000132
STATUS_TIME_DIFFERENCE_AT_DC	0xC0000133
STATUS_SYNCHRONIZATION_REQUIRED	0xC0000134
STATUS_DLL_NOT_FOUND	0xC0000135
STATUS_OPEN_FAILED	0xC0000136
STATUS_IO_PRIVILEGE_FAILED	0xC0000137
STATUS_ORDINAL_NOT_FOUND	0xC0000138
STATUS_ENTRYPOINT_NOT_FOUND	0xC0000139
STATUS_CONTROL_C_EXIT	0xC000013A
STATUS_LOCAL_DISCONNECT	0xC000013B
STATUS_REMOTE_DISCONNECT	0xC000013C
STATUS_REMOTE_RESOURCES	0xC000013D
STATUS_LINK_FAILED	0xC000013E
STATUS_LINK_TIMEOUT	0xC000013F

Status	Valeur
STATUS_INVALID_CONNECTION	0xC0000140
STATUS_INVALID_ADDRESS	0xC0000141
STATUS_DLL_INIT_FAILED	0xC0000142
STATUS_MISSING_SYSTEMFILE	0xC0000143
STATUS_UNHANDLED_EXCEPTION	0xC0000144
STATUS_APP_INIT_FAILURE	0xC0000145
STATUS_PAGEFILE_CREATE_FAILED	0xC0000146
STATUS_NO_PAGEFILE	0xC0000147
STATUS_INVALID_LEVEL	0xC0000148
STATUS_WRONG_PASSWORD_CORE	0xC0000149
STATUS_ILLEGAL_FLOAT_CONTEXT	0xC000014A
STATUS_PIPE_BROKEN	0xC000014B
STATUS_REGISTRY_CORRUPT	0xC000014C
STATUS_REGISTRY_IO_FAILED	0xC000014D
STATUS_NO_EVENT_PAIR	0xC000014E
STATUS_UNRECOGNIZED_VOLUME	0xC000014F
STATUS_SERIAL_NO_DEVICE_INITED	0xC0000150
STATUS_NO_SUCH_ALIAS	0xC0000151
STATUS_MEMBER_NOT_IN_ALIAS	0xC0000152
STATUS_MEMBER_IN_ALIAS	0xC0000153
STATUS_ALIAS_EXISTS	0xC0000154
STATUS_LOGON_NOT_GRANTED	0xC0000155
STATUS_TOO_MANY_SECRETS	0xC0000156
STATUS_SECRET_TOO_LONG	0xC0000157
STATUS_INTERNAL_DB_ERROR	0xC0000158
STATUS_FULLSCREEN_MODE	0xC0000159
STATUS_TOO_MANY_CONTEXT_IDS	0xC000015A
STATUS_LOGON_TYPE_NOT_GRANTED	0xC000015B
STATUS_NOT_REGISTRY_FILE	0xC000015C
STATUS_FT_MISSING_MEMBER	0xC000015F
STATUS_ILL_FORMED_SERVICE_ENTRY	0xC0000160
STATUS_ILLEGAL_CHARACTER	0xC0000161
STATUS_UNMAPPABLE_CHARACTER	0xC0000162
STATUS_UNDEFINED_CHARACTER	0xC0000163
STATUS_FLOPPY_VOLUME	0xC0000164
STATUS_FLOPPY_ID_MARK_NOT_FOUND	0xC0000165
STATUS_FLOPPY_WRONG_CYLINDER	0xC0000166
STATUS_FLOPPY_UNKNOWN_ERROR	0xC0000167
STATUS_FLOPPY_BAD_REGISTERS	0xC0000168
STATUS_DISK_RECALIBRATE_FAILED	0xC0000169
STATUS_DISK_OPERATION_FAILED	0xC000016A
STATUS_DISK_RESET_FAILED	0xC000016B
STATUS_SHARED_IRQ_BUSY	0xC000016C
STATUS_FT_ORPHANING	0xC000016D
STATUS_PARTITION_FAILURE	0xC0000172
STATUS_INVALID_BLOCK_LENGTH	0xC0000173
STATUS_DEVICE_NOT_PARTITIONED	0xC0000174

Status	Valeur
STATUS_UNABLE_TO_LOCK_MEDIA	0xC0000175
STATUS_UNABLE_TO_UNLOAD_MEDIA	0xC0000176
STATUS_EOM_OVERFLOW	0xC0000177
STATUS_NO_MEDIA	0xC0000178
STATUS_NO_SUCH_MEMBER	0xC000017A
STATUS_INVALID_MEMBER	0xC000017B
STATUS_KEY_DELETED	0xC000017C
STATUS_NO_LOG_SPACE	0xC000017D
STATUS_TOO_MANY_SIDS	0xC000017E
STATUS_KEY_HAS_CHILDREN	0xC0000180
STATUS_CHILD_MUST_BE_VOLATILE	0xC0000181
STATUS_DRIVER_INTERNAL_ERROR	0xC0000183
STATUS_INVALID_DEVICE_STATE	0xC0000184
STATUS_IO_DEVICE_ERROR	0xC0000185
STATUS_DEVICE_PROTOCOL_ERROR	0xC0000186
STATUS_BACKUP_CONTROLLER	0xC0000187
STATUS_LOG_FILE_FULL	0xC0000188
STATUS_TOO_LATE	0xC0000189
STATUS_NO_TRUST_LSA_SECRET	0xC000018A
STATUS_NO_TRUST_SAM_ACCOUNT	0xC000018B
STATUS_TRUSTED_DOMAIN_FAILURE	0xC000018C
STATUS_EVENTLOG_FILE_CORRUPT	0xC000018E
STATUS_EVENTLOG_CANT_START	0xC000018F
STATUS_TRUST_FAILURE	0xC0000190
STATUS_MUTANT_LIMIT_EXCEEDED	0xC0000191
STATUS_NETLOGON_NOT_STARTED	0xC0000192
STATUS_ACCOUNT_EXPIRED	0xC0000193
STATUS_POSSIBLE_DEADLOCK	0xC0000194
STATUS_REMOTE_SESSION_LIMIT	0xC0000196
STATUS_EVENTLOG_FILE_CHANGED	0xC0000197
STATUS_FS_DRIVER_REQUIRED	0xC000019C
STATUS_INVALID_LOCK_RANGE	0xC00001A1
STATUS_INVALID_ACE_CONDITION	0xC00001A2
STATUS_DUPLICATE_PRIVILEGES	0xC00001A6
STATUS_REPAIR_NEEDED	0xC00001A8
STATUS_QUOTA_NOT_ENABLED	0xC00001A9
STATUS_NO_APPLICATION_PACKAGE	0xC00001AA
STATUS_NETWORK_OPEN_RESTRICTION	0xC0000201
STATUS_NO_USER_SESSION_KEY	0xC0000202
STATUS_USER_SESSION_DELETED	0xC0000203
STATUS_RESOURCE_LANG_NOT_FOUND	0xC0000204
STATUS_INSUFF_SERVER_RESOURCES	0xC0000205
STATUS_INVALID_BUFFER_SIZE	0xC0000206
STATUS_INVALID_ADDRESS_WILDCARD	0xC0000208
STATUS_TOO_MANY_ADDRESSES	0xC0000209
STATUS_ADDRESS_ALREADY_EXISTS	0xC000020A
STATUS_ADDRESS_CLOSED	0xC000020B

Status	Valeur
STATUS_CONNECTION_DISCONNECTED	0xC000020C
STATUS_CONNECTION_RESET	0xC000020D
STATUS_TOO_MANY_NODES	0xC000020E
STATUS_TRANSACTION_ABORTED	0xC000020F
STATUS_TRANSACTION_TIMED_OUT	0xC0000210
STATUS_TRANSACTION_NO_RELEASE	0xC0000211
STATUS_TRANSACTION_NO_MATCH	0xC0000212
STATUS_TRANSACTION_RESPONDED	0xC0000213
STATUS_TRANSACTION_INVALID_ID	0xC0000214
STATUS_TRANSACTION_INVALID_TYPE	0xC0000215
STATUS_NOT_SERVER_SESSION	0xC0000216
STATUS_NOT_CLIENT_SESSION	0xC0000217
STATUS_DEBUG_ATTACH_FAILED	0xC0000219
STATUS_DATA_NOT_ACCEPTED	0xC000021B
STATUS_NO_BROWSER_SERVERS_FOUND	0xC000021C
STATUS_VDM_HARD_ERROR	0xC000021D
STATUS_DRIVER_CANCEL_TIMEOUT	0xC000021E
STATUS_REPLY_MESSAGE_MISMATCH	0xC000021F
STATUS_MAPPED_ALIGNMENT	0xC0000220
STATUS_IMAGE_CHECKSUM_MISMATCH	0xC0000221
STATUS_LOST_WRITEBEHIND_DATA	0xC0000222
STATUS_PASSWORD_MUST_CHANGE	0xC0000224
STATUS_NOT_FOUND	0xC0000225
STATUS_NOT_TINY_STREAM	0xC0000226
STATUS_RECOVERY_FAILURE	0xC0000227
STATUS_STACK_OVERFLOW_READ	0xC0000228
STATUS_FAIL_CHECK	0xC0000229
STATUS_DUPLICATE_OBJECTID	0xC000022A
STATUS_OBJECTID_EXISTS	0xC000022B
STATUS_CONVERT_TO_LARGE	0xC000022C
STATUS_RETRY	0xC000022D
STATUS_FOUND_OUT_OF_SCOPE	0xC000022E
STATUS_ALLOCATE_BUCKET	0xC000022F
STATUS_PROPSET_NOT_FOUND	0xC0000230
STATUS_MARSHALL_OVERFLOW	0xC0000231
STATUS_INVALID_VARIANT	0xC0000232
STATUS_ACCOUNT_LOCKED_OUT	0xC0000234
STATUS_HANDLE_NOT_CLOSABLE	0xC0000235
STATUS_CONNECTION_REFUSED	0xC0000236
STATUS_GRACEFUL_DISCONNECT	0xC0000237
STATUS_ADDRESS_NOT_ASSOCIATED	0xC0000239
STATUS_CONNECTION_INVALID	0xC000023A
STATUS_CONNECTION_ACTIVE	0xC000023B
STATUS_NETWORK_UNREACHABLE	0xC000023C
STATUS_HOST_UNREACHABLE	0xC000023D
STATUS_PROTOCOL_UNREACHABLE	0xC000023E
STATUS_PORT_UNREACHABLE	0xC000023F

Status	Valeur
STATUS_REQUEST_ABORTED	0xC0000240
STATUS_CONNECTION_ABORTED	0xC0000241
STATUS_BAD_COMPRESSION_BUFFER	0xC0000242
STATUS_USER_MAPPED_FILE	0xC0000243
STATUS_AUDIT_FAILED	0xC0000244
STATUS_TIMER_RESOLUTION_NOT_SET	0xC0000245
STATUS_CONNECTION_COUNT_LIMIT	0xC0000246
STATUS_LOGIN_TIME_RESTRICTION	0xC0000247
STATUS_LOGIN_WKSTA_RESTRICTION	0xC0000248
STATUS_IMAGE_MP_UP_MISMATCH	0xC0000249
STATUS_INSUFFICIENT_LOGON_INFO	0xC0000250
STATUS_BAD_DLL_ENTRYPOINT	0xC0000251
STATUS_BAD_SERVICE_ENTRYPOINT	0xC0000252
STATUS_LPC_REPLY_LOST	0xC0000253
STATUS_IP_ADDRESS_CONFLICT1	0xC0000254
STATUS_IP_ADDRESS_CONFLICT2	0xC0000255
STATUS_REGISTRY_QUOTA_LIMIT	0xC0000256
STATUS_PATH_NOT_COVERED	0xC0000257
STATUS_NO_CALLBACK_ACTIVE	0xC0000258
STATUS_LICENSE_QUOTA_EXCEEDED	0xC0000259
STATUS_PWD_TOO_SHORT	0xC000025A
STATUS_PWD_TOO_RECENT	0xC000025B
STATUS_PWD_HISTORY_CONFLICT	0xC000025C
STATUS_PLUGPLAY_NO_DEVICE	0xC000025E
STATUS_UNSUPPORTED_COMPRESSION	0xC000025F
STATUS_INVALID_HW_PROFILE	0xC0000260
STATUS_DRIVER_ORDINAL_NOT_FOUND	0xC0000262
STATUS_RESOURCE_NOT_OWNED	0xC0000264
STATUS_TOO_MANY_LINKS	0xC0000265
STATUS_QUOTA_LIST_INCONSISTENT	0xC0000266
STATUS_FILE_IS_OFFLINE	0xC0000267
STATUS_EVALUATION_EXPIRATION	0xC0000268
STATUS_ILLEGAL_DLL_RELOCATION	0xC0000269
STATUS_LICENSE_VIOLATION	0xC000026A
STATUS_DLL_INIT_FAILED_LOGOFF	0xC000026B
STATUS_DRIVER_UNABLE_TO_LOAD	0xC000026C
STATUS_DFS_UNAVAILABLE	0xC000026D
STATUS_VOLUME_DISMOUNTED	0xC000026E
STATUS_WX86_INTERNAL_ERROR	0xC000026F
STATUS_WX86_FLOAT_STACK_CHECK	0xC0000270
STATUS_VALIDATE_CONTINUE	0xC0000271
STATUS_NO_MATCH	0xC0000272
STATUS_NO_MORE_MATCHES	0xC0000273
STATUS_NOT_A_REPARSE_POINT	0xC0000275
STATUS_IO_REPARSE_TAG_INVALID	0xC0000276
STATUS_IO_REPARSE_TAG_MISMATCH	0xC0000277
STATUS_IO_REPARSE_DATA_INVALID	0xC0000278

Status	Valeur
STATUS_PWD_TOO_LONG	0xC000027A
STATUS_STOWED_EXCEPTION	0xC000027B
STATUS_RANGE_LIST_CONFLICT	0xC0000282
STATUS_SOURCE_ELEMENT_EMPTY	0xC0000283
STATUS_DESTINATION_ELEMENT_FULL	0xC0000284
STATUS_ILLEGAL_ELEMENT_ADDRESS	0xC0000285
STATUS_MAGAZINE_NOT_PRESENT	0xC0000286
STATUS_REINITIALIZATION_NEEDED	0xC0000287
STATUS_DEVICE_REQUIRES_CLEANING	0x80000288
STATUS_DEVICE_DOOR_OPEN	0x80000289
STATUS_ENCRYPTION_FAILED	0xC000028A
STATUS_DECRYPTION_FAILED	0xC000028B
STATUS_RANGE_NOT_FOUND	0xC000028C
STATUS_NO_RECOVERY_POLICY	0xC000028D
STATUS_NO_EFS	0xC000028E
STATUS_WRONG_EFS	0xC000028F
STATUS_NO_USER_KEYS	0xC0000290
STATUS_FILE_NOT_ENCRYPTED	0xC0000291
STATUS_NOT_EXPORT_FORMAT	0xC0000292
STATUS_FILE_ENCRYPTED	0xC0000293
STATUS_WAKE_SYSTEM	0x40000294
STATUS_WMI_GUID_NOT_FOUND	0xC0000295
STATUS_WMI_INSTANCE_NOT_FOUND	0xC0000296
STATUS_WMI_ITEMID_NOT_FOUND	0xC0000297
STATUS_WMI_TRY_AGAIN	0xC0000298
STATUS_SHARED_POLICY	0xC0000299
STATUS_POLICY_OBJECT_NOT_FOUND	0xC000029A
STATUS_POLICY_ONLY_IN_DS	0xC000029B
STATUS_VOLUME_NOT_UPGRADED	0xC000029C
STATUS_NO_TRACKING_SERVICE	0xC000029F
STATUS_SERVER_SID_MISMATCH	0xC00002A0
STATUS_DS_NO_ATTRIBUTE_OR_VALUE	0xC00002A1
STATUS_DS_BUSY	0xC00002A5
STATUS_DS_UNAVAILABLE	0xC00002A6
STATUS_DS_NO_RIDS_ALLOCATED	0xC00002A7
STATUS_DS_NO_MORE_RIDS	0xC00002A8
STATUS_DS_INCORRECT_ROLE_OWNER	0xC00002A9
STATUS_DS_RIDMGR_INIT_ERROR	0xC00002AA
STATUS_DS_OBJ_CLASS_VIOLATION	0xC00002AB
STATUS_DS_CANT_ON_NON_LEAF	0xC00002AC
STATUS_DS_CANT_ON_RDN	0xC00002AD
STATUS_DS_CANT_MOD_OBJ_CLASS	0xC00002AE
STATUS_DS_CROSS_DOM_MOVE_FAILED	0xC00002AF
STATUS_DS_GC_NOT_AVAILABLE	0xC00002B0
STATUS_CANT_ENABLE_DENY_ONLY	0xC00002B3
STATUS_FLOAT_MULTIPLE_FAULTS	0xC00002B4
STATUS_FLOAT_MULTIPLE_TRAPS	0xC00002B5

Status	Valeur
STATUS_DEVICE_REMOVED	0xC00002B6
STATUS_JOURNAL_NOT_ACTIVE	0xC00002B8
STATUS_NOINTERFACE	0xC00002B9
STATUS_DS_RIDMGR_DISABLED	0xC00002BA
STATUS_DS_ADMIN_LIMIT_EXCEEDED	0xC00002C1
STATUS_DRIVER_FAILED_SLEEP	0xC00002C2
STATUS_WMI_READ_ONLY	0xC00002C6
STATUS_WMI_SET_FAILURE	0xC00002C7
STATUS_COMMITMENT_MINIMUM	0xC00002C8
STATUS_REG_NAT_CONSUMPTION	0xC00002C9
STATUS_TRANSPORT_FULL	0xC00002CA
STATUS_DS_SAM_INIT_FAILURE	0xC00002CB
STATUS_ONLY_IF_CONNECTED	0xC00002CC
STATUS_PNP_RESTART_ENUMERATION	0xC00002CE
STATUS_JOURNAL_ENTRY_DELETED	0xC00002CF
STATUS_PNP_REBOOT_REQUIRED	0xC00002D2
STATUS_POWER_STATE_INVALID	0xC00002D3
STATUS_DS_INVALID_GROUP_TYPE	0xC00002D4
STATUS_DS_HAVE_PRIMARY_MEMBERS	0xC00002DC
STATUS_WMI_NOT_SUPPORTED	0xC00002DD
STATUS_INSUFFICIENT_POWER	0xC00002DE
STATUS_SAM_NEED_BOOTKEY_FLOPPY	0xC00002E0
STATUS_DS_CANT_START	0xC00002E1
STATUS_DS_INIT_FAILURE	0xC00002E2
STATUS_SAM_INIT_FAILURE	0xC00002E3
STATUS_DS_GC_REQUIRED	0xC00002E4
STATUS_MULTIPLE_FAULT_VIOLATION	0xC00002E8
STATUS_CANNOT_MAKE	0xC00002EA
STATUS_SYSTEM_SHUTDOWN	0xC00002EB
STATUS_DS_INIT_FAILURE_CONSOLE	0xC00002EC
STATUS_NO_TGT_REPLY	0xC00002EF
STATUS_OBJECTID_NOT_FOUND	0xC00002F0
STATUS_NO_IP_ADDRESSES	0xC00002F1
STATUS_WRONG_CREDENTIAL_HANDLE	0xC00002F2
STATUS_CRYPTO_SYSTEM_INVALID	0xC00002F3
STATUS_MAX_REFERRALS_EXCEEDED	0xC00002F4
STATUS_MUST_BE_KDC	0xC00002F5
STATUS_TOO_MANY_PRINCIPALS	0xC00002F7
STATUS_NO_PA_DATA	0xC00002F8
STATUS_PKINIT_NAME_MISMATCH	0xC00002F9
STATUS_SMARTCARD_LOGON_REQUIRED	0xC00002FA
STATUS_KDC_INVALID_REQUEST	0xC00002FB
STATUS_KDC_UNABLE_TO_REFER	0xC00002FC
STATUS_KDC_UNKNOWN_ETYPE	0xC00002FD
STATUS_SHUTDOWN_IN_PROGRESS	0xC00002FE
STATUS_NOT_SUPPORTED_ON_SBS	0xC0000300
STATUS_WMI_GUID_DISCONNECTED	0xC0000301

Status	Valeur
STATUS_WMI_ALREADY_DISABLED	0xC0000302
STATUS_WMI_ALREADY_ENABLED	0xC0000303
STATUS_MFT_TOO_FRAGMENTED	0xC0000304
STATUS_COPY_PROTECTION_FAILURE	0xC0000305
STATUS_CSS_KEY_NOT_PRESENT	0xC0000307
STATUS_CSS_KEY_NOT_ESTABLISHED	0xC0000308
STATUS_CSS_SCRAMBLED_SECTOR	0xC0000309
STATUS_CSS_REGION_MISMATCH	0xC000030A
STATUS_CSS_RESETS_EXHAUSTED	0xC000030B
STATUS_PASSWORD_CHANGE_REQUIRED	0xC000030C
STATUS_PKINIT_FAILURE	0xC0000320
STATUS_NO_KERB_KEY	0xC0000322
STATUS_HOST_DOWN	0xC0000350
STATUS_UNSUPPORTED_PREAUTH	0xC0000351
STATUS_EFS_ALG_BLOB_TOO_BIG	0xC0000352
STATUS_PORT_NOT_SET	0xC0000353
STATUS_DEBUGGER_INACTIVE	0xC0000354
STATUS_DS_VERSION_CHECK_FAILURE	0xC0000355
STATUS_AUDITING_DISABLED	0xC0000356
STATUS_PRENT4_MACHINE_ACCOUNT	0xC0000357
STATUS_INVALID_IMAGE_WIN_32	0xC0000359
STATUS_INVALID_IMAGE_WIN_64	0xC000035A
STATUS_BAD_BINDINGS	0xC000035B
STATUS_NETWORK_SESSION_EXPIRED	0xC000035C
STATUS_APPHELP_BLOCK	0xC000035D
STATUS_ALL_SIDS_FILTERED	0xC000035E
STATUS_NOT_SAFE_MODE_DRIVER	0xC000035F
STATUS_FAILED_DRIVER_ENTRY	0xC0000365
STATUS_DEVICE_ENUMERATION_ERROR	0xC0000366
STATUS_MOUNT_POINT_NOT_RESOLVED	0xC0000368
STATUS_MCA_OCCURED	0xC000036A
STATUS_SYSTEM_HIVE_TOO_LARGE	0xC000036E
STATUS_DS_SHUTTING_DOWN	0x40000370
STATUS_NO_SECRETS	0xC0000371
STATUS_FAILED_STACK_SWITCH	0xC0000373
STATUS_HEAP_CORRUPTION	0xC0000374
STATUS_SMARTCARD_WRONG_PIN	0xC0000380
STATUS_SMARTCARD_CARD_BLOCKED	0xC0000381
STATUS_SMARTCARD_NO_CARD	0xC0000383
STATUS_SMARTCARD_NO_CERTIFICATE	0xC0000385
STATUS_SMARTCARD_NO_KEYSET	0xC0000386
STATUS_SMARTCARD_IO_ERROR	0xC0000387
STATUS_DOWNGRADE_DETECTED	0xC0000388
STATUS_SMARTCARD_CERT_REVOKED	0xC0000389
STATUS_ISSUING_CA_UNTRUSTED	0xC000038A
STATUS_REVOCATION_OFFLINE_C	0xC000038B
STATUS_PKINIT_CLIENT_FAILURE	0xC000038C

Status	Valeur
STATUS_SMARTCARD_CERT_EXPIRED	0xC000038D
STATUS_SMARTCARD_SILENT_CONTEXT	0xC000038F
STATUS_DS_NAME_NOT_UNIQUE	0xC0000404
STATUS_DS_DUPLICATE_ID_FOUND	0xC0000405
STATUS_USER2USER_REQUIRED	0xC0000408
STATUS_STACK_BUFFER_OVERRUN	0xC0000409
STATUS_NO_S4U_PROT_SUPPORT	0xC000040A
STATUS_REVOCATION_OFFLINE_KDC	0xC000040C
STATUS_ISSUING_CA_UNTRUSTED_KDC	0xC000040D
STATUS_KDC_CERT_EXPIRED	0xC000040E
STATUS_KDC_CERT_REVOKED	0xC000040F
STATUS_PARAMETER_QUOTA_EXCEEDED	0xC0000410
STATUS_HIBERNATION_FAILURE	0xC0000411
STATUS_DELAY_LOAD_FAILED	0xC0000412
STATUS_VDM_DISALLOWED	0xC0000414
STATUS_NTLM_BLOCKED	0xC0000418
STATUS_ASSERTION_FAILURE	0xC0000420
STATUS_VERIFIER_STOP	0xC0000421
STATUS_CALLBACK_POP_STACK	0xC0000423
STATUS_HIVE_UNLOADED	0xC0000425
STATUS_COMPRESSION_DISABLED	0xC0000426
STATUS_FILE_SYSTEM_LIMITATION	0xC0000427
STATUS_INVALID_IMAGE_HASH	0xC0000428
STATUS_NOT_CAPABLE	0xC0000429
STATUS_REQUEST_OUT_OF_SEQUENCE	0xC000042A
STATUS_IMPLEMENTATION_LIMIT	0xC000042B
STATUS_ELEVATION_REQUIRED	0xC000042C
STATUS_NO_SECURITY_CONTEXT	0xC000042D
STATUS_PKU2U_CERT_FAILURE	0xC000042F
STATUS_BEYOND_VDL	0xC0000432
STATUS_PTE_CHANGED	0xC0000434
STATUS_PURGE_FAILED	0xC0000435
STATUS_INVALID_LABEL	0xC0000446
STATUS_AMBIGUOUS_SYSTEM_DEVICE	0xC0000451
STATUS_SYSTEM_DEVICE_NOT_FOUND	0xC0000452
STATUS_RESTART_BOOT_APPLICATION	0xC0000453
STATUS_INVALID_SESSION	0xC0000455
STATUS_THREAD_NOT_IN_SESSION	0xC0000457
STATUS_INVALID_WEIGHT	0xC0000458
STATUS_REQUEST_PAUSED	0xC0000459
STATUS_NO_RANGES_PROCESSED	0xC0000460
STATUS_DISK_RESOURCES_EXHAUSTED	0xC0000461
STATUS_NEEDS_REMEDIATION	0xC0000462
STATUS_DEVICE_UNREACHABLE	0xC0000464
STATUS_INVALID_TOKEN	0xC0000465
STATUS_SERVER_UNAVAILABLE	0xC0000466
STATUS_FILE_NOT_AVAILABLE	0xC0000467

Status	Valeur
STATUS_PACKAGE_UPDATING	0xC0000469
STATUS_NOT_READ_FROM_COPY	0xC000046A
STATUS_FT_WRITE_FAILURE	0xC000046B
STATUS_FT_DI_SCAN_REQUIRED	0xC000046C
STATUS_DATA_CHECKSUM_ERROR	0xC0000470
STATUS_INVALID_OFFSET_ALIGNMENT	0xC0000474
STATUS_OPERATION_IN_PROGRESS	0xC0000476
STATUS_SCRUB_DATA_DISABLED	0xC0000478
STATUS_NOT_REDUNDANT_STORAGE	0xC0000479
STATUS_DIRECTORY_NOT_SUPPORTED	0xC000047C
STATUS_IO_OPERATION_TIMEOUT	0xC000047D
STATUS_SYSTEM_NEEDS_REMEDIATION	0xC000047E
STATUS_SHARE_UNAVAILABLE	0xC0000480
STATUS_INVALID_TASK_NAME	0xC0000500
STATUS_INVALID_TASK_INDEX	0xC0000501
STATUS_THREAD_ALREADY_IN_TASK	0xC0000502
STATUS_CALLBACK_BYPASS	0xC0000503
STATUS_UNDEFINED_SCOPE	0xC0000504
STATUS_INVALID_CAP	0xC0000505
STATUS_NOT_GUI_PROCESS	0xC0000506
STATUS_FAIL_FAST_EXCEPTION	0xC0000602
STATUS_IMAGE_CERT_REVOKED	0xC0000603
STATUS_PORT_CLOSED	0xC0000700
STATUS_MESSAGE_LOST	0xC0000701
STATUS_INVALID_MESSAGE	0xC0000702
STATUS_REQUEST_CANCELED	0xC0000703
STATUS_RECURSIVE_DISPATCH	0xC0000704
STATUS_LPC_REQUESTS_NOT_ALLOWED	0xC0000707
STATUS_RESOURCE_IN_USE	0xC0000708
STATUS_HARDWARE_MEMORY_ERROR	0xC0000709
STATUS_PROCESS_IS_PROTECTED	0xC0000712
STATUS_MCA_EXCEPTION	0xC0000713
STATUS_SYMLINK_CLASS_DISABLED	0xC0000715
STATUS_NO_UNICODE_TRANSLATION	0xC0000717
STATUS_ALREADY_REGISTERED	0xC0000718
STATUS_CONTEXT_MISMATCH	0xC0000719
STATUS_INVALID_THREAD	0xC000071C
STATUS_CALLBACK_RETURNED_LANG	0xC000071F
STATUS_DISK_REPAIR_DISABLED	0xC0000800
STATUS_DISK_QUOTA_EXCEEDED	0xC0000802
STATUS_DATA_LOST_REPAIR	0x80000803
STATUS_CONTENT_BLOCKED	0xC0000804
STATUS_BAD_CLUSTERS	0xC0000805
STATUS_VOLUME_DIRTY	0xC0000806
STATUS_DISK_REPAIR_REDIRECTED	0x40000807
STATUS_DISK_REPAIR_UNSUCCESSFUL	0xC0000808
STATUS_CORRUPT_LOG_OVERFULL	0xC0000809

Status	Valeur
STATUS_CORRUPT_LOG_CORRUPTED	0xC000080A
STATUS_CORRUPT_LOG_UNAVAILABLE	0xC000080B
STATUS_CORRUPT_LOG_DELETED_FULL	0xC000080C
STATUS_CORRUPT_LOG_CLEARED	0xC000080D
STATUS_ORPHAN_NAME_EXHAUSTED	0xC000080E
STATUS_FILE_CHECKED_OUT	0xC0000901
STATUS_CHECKOUT_REQUIRED	0xC0000902
STATUS_BAD_FILE_TYPE	0xC0000903
STATUS_FILE_TOO_LARGE	0xC0000904
STATUS_FORMS_AUTH_REQUIRED	0xC0000905
STATUS_VIRUS_INFECTED	0xC0000906
STATUS_VIRUS_DELETED	0xC0000907
STATUS_BAD_MCFG_TABLE	0xC0000908
STATUS_CANNOT_BREAK_OPLOCK	0xC0000909
STATUS_BAD_KEY	0xC000090A
STATUS_BAD_DATA	0xC000090B
STATUS_NO_KEY	0xC000090C
STATUS_FILE_HANDLE_REVOKED	0xC0000910
STATUS_WOW_ASSERTION	0xC0009898
STATUS_INVALID_SIGNATURE	0xC000A000
STATUS_HMAC_NOT_SUPPORTED	0xC000A001
STATUS_AUTH_TAG_MISMATCH	0xC000A002
STATUS_INVALID_STATE_TRANSITION	0xC000A003
STATUS_INVALID_PEP_INFO_VERSION	0xC000A005
STATUS_IPSEC_QUEUE_OVERFLOW	0xC000A010
STATUS_ND_QUEUE_OVERFLOW	0xC000A011
STATUS_HOPLIMIT_EXCEEDED	0xC000A012
STATUS_PROTOCOL_NOT_SUPPORTED	0xC000A013
STATUS_FASTPATH_REJECTED	0xC000A014
STATUS_XML_PARSE_ERROR	0xC000A083
STATUS_XMLDSIG_ERROR	0xC000A084
STATUS_WRONG_COMPARTMENT	0xC000A085
STATUS_AUTHIP_FAILURE	0xC000A086
STATUS_DS_OID_NOT_FOUND	0xC000A088
STATUS_INCORRECT_ACCOUNT_TYPE	0xC000A089
STATUS_HASH_NOT_SUPPORTED	0xC000A100
STATUS_HASH_NOT_PRESENT	0xC000A101
STATUS_GPIO_OPERATION_DENIED	0xC000A125
STATUS_CANNOT_SWITCH_RUNLEVEL	0xC000A141
STATUS_INVALID_RUNLEVEL_SETTING	0xC000A142
STATUS_RUNLEVEL_SWITCH_TIMEOUT	0xC000A143
STATUS_NOT_APPCONTAINER	0xC000A200
STATUS_APP_DATA_NOT_FOUND	0xC000A281
STATUS_APP_DATA_EXPIRED	0xC000A282
STATUS_APP_DATA_CORRUPT	0xC000A283
STATUS_APP_DATA_LIMIT_EXCEEDED	0xC000A284
STATUS_APP_DATA_REBOOT_REQUIRED	0xC000A285

Status	Valeur
DBG_NO_STATE_CHANGE	0xC0010001
DBG_APP_NOT_IDLE	0xC0010002
RPC_NT_INVALID_STRING_BINDING	0xC0020001
RPC_NT_WRONG_KIND_OF_BINDING	0xC0020002
RPC_NT_INVALID_BINDING	0xC0020003
RPC_NT_PROTSEQ_NOT_SUPPORTED	0xC0020004
RPC_NT_INVALID_RPC_PROTSEQ	0xC0020005
RPC_NT_INVALID_STRING_UUID	0xC0020006
RPC_NT_INVALID_ENDPOINT_FORMAT	0xC0020007
RPC_NT_INVALID_NET_ADDR	0xC0020008
RPC_NT_NO_ENDPOINT_FOUND	0xC0020009
RPC_NT_INVALID_TIMEOUT	0xC002000A
RPC_NT_OBJECT_NOT_FOUND	0xC002000B
RPC_NT_ALREADY_REGISTERED	0xC002000C
RPC_NT_TYPE_ALREADY_REGISTERED	0xC002000D
RPC_NT_ALREADY_LISTENING	0xC002000E
RPC_NT_NO_PROTSEQS_REGISTERED	0xC002000F
RPC_NT_NOT_LISTENING	0xC0020010
RPC_NT_UNKNOWN_MGR_TYPE	0xC0020011
RPC_NT_UNKNOWN_IF	0xC0020012
RPC_NT_NO_BINDINGS	0xC0020013
RPC_NT_NO_PROTSEQS	0xC0020014
RPC_NT_CANT_CREATE_ENDPOINT	0xC0020015
RPC_NT_OUT_OF_RESOURCES	0xC0020016
RPC_NT_SERVER_UNAVAILABLE	0xC0020017
RPC_NT_SERVER_TOO_BUSY	0xC0020018
RPC_NT_INVALID_NETWORK_OPTIONS	0xC0020019
RPC_NT_NO_CALL_ACTIVE	0xC002001A
RPC_NT_CALL_FAILED	0xC002001B
RPC_NT_CALL_FAILED_DNE	0xC002001C
RPC_NT_PROTOCOL_ERROR	0xC002001D
RPC_NT_UNSUPPORTED_TRANS_SYN	0xC002001F
RPC_NT_UNSUPPORTED_TYPE	0xC0020021
RPC_NT_INVALID_TAG	0xC0020022
RPC_NT_INVALID_BOUND	0xC0020023
RPC_NT_NO_ENTRY_NAME	0xC0020024
RPC_NT_INVALID_NAME_SYNTAX	0xC0020025
RPC_NT_UNSUPPORTED_NAME_SYNTAX	0xC0020026
RPC_NT_UUID_NO_ADDRESS	0xC0020028
RPC_NT_DUPLICATE_ENDPOINT	0xC0020029
RPC_NT_UNKNOWN_AUTHN_TYPE	0xC002002A
RPC_NT_MAX_CALLS_TOO_SMALL	0xC002002B
RPC_NT_STRING_TOO_LONG	0xC002002C
RPC_NT_PROTSEQ_NOT_FOUND	0xC002002D
RPC_NT_PROCNUM_OUT_OF_RANGE	0xC002002E
RPC_NT_BINDING_HAS_NO_AUTH	0xC002002F
RPC_NT_UNKNOWN_AUTHN_SERVICE	0xC0020030

Status	Valeur
RPC_NT_UNKNOWN_AUTHN_LEVEL	0xC0020031
RPC_NT_INVALID_AUTH_IDENTITY	0xC0020032
RPC_NT_UNKNOWN_AUTHZ_SERVICE	0xC0020033
EPT_NT_INVALID_ENTRY	0xC0020034
EPT_NT_CANT_PERFORM_OP	0xC0020035
EPT_NT_NOT_REGISTERED	0xC0020036
RPC_NT_NOTHING_TO_EXPORT	0xC0020037
RPC_NT_INCOMPLETE_NAME	0xC0020038
RPC_NT_INVALID_VERS_OPTION	0xC0020039
RPC_NT_NO_MORE_MEMBERS	0xC002003A
RPC_NT_NOT_ALL_OBJS_UNEXPORTED	0xC002003B
RPC_NT_INTERFACE_NOT_FOUND	0xC002003C
RPC_NT_ENTRY_ALREADY_EXISTS	0xC002003D
RPC_NT_ENTRY_NOT_FOUND	0xC002003E
RPC_NT_NAME_SERVICE_UNAVAILABLE	0xC002003F
RPC_NT_INVALID_NAF_ID	0xC0020040
RPC_NT_CANNOT_SUPPORT	0xC0020041
RPC_NT_NO_CONTEXT_AVAILABLE	0xC0020042
RPC_NT_INTERNAL_ERROR	0xC0020043
RPC_NT_ZERO_DIVIDE	0xC0020044
RPC_NT_ADDRESS_ERROR	0xC0020045
RPC_NT_FP_DIV_ZERO	0xC0020046
RPC_NT_FP_UNDERFLOW	0xC0020047
RPC_NT_FP_OVERFLOW	0xC0020048
RPC_NT_NO_MORE_ENTRIES	0xC0030001
RPC_NT_SS_CHAR_TRANS_OPEN_FAIL	0xC0030002
RPC_NT_SS_CHAR_TRANS_SHORT_FILE	0xC0030003
RPC_NT_SS_IN_NULL_CONTEXT	0xC0030004
RPC_NT_SS_CONTEXT_MISMATCH	0xC0030005
RPC_NT_SS_CONTEXT_DAMAGED	0xC0030006
RPC_NT_SS_HANDLES_MISMATCH	0xC0030007
RPC_NT_NULL_REF_POINTER	0xC0030009
RPC_NT_ENUM_VALUE_OUT_OF_RANGE	0xC003000A
RPC_NT_BYTE_COUNT_TOO_SMALL	0xC003000B
RPC_NT_BAD_STUB_DATA	0xC003000C
RPC_NT_CALL_IN_PROGRESS	0xC0020049
RPC_NT_NO_MORE_BINDINGS	0xC002004A
RPC_NT_GROUP_MEMBER_NOT_FOUND	0xC002004B
EPT_NT_CANT_CREATE	0xC002004C
RPC_NT_INVALID_OBJECT	0xC002004D
RPC_NT_NO_INTERFACES	0xC002004F
RPC_NT_CALL_CANCELLED	0xC0020050
RPC_NT_BINDING_INCOMPLETE	0xC0020051
RPC_NT_COMM_FAILURE	0xC0020052
RPC_NT_UNSUPPORTED_AUTHN_LEVEL	0xC0020053
RPC_NT_NO_PRINC_NAME	0xC0020054
RPC_NT_NOT_RPC_ERROR	0xC0020055

Status	Valeur
RPC_NT_UUID_LOCAL_ONLY	0x40020056
RPC_NT_SEC_PKG_ERROR	0xC0020057
RPC_NT_NOT_CANCELLED	0xC0020058
RPC_NT_INVALID_ES_ACTION	0xC0030059
RPC_NT_WRONG_ES_VERSION	0xC003005A
RPC_NT_WRONG_STUB_VERSION	0xC003005B
RPC_NT_INVALID_PIPE_OBJECT	0xC003005C
RPC_NT_INVALID_PIPE_OPERATION	0xC003005D
RPC_NT_WRONG_PIPE_VERSION	0xC003005E
RPC_NT_PIPE_CLOSED	0xC003005F
RPC_NT_PIPE_DISCIPLINE_ERROR	0xC0030060
RPC_NT_PIPE_EMPTY	0xC0030061
RPC_NT_INVALID_ASYNC_HANDLE	0xC0020062
RPC_NT_INVALID_ASYNC_CALL	0xC0020063
RPC_NT_PROXY_ACCESS_DENIED	0xC0020064
RPC_NT_COOKIE_AUTH_FAILED	0xC0020065
RPC_NT_SEND_INCOMPLETE	0x400200AF
STATUS_ACPI_INVALID_OPCODE	0xC0140001
STATUS_ACPI_STACK_OVERFLOW	0xC0140002
STATUS_ACPI_ASSERT_FAILED	0xC0140003
STATUS_ACPI_INVALID_INDEX	0xC0140004
STATUS_ACPI_INVALID_ARGUMENT	0xC0140005
STATUS_ACPI_FATAL	0xC0140006
STATUS_ACPI_INVALID_SUPERNAME	0xC0140007
STATUS_ACPI_INVALID_ARGTYPE	0xC0140008
STATUS_ACPI_INVALID_OBJTYPE	0xC0140009
STATUS_ACPI_INVALID_TARGETTYPE	0xC014000A
STATUS_ACPI_ADDRESS_NOT_MAPPED	0xC014000C
STATUS_ACPI_INVALID_EVENTTYPE	0xC014000D
STATUS_ACPI_HANDLER_COLLISION	0xC014000E
STATUS_ACPI_INVALID_DATA	0xC014000F
STATUS_ACPI_INVALID_REGION	0xC0140010
STATUS_ACPI_INVALID_ACCESS_SIZE	0xC0140011
STATUS_ACPI_ACQUIRE_GLOBAL_LOCK	0xC0140012
STATUS_ACPI_ALREADY_INITIALIZED	0xC0140013
STATUS_ACPI_NOT_INITIALIZED	0xC0140014
STATUS_ACPI_INVALID_MUTEX_LEVEL	0xC0140015
STATUS_ACPI_MUTEX_NOT_OWNED	0xC0140016
STATUS_ACPI_MUTEX_NOT_OWNER	0xC0140017
STATUS_ACPI_RS_ACCESS	0xC0140018
STATUS_ACPI_INVALID_TABLE	0xC0140019
STATUS_ACPI_REG_HANDLER_FAILED	0xC0140020
STATUS_CTX_CDM_CONNECT	0x400A0004
STATUS_CTX_CDM_DISCONNECT	0x400A0005
STATUS_CTX_CLOSE_PENDING	0xC00A0006
STATUS_CTX_NO_OUTBUF	0xC00A0007
STATUS_CTX_MODEM_INF_NOT_FOUND	0xC00A0008

Status	Valeur
STATUS_CTX_RESPONSE_ERROR	0xC00A000A
STATUS_CTX_MODEM_RESPONSE_BUSY	0xC00A000E
STATUS_CTX_MODEM_RESPONSE_VOICE	0xC00A000F
STATUS_CTX_TD_ERROR	0xC00A0010
STATUS_CTX_LICENSE_EXPIRED	0xC00A0014
STATUS_CTX_WINSTATION_NOT_FOUND	0xC00A0015
STATUS_CTX_WINSTATION_BUSY	0xC00A0017
STATUS_CTX_BAD_VIDEO_MODE	0xC00A0018
STATUS_CTX_GRAPHICS_INVALID	0xC00A0022
STATUS_CTX_NOT_CONSOLE	0xC00A0024
STATUS_CTX_CLIENT_QUERY_TIMEOUT	0xC00A0026
STATUS_CTX_CONSOLE_DISCONNECT	0xC00A0027
STATUS_CTX_CONSOLE_CONNECT	0xC00A0028
STATUS_CTX_SHADOW_DENIED	0xC00A002A
STATUS_CTX_SHADOW_INVALID	0xC00A0030
STATUS_CTX_SHADOW_DISABLED	0xC00A0031
STATUS_CTX_SHADOW_NOT_RUNNING	0xC00A0036
STATUS_CTX_LOGON_DISABLED	0xC00A0037
STATUS_TS_INCOMPATIBLE_SESSIONS	0xC00A0039
STATUS_TS_VIDEO_SUBSYSTEM_ERROR	0xC00A003A
STATUS_PNP_BAD_MPS_TABLE	0xC0040035
STATUS_PNP_TRANSLATION_FAILED	0xC0040036
STATUS_IO_REISSUE_AS_CACHED	0xC0040039
STATUS_MUI_FILE_NOT_FOUND	0xC00B0001
STATUS_MUI_INVALID_FILE	0xC00B0002
STATUS_MUI_INVALID_RC_CONFIG	0xC00B0003
STATUS_MUI_INVALID_LOCALE_NAME	0xC00B0004
STATUS_MUI_FILE_NOT_LOADED	0xC00B0006
STATUS_RESOURCE_ENUM_USER_STOP	0xC00B0007
STATUS_FLT_NO_HANDLER_DEFINED	0xC01C0001
STATUS_FLT_DISALLOW_FAST_IO	0xC01C0004
STATUS_FLT_INVALID_NAME_REQUEST	0xC01C0005
STATUS_FLT_NOT_INITIALIZED	0xC01C0007
STATUS_FLT_FILTER_NOT_READY	0xC01C0008
STATUS_FLT_INTERNAL_ERROR	0xC01C000A
STATUS_FLT_DELETING_OBJECT	0xC01C000B
STATUS_FLT_DUPLICATE_ENTRY	0xC01C000D
STATUS_FLT_CBDQ_DISABLED	0xC01C000E
STATUS_FLT_DO_NOT_ATTACH	0xC01C000F
STATUS_FLT_DO_NOT_DETACH	0xC01C0010
STATUS_FLT_FILTER_NOT_FOUND	0xC01C0013
STATUS_FLT_VOLUME_NOT_FOUND	0xC01C0014
STATUS_FLT_INSTANCE_NOT_FOUND	0xC01C0015
STATUS_FLT_NAME_CACHE_MISS	0xC01C0018
STATUS_FLT_NO_DEVICE_OBJECT	0xC01C0019
STATUS_FLT_ALREADY_ENLISTED	0xC01C001B
STATUS_FLT_NO_WAITER_FOR_REPLY	0xC01C0020

Status	Valeur
STATUS_FLT_REGISTRATION_BUSY	0xC01C0023
STATUS_SXS_SECTION_NOT_FOUND	0xC0150001
STATUS_SXS_CANT_GEN_ACTCTX	0xC0150002
STATUS_SXS_ASSEMBLY_NOT_FOUND	0xC0150004
STATUS_SXS_MANIFEST_PARSE_ERROR	0xC0150006
STATUS_SXS_KEY_NOT_FOUND	0xC0150008
STATUS_SXS_VERSION_CONFLICT	0xC0150009
STATUS_SXS_WRONG_SECTION_TYPE	0xC015000A
STATUS_SXS_ASSEMBLY_MISSING	0xC015000C
STATUS_SXS_EARLY_DEACTIVATION	0xC015000F
STATUS_SXS_INVALID_DEACTIVATION	0xC0150010
STATUS_SXS_CORRUPTION	0xC0150015
STATUS_SXS_IDENTITY_PARSE_ERROR	0xC0150019
STATUS_SXS_FILE_HASH_MISMATCH	0xC015001B
STATUS_SXS_IDENTITIES_DIFFERENT	0xC015001D
STATUS_XML_ENCODING_MISMATCH	0xC0150021
STATUS_SXS_MANIFEST_TOO_BIG	0xC0150022
STATUS_GENERIC_COMMAND_FAILED	0xC0150026
STATUS_SXS_FILE_HASH_MISSING	0xC0150027
STATUS_CLUSTER_INVALID_NODE	0xC0130001
STATUS_CLUSTER_NODE_EXISTS	0xC0130002
STATUS_CLUSTER_JOIN_IN_PROGRESS	0xC0130003
STATUS_CLUSTER_NODE_NOT_FOUND	0xC0130004
STATUS_CLUSTER_NETWORK_EXISTS	0xC0130006
STATUS_CLUSTER_INVALID_REQUEST	0xC013000A
STATUS_CLUSTER_NODE_DOWN	0xC013000C
STATUS_CLUSTER_NODE_UNREACHABLE	0xC013000D
STATUS_CLUSTER_NODE_NOT_MEMBER	0xC013000E
STATUS_CLUSTER_INVALID_NETWORK	0xC0130010
STATUS_CLUSTER_NO_NET_ADAPTERS	0xC0130011
STATUS_CLUSTER_NODE_UP	0xC0130012
STATUS_CLUSTER_NODE_PAUSED	0xC0130013
STATUS_CLUSTER_NODE_NOT_PAUSED	0xC0130014
STATUS_CLUSTER_POISONED	0xC0130017
STATUS_CLUSTER_NON_CSV_PATH	0xC0130018
STATUS_CLUSTER_CSV_REDIRECTED	0xC0130022
STATUS_TRANSACTIONAL_CONFLICT	0xC0190001
STATUS_INVALID_TRANSACTION	0xC0190002
STATUS_TRANSACTION_NOT_ACTIVE	0xC0190003
STATUS_TM_INITIALIZATION_FAILED	0xC0190004
STATUS_RM_NOT_ACTIVE	0xC0190005
STATUS_RM_METADATA_CORRUPT	0xC0190006
STATUS_TRANSACTION_NOT_JOINED	0xC0190007
STATUS_DIRECTORY_NOT_RM	0xC0190008
STATUS_COULD_NOT_RESIZE_LOG	0x80190009
STATUS_LOG_RESIZE_INVALID_SIZE	0xC019000B
STATUS_CRM_PROTOCOL_NOT_FOUND	0xC0190011

Status	Valeur
STATUS_LOG_GROWTH_FAILED	0xC0190019
STATUS_OBJECT_NO_LONGER_EXISTS	0xC0190021
STATUS_HANDLE_NO_LONGER_VALID	0xC0190028
STATUS_NO_TXF_METADATA	0x80190029
STATUS_LOG_CORRUPTION_DETECTED	0xC0190030
STATUS_RM_DISCONNECTED	0xC0190032
STATUS_ENLISTMENT_NOT_SUPERIOR	0xC0190033
STATUS_RECOVERY_NOT_NEEDED	0x40190034
STATUS_RM_ALREADY_STARTED	0x40190035
STATUS_CANT_CROSS_RM_BOUNDARY	0xC0190038
STATUS_TXF_DIR_NOT_EMPTY	0xC0190039
STATUS_TM_VOLATILE	0xC019003B
STATUS_ROLLBACK_TIMER_EXPIRED	0xC019003C
STATUS_TXF_ATTRIBUTE_CORRUPT	0xC019003D
STATUS_TRANSACTIONS_NOT_FROZEN	0xC0190045
STATUS_NOT_SNAPSHOT_VOLUME	0xC0190047
STATUS_TM_IDENTITY_MISMATCH	0xC019004A
STATUS_FLOATED_SECTION	0xC019004B
STATUS_TRANSACTION_NOT_FOUND	0xC019004E
STATUS_ENLISTMENT_NOT_FOUND	0xC0190050
STATUS_TRANSACTION_NOT_ROOT	0xC0190054
STATUS_TRANSACTION_NO_SUPERIOR	0xC019005F
STATUS_EXPIRED_HANDLE	0xC0190060
STATUS_TRANSACTION_NOT_ENLISTED	0xC0190061
STATUS_LOG_SECTOR_INVALID	0xC01A0001
STATUS_LOG_SECTOR_REMAPPED	0xC01A0003
STATUS_LOG_BLOCK_INCOMPLETE	0xC01A0004
STATUS_LOG_INVALID_RANGE	0xC01A0005
STATUS_LOG_BLOCKS_EXHAUSTED	0xC01A0006
STATUS_LOG_READ_CONTEXT_INVALID	0xC01A0007
STATUS_LOG_RESTART_INVALID	0xC01A0008
STATUS_LOG_BLOCK_VERSION	0xC01A0009
STATUS_LOG_BLOCK_INVALID	0xC01A000A
STATUS_LOG_READ_MODE_INVALID	0xC01A000B
STATUS_LOG_NO_RESTART	0x401A000C
STATUS_LOG_METADATA_CORRUPT	0xC01A000D
STATUS_LOG_METADATA_INVALID	0xC01A000E
STATUS_LOG_RESERVATION_INVALID	0xC01A0010
STATUS_LOG_CANT_DELETE	0xC01A0011
STATUS_LOG_START_OF_LOG	0xC01A0013
STATUS_LOG_POLICY_NOT_INSTALLED	0xC01A0015
STATUS_LOG_POLICY_INVALID	0xC01A0016
STATUS_LOG_POLICY_CONFLICT	0xC01A0017
STATUS_LOG_PINNED_ARCHIVE_TAIL	0xC01A0018
STATUS_LOG_RECORD_NONEXISTENT	0xC01A0019
STATUS_LOG_TAIL_INVALID	0xC01A001C
STATUS_LOG_FULL	0xC01A001D

Status	Valeur
STATUS_LOG_MULTIPLEXED	0xC01A001E
STATUS_LOG_DEDICATED	0xC01A001F
STATUS_LOG_ARCHIVE_IN_PROGRESS	0xC01A0021
STATUS_LOG_EPHEMERAL	0xC01A0022
STATUS_LOG_STATE_INVALID	0xC01A002B
STATUS_LOG_PINNED	0xC01A002C
STATUS_LOG_PINNED_RESERVATION	0xC01A0030
STATUS_MONITOR_NO_DESCRIPTOR	0xC01D0001
STATUS_GRAPHICS_PRESENT_DENIED	0xC01E0007
STATUS_GRAPHICS_DRIVER_MISMATCH	0xC01E0009
STATUS_GRAPHICS_NO_VIDEO_MEMORY	0xC01E0100
STATUS_GRAPHICS_ALLOCATION_BUSY	0xC01E0102
STATUS_GRAPHICS_TRY_AGAIN_LATER	0xC01E0104
STATUS_GRAPHICS_TRY_AGAIN_NOW	0xC01E0105
STATUS_GRAPHICS_INVALID_VIDPN	0xC01E0303
STATUS_GRAPHICS_MODE_NOT_PINNED	0x401E0307
STATUS_GRAPHICS_STALE_MODESET	0xC01E0320
STATUS_GRAPHICS_NO_VIDPNMGR	0xC01E0335
STATUS_GRAPHICS_NO_ACTIVE_VIDPN	0xC01E0336
STATUS_GRAPHICS_INVALID_STRIDE	0xC01E033C
STATUS_GRAPHICS_START_DEFERRED	0x401E043A
STATUS_GRAPHICS_PVP_HFS_FAILED	0xC01E0511
STATUS_GRAPHICS_OPM_INVALID_SRM	0xC01E0512
STATUS_GRAPHICS_INVALID_POINTER	0xC01E05E4
STATUS_GRAPHICS_INTERNAL_ERROR	0xC01E05E7
STATUS_FVE_LOCKED_VOLUME	0xC0210000
STATUS_FVE_NOT_ENCRYPTED	0xC0210001
STATUS_FVE_BAD_INFORMATION	0xC0210002
STATUS_FVE_TOO_SMALL	0xC0210003
STATUS_FVE_FAILED_WRONG_FS	0xC0210004
STATUS_FVE_BAD_PARTITION_SIZE	0xC0210005
STATUS_FVE_FS_NOT_EXTENDED	0xC0210006
STATUS_FVE_FS_MOUNTED	0xC0210007
STATUS_FVE_NO_LICENSE	0xC0210008
STATUS_FVE_ACTION_NOT_ALLOWED	0xC0210009
STATUS_FVE_BAD_DATA	0xC021000A
STATUS_FVE_VOLUME_NOT_BOUND	0xC021000B
STATUS_FVE_NOT_DATA_VOLUME	0xC021000C
STATUS_FVE_CONV_READ_ERROR	0xC021000D
STATUS_FVE_CONV_WRITE_ERROR	0xC021000E
STATUS_FVE_OVERLAPPED_UPDATE	0xC021000F
STATUS_FVE_FAILED_SECTOR_SIZE	0xC0210010
STATUS_FVE_NOT_OS_VOLUME	0xC0210012
STATUS_FVE_KEYFILE_NOT_FOUND	0xC0210013
STATUS_FVE_KEYFILE_INVALID	0xC0210014
STATUS_FVE_KEYFILE_NO_VMK	0xC0210015
STATUS_FVE_TPM_DISABLED	0xC0210016

Status	Valeur
STATUS_FVE_TPM_INVALID_PCR	0xC0210018
STATUS_FVE_TPM_NO_VMK	0xC0210019
STATUS_FVE_PIN_INVALID	0xC021001A
STATUS_FVE_AUTH_INVALID_CONFIG	0xC021001C
STATUS_FVE_DEBUGGER_ENABLED	0xC021001D
STATUS_FVE_DRY_RUN_FAILED	0xC021001E
STATUS_FVE_BAD_METADATA_POINTER	0xC021001F
STATUS_FVE_OLD_METADATA_COPY	0xC0210020
STATUS_FVE_REBOOT_REQUIRED	0xC0210021
STATUS_FVE_RAW_ACCESS	0xC0210022
STATUS_FVE_RAW_BLOCKED	0xC0210023
STATUS_FVE_MOR_FAILED	0xC0210025
STATUS_FVE_NO_FEATURE_LICENSE	0xC0210026
STATUS_FVE_CONV_RECOVERY_FAILED	0xC0210028
STATUS_FVE_INVALID_DATUM_TYPE	0xC021002A
STATUS_FVE_VOLUME_TOO_SMALL	0xC0210030
STATUS_FVE_ENH_PIN_INVALID	0xC0210031
STATUS_FVE_SECUREBOOT_DISABLED	0xC0210039
STATUS_FVE_DEVICE_LOCKEDOUT	0xC021003B
STATUS_FWP_CALLOUT_NOT_FOUND	0xC0220001
STATUS_FWP_CONDITION_NOT_FOUND	0xC0220002
STATUS_FWP_FILTER_NOT_FOUND	0xC0220003
STATUS_FWP_LAYER_NOT_FOUND	0xC0220004
STATUS_FWP_PROVIDER_NOT_FOUND	0xC0220005
STATUS_FWP_SUBLAYER_NOT_FOUND	0xC0220007
STATUS_FWP_NOT_FOUND	0xC0220008
STATUS_FWP_ALREADY_EXISTS	0xC0220009
STATUS_FWP_IN_USE	0xC022000A
STATUS_FWP_WRONG_SESSION	0xC022000C
STATUS_FWP_NO_TXN_IN_PROGRESS	0xC022000D
STATUS_FWP_TXN_IN_PROGRESS	0xC022000E
STATUS_FWP_TXN_ABORTED	0xC022000F
STATUS_FWP_SESSION_ABORTED	0xC0220010
STATUS_FWP_INCOMPATIBLE_TXN	0xC0220011
STATUS_FWP_TIMEOUT	0xC0220012
STATUS_FWP_NET_EVENTS_DISABLED	0xC0220013
STATUS_FWP_INCOMPATIBLE_LAYER	0xC0220014
STATUS_FWP_KM_CLIENTS_ONLY	0xC0220015
STATUS_FWP_LIFETIME_MISMATCH	0xC0220016
STATUS_FWP_BUILTIN_OBJECT	0xC0220017
STATUS_FWP_TOO_MANY_CALLOUTS	0xC0220018
STATUS_FWP_NOTIFICATION_DROPPED	0xC0220019
STATUS_FWP_TRAFFIC_MISMATCH	0xC022001A
STATUS_FWP_NULL_POINTER	0xC022001C
STATUS_FWP_INVALID_ENUMERATOR	0xC022001D
STATUS_FWP_INVALID_FLAGS	0xC022001E
STATUS_FWP_INVALID_NET_MASK	0xC022001F

Status	Valeur
STATUS_FWP_INVALID_RANGE	0xC0220020
STATUS_FWP_INVALID_INTERVAL	0xC0220021
STATUS_FWP_ZERO_LENGTH_ARRAY	0xC0220022
STATUS_FWP_NULL_DISPLAY_NAME	0xC0220023
STATUS_FWP_INVALID_ACTION_TYPE	0xC0220024
STATUS_FWP_INVALID_WEIGHT	0xC0220025
STATUS_FWP_MATCH_TYPE_MISMATCH	0xC0220026
STATUS_FWP_TYPE_MISMATCH	0xC0220027
STATUS_FWP_OUT_OF_BOUNDS	0xC0220028
STATUS_FWP_RESERVED	0xC0220029
STATUS_FWP_DUPLICATE_CONDITION	0xC022002A
STATUS_FWP_DUPLICATE_KEYMOD	0xC022002B
STATUS_FWP_EM_NOT_SUPPORTED	0xC0220032
STATUS_FWP_NEVER_MATCH	0xC0220033
STATUS_FWP_INVALID_PARAMETER	0xC0220035
STATUS_FWP_TOO_MANY_SUBLAYERS	0xC0220036
STATUS_FWP_L2_DRIVER_NOT_READY	0xC022003E
STATUS_FWP_CONNECTIONS_DISABLED	0xC0220041
STATUS_FWP_INVALID_DNS_NAME	0xC0220042
STATUS_FWP_STILL_ON	0xC0220043
STATUS_FWP_IKEEXT_NOT_RUNNING	0xC0220044
STATUS_FWP_TCPIP_NOT_READY	0xC0220100
STATUS_FWP_INJECT_HANDLE_STALE	0xC0220102
STATUS_FWP_CANNOT_PEND	0xC0220103
STATUS_FWP_DROP_NOICMP	0xC0220104
STATUS_NDIS_CLOSING	0xC0230002
STATUS_NDIS_BAD_VERSION	0xC0230004
STATUS_NDIS_BAD_CHARACTERISTICS	0xC0230005
STATUS_NDIS_ADAPTER_NOT_FOUND	0xC0230006
STATUS_NDIS_OPEN_FAILED	0xC0230007
STATUS_NDIS_DEVICE_FAILED	0xC0230008
STATUS_NDIS_MULTICAST_FULL	0xC0230009
STATUS_NDIS_MULTICAST_EXISTS	0xC023000A
STATUS_NDIS_MULTICAST_NOT_FOUND	0xC023000B
STATUS_NDIS_REQUEST_ABORTED	0xC023000C
STATUS_NDIS_RESET_IN_PROGRESS	0xC023000D
STATUS_NDIS_NOT_SUPPORTED	0xC02300BB
STATUS_NDIS_INVALID_PACKET	0xC023000F
STATUS_NDIS_ADAPTER_NOT_READY	0xC0230011
STATUS_NDIS_INVALID_LENGTH	0xC0230014
STATUS_NDIS_INVALID_DATA	0xC0230015
STATUS_NDIS_BUFFER_TOO_SHORT	0xC0230016
STATUS_NDIS_INVALID_OID	0xC0230017
STATUS_NDIS_ADAPTER_REMOVED	0xC0230018
STATUS_NDIS_UNSUPPORTED_MEDIA	0xC0230019
STATUS_NDIS_FILE_NOT_FOUND	0xC023001B
STATUS_NDIS_ERROR_READING_FILE	0xC023001C

Status	Valeur
STATUS_NDIS_ALREADY_MAPPED	0xC023001D
STATUS_NDIS_RESOURCE_CONFLICT	0xC023001E
STATUS_NDIS_MEDIA_DISCONNECTED	0xC023001F
STATUS_NDIS_INVALID_ADDRESS	0xC0230022
STATUS_NDIS_PAUSED	0xC023002A
STATUS_NDIS_INTERFACE_NOT_FOUND	0xC023002B
STATUS_NDIS_INVALID_PORT	0xC023002D
STATUS_NDIS_INVALID_PORT_STATE	0xC023002E
STATUS_NDIS_LOW_POWER_STATE	0xC023002F
STATUS_NDIS_REINIT_REQUIRED	0xC0230030
STATUS_NDIS_DOT11_MEDIA_IN_USE	0xC0232001
STATUS_NDIS_INDICATION_REQUIRED	0x40230001
STATUS_NDIS_OFFLOAD_POLICY	0xC023100F
STATUS_TPM_ERROR_MASK	0xC0290000
STATUS_TPM_AUTHFAIL	0xC0290001
STATUS_TPM_BADINDEX	0xC0290002
STATUS_TPM_BAD_PARAMETER	0xC0290003
STATUS_TPM_AUDITFAILURE	0xC0290004
STATUS_TPM_CLEAR_DISABLED	0xC0290005
STATUS_TPM_DEACTIVATED	0xC0290006
STATUS_TPM_DISABLED	0xC0290007
STATUS_TPM_DISABLED_CMD	0xC0290008
STATUS_TPM_FAIL	0xC0290009
STATUS_TPM_BAD_ORDINAL	0xC029000A
STATUS_TPM_INSTALL_DISABLED	0xC029000B
STATUS_TPM_INVALID_KEYHANDLE	0xC029000C
STATUS_TPM_KEYNOTFOUND	0xC029000D
STATUS_TPM_INAPPROPRIATE_ENC	0xC029000E
STATUS_TPM_MIGRATEFAIL	0xC029000F
STATUS_TPM_INVALID_PCR_INFO	0xC0290010
STATUS_TPM_NOSPACE	0xC0290011
STATUS_TPM_NOSRK	0xC0290012
STATUS_TPM_NOTSEALED_BLOB	0xC0290013
STATUS_TPM_OWNER_SET	0xC0290014
STATUS_TPM_RESOURCES	0xC0290015
STATUS_TPM_SHORTRANDOM	0xC0290016
STATUS_TPM_SIZE	0xC0290017
STATUS_TPM_WRONGPCRVAL	0xC0290018
STATUS_TPM_BAD_PARAM_SIZE	0xC0290019
STATUS_TPM_SHA_THREAD	0xC029001A
STATUS_TPM_SHA_ERROR	0xC029001B
STATUS_TPM_FAILEDSELFTEST	0xC029001C
STATUS_TPM_AUTH2FAIL	0xC029001D
STATUS_TPM_BADTAG	0xC029001E
STATUS_TPM_IOERROR	0xC029001F
STATUS_TPM_ENCRYPT_ERROR	0xC0290020
STATUS_TPM_DECRYPT_ERROR	0xC0290021

Status	Valeur
STATUS_TPM_INVALID_AUTHHANDLE	0xC0290022
STATUS_TPM_NO_ENDORSEMENT	0xC0290023
STATUS_TPM_INVALID_KEYUSAGE	0xC0290024
STATUS_TPM_WRONG_ENTITYTYPE	0xC0290025
STATUS_TPM_INVALID_POSTINIT	0xC0290026
STATUS_TPM_INAPPROPRIATE_SIG	0xC0290027
STATUS_TPM_BAD_KEY_PROPERTY	0xC0290028
STATUS_TPM_BAD_MIGRATION	0xC0290029
STATUS_TPM_BAD_SCHEME	0xC029002A
STATUS_TPM_BAD_DATASIZE	0xC029002B
STATUS_TPM_BAD_MODE	0xC029002C
STATUS_TPM_BAD_PRESENCE	0xC029002D
STATUS_TPM_BAD_VERSION	0xC029002E
STATUS_TPM_NO_WRAP_TRANSPORT	0xC029002F
STATUS_TPM_AUDITFAIL_SUCCESSFUL	0xC0290031
STATUS_TPM_NOTRESETABLE	0xC0290032
STATUS_TPM_NOTLOCAL	0xC0290033
STATUS_TPM_BAD_TYPE	0xC0290034
STATUS_TPM_INVALID_RESOURCE	0xC0290035
STATUS_TPM_NOTFIPS	0xC0290036
STATUS_TPM_INVALID_FAMILY	0xC0290037
STATUS_TPM_NO_NV_PERMISSION	0xC0290038
STATUS_TPM_REQUIRES_SIGN	0xC0290039
STATUS_TPM_KEY_NOTSUPPORTED	0xC029003A
STATUS_TPM_AUTH_CONFLICT	0xC029003B
STATUS_TPM_AREA_LOCKED	0xC029003C
STATUS_TPM_BAD_LOCALITY	0xC029003D
STATUS_TPM_READ_ONLY	0xC029003E
STATUS_TPM_PER_NOWRITE	0xC029003F
STATUS_TPM_FAMILYCOUNT	0xC0290040
STATUS_TPM_WRITE_LOCKED	0xC0290041
STATUS_TPM_BAD_ATTRIBUTES	0xC0290042
STATUS_TPM_INVALID_STRUCTURE	0xC0290043
STATUS_TPM_KEY_OWNER_CONTROL	0xC0290044
STATUS_TPM_BAD_COUNTER	0xC0290045
STATUS_TPM_NOT_FULLWRITE	0xC0290046
STATUS_TPM_CONTEXT_GAP	0xC0290047
STATUS_TPM_MAXNVWRITES	0xC0290048
STATUS_TPM_NOOPERATOR	0xC0290049
STATUS_TPM_RESOURCEMISSING	0xC029004A
STATUS_TPM_DELEGATE_LOCK	0xC029004B
STATUS_TPM_DELEGATE_FAMILY	0xC029004C
STATUS_TPM_DELEGATE_ADMIN	0xC029004D
STATUS_TPM_OWNER_CONTROL	0xC029004F
STATUS_TPM_DAA_RESOURCES	0xC0290050
STATUS_TPM_DAA_INPUT_DATA0	0xC0290051
STATUS_TPM_DAA_INPUT_DATA1	0xC0290052

Status	Valeur
STATUS_TPM_DAA_ISSUER_SETTINGS	0xC0290053
STATUS_TPM_DAA_TPM_SETTINGS	0xC0290054
STATUS_TPM_DAA_STAGE	0xC0290055
STATUS_TPM_DAA_ISSUER_VALIDITY	0xC0290056
STATUS_TPM_DAA_WRONG_W	0xC0290057
STATUS_TPM_BAD_HANDLE	0xC0290058
STATUS_TPM_BAD_DELEGATE	0xC0290059
STATUS_TPM_BADCONTEXT	0xC029005A
STATUS_TPM_TOOMANYCONTEXTS	0xC029005B
STATUS_TPM_MA_TICKET_SIGNATURE	0xC029005C
STATUS_TPM_MA_DESTINATION	0xC029005D
STATUS_TPM_MA_SOURCE	0xC029005E
STATUS_TPM_MA_AUTHORITY	0xC029005F
STATUS_TPM_PERMANENTEK	0xC0290061
STATUS_TPM_BAD_SIGNATURE	0xC0290062
STATUS_TPM_NOCONTEXTSPACE	0xC0290063
STATUS_TPM_COMMAND_BLOCKED	0xC0290400
STATUS_TPM_INVALID_HANDLE	0xC0290401
STATUS_TPM_DUPLICATE_VHANDLE	0xC0290402
STATUS_TPM_RETRY	0xC0290800
STATUS_TPM_NEEDS_SELFTEST	0xC0290801
STATUS_TPM_DOING_SELFTEST	0xC0290802
STATUS_TPM_DEFEND_LOCK_RUNNING	0xC0290803
STATUS_TPM_COMMAND_CANCELED	0xC0291001
STATUS_TPM_TOO_MANY_CONTEXTS	0xC0291002
STATUS_TPM_NOT_FOUND	0xC0291003
STATUS_TPM_ACCESS_DENIED	0xC0291004
STATUS_TPM_INSUFFICIENT_BUFFER	0xC0291005
STATUS_PCP_ERROR_MASK	0xC0292000
STATUS_PCP_DEVICE_NOT_READY	0xC0292001
STATUS_PCP_INVALID_HANDLE	0xC0292002
STATUS_PCP_INVALID_PARAMETER	0xC0292003
STATUS_PCP_FLAG_NOT_SUPPORTED	0xC0292004
STATUS_PCP_NOT_SUPPORTED	0xC0292005
STATUS_PCP_BUFFER_TOO_SMALL	0xC0292006
STATUS_PCP_INTERNAL_ERROR	0xC0292007
STATUS_PCP_POLICY_NOT_FOUND	0xC029200A
STATUS_PCP_PROFILE_NOT_FOUND	0xC029200B
STATUS_PCP_VALIDATION_FAILED	0xC029200C
STATUS_HV_INVALID_ALIGNMENT	0xC0350004
STATUS_HV_INVALID_PARAMETER	0xC0350005
STATUS_HV_ACCESS_DENIED	0xC0350006
STATUS_HV_OPERATION_DENIED	0xC0350008
STATUS_HV_UNKNOWN_PROPERTY	0xC0350009
STATUS_HV_INSUFFICIENT_MEMORY	0xC035000B
STATUS_HV_PARTITION_TOO_DEEP	0xC035000C
STATUS_HV_INVALID_PARTITION_ID	0xC035000D

Status	Valeur
STATUS_HV_INVALID_VP_INDEX	0xC035000E
STATUS_HV_INVALID_PORT_ID	0xC0350011
STATUS_HV_INVALID_CONNECTION_ID	0xC0350012
STATUS_HV_INSUFFICIENT_BUFFERS	0xC0350013
STATUS_HV_NOT_ACKNOWLEDGED	0xC0350014
STATUS_HV_ACKNOWLEDGED	0xC0350016
STATUS_HV_INVALID_SYNIC_STATE	0xC0350018
STATUS_HV_OBJECT_IN_USE	0xC0350019
STATUS_HV_NO_DATA	0xC035001B
STATUS_HV_INACTIVE	0xC035001C
STATUS_HV_NO_RESOURCES	0xC035001D
STATUS_HV_FEATURE_UNAVAILABLE	0xC035001E
STATUS_HV_INVALID_LP_INDEX	0xC0350041
STATUS_HV_NOT_PRESENT	0xC0351000
STATUS_VID_DUPLICATE_HANDLER	0xC0370001
STATUS_VID_TOO_MANY_HANDLERS	0xC0370002
STATUS_VID_QUEUE_FULL	0xC0370003
STATUS_VID_HANDLER_NOT_PRESENT	0xC0370004
STATUS_VID_INVALID_OBJECT_NAME	0xC0370005
STATUS_VID_MB_STILL_REFERENCED	0xC037000D
STATUS_VID_PAGE_RANGE_OVERFLOW	0xC0370013
STATUS_VID_INVALID_PPM_HANDLE	0xC0370018
STATUS_VID_MBPS_ARE_LOCKED	0xC0370019
STATUS_VID_MESSAGE_QUEUE_CLOSED	0xC037001A
STATUS_VID_STOP_PENDING	0xC037001C
STATUS_VID_MMIO_RANGE_DESTROYED	0xC0370021
STATUS_VID_SAVED_STATE_CORRUPT	0xC0370027
STATUS_IPSEC_BAD_SPI	0xC0360001
STATUS_IPSEC_WRONG_SA	0xC0360003
STATUS_IPSEC_INVALID_PACKET	0xC0360005
STATUS_IPSEC_CLEAR_TEXT_DROP	0xC0360007
STATUS_IPSEC_AUTH_FIREWALL_DROP	0xC0360008
STATUS_IPSEC_THROTTLE_DROP	0xC0360009
STATUS_IPSEC_DOSP_BLOCK	0xC0368000
STATUS_IPSEC_DOSP_MAX_ENTRIES	0xC0368004
STATUS_VOLMGR_DATABASE_FULL	0xC0380001
STATUS_VOLMGR_DISK_DUPLICATE	0xC0380006
STATUS_VOLMGR_DISK_DYNAMIC	0xC0380007
STATUS_VOLMGR_DISK_ID_INVALID	0xC0380008
STATUS_VOLMGR_DISK_INVALID	0xC0380009
STATUS_VOLMGR_DISK_LAST_VOTER	0xC038000A
STATUS_VOLMGR_DISK_MISSING	0xC0380011
STATUS_VOLMGR_DISK_NOT_EMPTY	0xC0380012
STATUS_VOLMGR_MEMBER_IN_SYNC	0xC0380023
STATUS_VOLMGR_MEMBER_MISSING	0xC0380026
STATUS_VOLMGR_ALL_DISKS_FAILED	0xC0380029
STATUS_VOLMGR_NO_SUCH_USER	0xC038002B

Status	Valeur
STATUS_VOLMGR_PACK_DUPLICATE	0xC038002F
STATUS_VOLMGR_PACK_ID_INVALID	0xC0380030
STATUS_VOLMGR_PACK_INVALID	0xC0380031
STATUS_VOLMGR_PACK_NAME_INVALID	0xC0380032
STATUS_VOLMGR_PACK_OFFLINE	0xC0380033
STATUS_VOLMGR_PACK_HAS_QUORUM	0xC0380034
STATUS_VOLMGR_PLEX_IN_SYNC	0xC0380038
STATUS_VOLMGR_PLEX_LAST_ACTIVE	0xC038003B
STATUS_VOLMGR_PLEX_MISSING	0xC038003C
STATUS_VOLMGR_PLEX_REGENERATING	0xC038003D
STATUS_VOLMGR_PLEX_TYPE_INVALID	0xC038003E
STATUS_VOLMGR_PLEX_NOT_RAID5	0xC038003F
STATUS_VOLMGR_PLEX_NOT_SIMPLE	0xC0380040
STATUS_VOLMGR_VOLUME_ID_INVALID	0xC0380046
STATUS_VOLMGR_VOLUME_OFFLINE	0xC038004B
STATUS_VOLMGR_VOLUME_RETAINED	0xC038004C
STATUS_VOLMGR_BAD_BOOT_DISK	0xC038004F
STATUS_VOLMGR_NOT_PRIMARY_PACK	0xC0380052
STATUS_VOLMGR_VOLUME_MIRRORED	0xC0380056
STATUS_BCD_TOO_MANY_ELEMENTS	0xC0390002
STATUS_VHD_DRIVE_FOOTER_MISSING	0xC03A0001
STATUS_VHD_DRIVE_FOOTER_CORRUPT	0xC03A0003
STATUS_VHD_FORMAT_UNKNOWN	0xC03A0004
STATUS_VHD_INVALID_BLOCK_SIZE	0xC03A000B
STATUS_VHD_BITMAP_MISMATCH	0xC03A000C
STATUS_VHD_PARENT_VHD_NOT_FOUND	0xC03A000D
STATUS_VHD_INVALID_SIZE	0xC03A0012
STATUS_VHD_INVALID_FILE_SIZE	0xC03A0013
STATUS_VIRTUAL_DISK_LIMITATION	0xC03A001A
STATUS_VHD_INVALID_TYPE	0xC03A001B
STATUS_VHD_INVALID_STATE	0xC03A001C
STATUS_VHD_METADATA_FULL	0xC03A0028
STATUS_QUERY_STORAGE_ERROR	0x803A0001
STATUS_DIS_NOT_PRESENT	0xC03C0001
STATUS_DIS_ATTRIBUTE_NOT_FOUND	0xC03C0002
STATUS_DIS_PARTIAL_DATA	0xC03C0004
STATUS_RKF_KEY_NOT_FOUND	0xC0400001
STATUS_RKF_DUPLICATE_KEY	0xC0400002
STATUS_RKF_BLOB_FULL	0xC0400003
STATUS_RKF_STORE_FULL	0xC0400004
STATUS_RKF_FILE_BLOCKED	0xC0400005
STATUS_RKF_ACTIVE_KEY	0xC0400006
STATUS_RDBSS_RESTART_OPERATION	0xC0410001
STATUS_RDBSS_CONTINUE_OPERATION	0xC0410002
STATUS_RDBSS_POST_OPERATION	0xC0410003
STATUS_BTH_ATT_INVALID_HANDLE	0xC0420001
STATUS_BTH_ATT_INVALID_PDU	0xC0420004

Status	Valeur
STATUS_BTH_ATT_INVALID_OFFSET	0xC0420007
STATUS_BTH_ATT_UNLIKELY	0xC042000E
STATUS_BTH_ATT_UNKNOWN_ERROR	0xC0421000
STATUS_SECUREBOOT_NOT_ENABLED	0x80430006
STATUS_SECUREBOOT_FILE_REPLACED	0xC0430007
STATUS_SPACES_NOT_ENOUGH_DRIVES	0xC0E7000B

Chaînes Unicode

De nombreuses fonctions Windows oeuvrant de concert avec des chaînes attendent une structure de type UNICODE_STRING.

```
typedef struct _UNICODE_STRING {
  USHORT Length;
  USHORT MaximumLength;
  PWCH Buffer;
} UNICODE_STRING;
typedef UNICODE_STRING *PUNICODE_STRING;
typedef const UNICODE_STRING *PCUNICODE_STRING;
```

L'attribut `Length` indique le nombre d'octets que totalise la chaîne. Cette valeur ne tient pas compte du caractère nul de terminaison - lequel n'est dans la perspective Unicode pas obligatoire. L'attribut `MaximumLength` enregistre le nombre jusqu'auquel peut croitre la chaine sans nécessiter un relogement mémoire. Cela inclut la taille de la chaîne plus un octet pour le caractère nul de terminaison, s'il existe. Enfin, l'attribut `Buffer` mémorise l'adresse de départ du tampon concerné.

Le tableau suivant répertorie certaines des fonctions courantes utilisables avec des chaînes Unicode.

Opération	Fonction Unicode	Équivalent ANSI
Mesure	wcslen	strlen
Concaténation	RtlAppendUnicodeToString, RtlAppendUnicodeStringToString, wcscat	strcat
Copie	RtlCopyUnicodeString, wcscpy	RtlCopyString, strcpy
Comparaison	RtlCompareUnicodeString	RtlCompareString
Inversion	_wcsrev	_strrev
Initialisation	RtlInitUnicodeString	RtlInitAnsiString, RtlInitString
Conversion	RtlUnicodeStringToAnsiString	RtlAnsiStringToUnicodeString
Libération	RtlFreeUnicodeString	RtlFreeAnsiString

Objets et handles

Windows emploie pour la représentation et la manipulation des ressources clé du système d'exploitation deux notions concomitantes mais complémentaires, incarnées l'une par les *objets*, l'autre par les *handles*.

Au plus haut niveau d'abstraction, un objet est un ensemble plus ou moins fermé (vu qu'extensible) regroupant les propriétés définitoires d'une même catégorie d'entités. L'objet de type processus, par exemple, contient toutes les informations de contrôle requises pour la gestion de tels éléments - et *un* objet processus (comprendre un en particulier) représente une instance donnée d'un objet processus. Il en va de même pour les threads, fichiers, périphériques, pilotes, clés de registre, et bien d'autres.

Si, en surface, peu de signes objectifs conviennent pour établir une distinction claire entre objets et structure de données ordinaire, quelques différences fondamentales les séparent. D'une part, la structure interne d'un objet est masquée. Il est pour cette raison indispensable avant d'interagir avec un objet de solliciter les services prévus à cet effet. D'autre part, l'implémentation du code de gestion des objets (générique par nature) étant disjointe du

code utilisant l'objet (qui elle est spécifique aux objets d'un même type), il n'est possible de lire ou de modifier directement les données d'un objet sans le faire en violation des politiques instaurées en matière de rétention des objets, qui précisent quand un objet est automatiquement détruit.

Les structures de données du système Windows ne sont pas toutes des objets. Seules les ressources qu'il faut partager, protéger, nommer ou rendre visibles aux programmes en mode utilisateur voient leur fonctionnalité exprimés en tant que tels. Notez également que certains ouvrages ou documentations ont à cet égard une définition plus ample de ce qu'est un objet, et vont jusqu'à utiliser ce concept pour faire référence à des structures communes en mode utilisateur, par exemple les sections critiques. Dans ce livre, nous utilisons le terme *objet* dans son sens le plus strict, à l'aune des principes exposés plus tôt.

Les processus mode utilisateur souhaitant manipuler les ressources vers lesquelles les objets fournissent un tremplin n'accèdent pas directement aux structures de données stockées dans l'espace mémoire du noyau, mais ont pour obligation d'obtenir un handle sur l'objet. Au plus large degré d'abstraction, un handle (ou identificateur d'objet) est une valeur opaque qui fait office de référence à une instance d'un objet en particulier.

De nombreuses fonctions au sein de l'API Windows conduisent à la création d'objets noyau de tout type (processus, thread, fichier, etc.) et renvoient alors à l'appelant un handle, lequel permet par la suite d'effectuer des opérations sur l'objet. Les handles vers un objet donné sont spécifiques au processus initiateur de l'objet. Il est cependant possible pour un processus, sous réserve que les conditions de sécurité s'y prêtent, de dupliquer un handle relevant d'un autre processus, et de la sorte se voir conférer un accès au même objet.

Le mécanisme d'objets et de handles a notamment les avantages suivants :

- Favoriser l'uniformité d'accès aux objets du noyau en proposant un intermédiaire commun pour accéder aux ressources qu'ils décrivent, cela sous forme d'un unique type de données (handle en l'occurence).
- Simplifier la mise à des jour des structures de données internes du système d'exploitation tout en assurant la rétro compatibilité (comprendre sans modifier les interfaces de programmation rendues visibles aux concepteurs de logiciels).
- Représenter facilement des droits d'accès différents pour un même objet (un processus peut exemple acquérir deux handles sur le même fichier, l'un en lecture seule, l'autre en écriture).

Interface binaire-programme

L'interface binaire-programme (ABI, application binary interface) de Windows définit les procédés bas niveau en oeuvre entre le système d'exploitation et les autres logiciels, dont applications, images natives et bibliothèques. L'ABI détermine quelles conventions d'appels peuvent être utilisées, et comment les arguments et valeurs de fonctions doivent être traités. A plus grande échelle, l'ABI couvre l'ensemble des fonctions du système, elle fixe outre les modalités opérationnelles des applications, liés directement au code machine exécutable, certaines d'ordre plus général, comme le format natif des images exécutables, ou encore la valeur, la taille et l'alignement des données.

Mot machine et types fondamentaux

Le mot machine est un concept en informatique théorique définissant la quantité de données qu'une machine peut traiter en une seule fois. Un mot machine est un nombre entier, une mesure conditionnant les tailles sur lesquelles opèrent les composants de l'architecture tels que registre processeur ou bus mémoire. Typiquement, la taille de l'espace d'adressage est égale à celle d'un mot machine, conséquemment celle d'un pointeur.

Confondu parfois avec le mot machine, le mot (notez l'absence de connotation à un matériel) est une désignation arbitraire faite par la plupart des systèmes d'exploitation, répartissant les tailles standards des entiers entre octets (8 bits), mots (16 bits), double mots (32 bits) et quadruple mots (64 mots).

Le langage de programmation C manquant d'un mécanisme pour ajouter de nouveaux types de données fondamentales (les spécifications du C dictent seulement les tailles minimales des types fondamentaux, à

l'exception du type char toujours encodés sur 8 bits d'information), chaque fois qu'une nouvelle architecture processeur a étendu les capacités de l'espace d'adressage, il a fallu repenser la définition des types existants ou ajouter de nouveaux types (non fondamentaux). Windows utilise les deux procédés.

Trois modèles peuvent être choisis afin de changer la carte des types fondamentaux : LP64, ILP64 et LLP64. Ces notations décrivent la quantité de stockage nécessaire aux types de données de base. La notation LP64 montre que les types long et pointer sont encodés sur 64 bits. ILP64 signifie que ce sont les types int, long et pointer qui le sont. LLP64 assigne 64 bits à un nouveau type de données long long) et aux pointeurs. Le tableau x indique la quantité de stockage nécessaire pour les types fondamentaux selon le modèle choisi. A titre informatif, on a également indiqué les notations LP32 et ILP32, cette dernière étant celle utilisée par la plupart des systèmes 32 bits actuels, où les types int, long et pointeur sont tous de même largeur (32 bits).

Tableau 2.6. Quantité de stockage des types fondamentaux

Type	LP64	ILP64	LLP64	ILP32	LP32
char	8	8	8	8	8
short	16	16	16	16	16
int	32	64	32	32	16
long	64	64	32	32	32
long long	64	64	64	32	32
pointer	64	64	64	32	32

Microsoft Management Console (MMC)

Microsoft Management Console (MMC) est une infrastructure logicielle commune à divers utilitaires d'administration système créés par Microsoft et d'autres éditeurs. Destiné à centraliser et simplifier les tâches quotidiennes de gestion de l'ordinateur, MMC agit comme conteneur pour des interfaces graphiques de configuration, fournissant à ce titre un moyen commode de visualiser l'état et de manipuler les composants logiciels, matériels et réseau de Windows. Servant de base à de nombreux outils incorporés dans Windows, tels Gestion de l'ordinateur, MMC permet de créer des interfaces utilisateur plus flexibles et de personnaliser les outils d'administration en regroupant leur fonctionnalité dans une fenêtre unique.

MMC comporte plusieurs outils qu'elle affiche sous forme de consoles. Ces outils, composés d'une ou de plusieurs applications, sont construits à partir de modules appelés composants logiciels enfichables. Ces composants peuvent eux-mêmes contenir d'autres composants logiciels enfichables d'extension. Les extensions livrées avec Windows permettent notamment de contrôler l'état des périphériques, lire les journaux d'activité, mesurer l'utilisation du processeur et de la mémoire, configurer des imprimantes, installer des logiciels, manipuler les services, planifier l'exécution en traitement par lots et manipuler les comptes utilisateurs.

L'image native correspondant à Microsoft Management Console est mmc.exe, située dans le répertoire %SystemRoot%\System32. Le tableau suivant énumère quelques-unes des consoles intégrées.

Nom anglais	Nom français	Console
Certificates	Certificats	certmgr.msc
Certificates template	Modèles de certificats	certtmpl.msc
Indexing Service	Service d'indexation	ciadv.msc
Component Services	Services des composants	comexp.msc
Computer Management	Gestion de l'ordinateur	compmgmt.msc
Device Manager	Gestionnaire de périphériques	devmgmt.msc
Disk Defragmenter	Défragmenteur de disque	dfrg.msc
Disk Management	Gestion des disques	diskmgmt.msc
Domain name system management	Gestionnaire DNS	dnsmgmt.msc
Event Viewer	Observateur d'événements	eventvwr.msc
Shared Folders	Dossiers partagés	fsmgmt.msc

Nom anglais	Nom français	Console
Group Policy	Stratégie de groupe	gpedit.msc
Local Users and Groups	Utilisateurs et groupes locaux	lusrmgr.msc
NAP Client Configuration	Configuration du client NAP	napclcfg.msc
Removable Storage	Stockage amovible	ntmsmgr.msc
Performance	Performance	perfmon.msc
Print Management	Gestion de l'impression	printmanagement.msc
Resultant Set of Policy	Jeu de stratégie résultant	rsop.msc
Local Security Settings	Paramètres de sécurité locaux	secpol.msc
Services	Services	services.msc
Task Scheduler	Planificateur de tâches	taskschd.msc
WMI Management	Contrôle WMI	wmimgmt.msc

Composants logiciels enfichables

Ainsi que nous l'avons déjà suggéré, MMC tient moins du logiciel autonome que de l'environnement de gestion partagé inter-applications. MMC est ainsi une application qui, en elle-même, ne fournit aucun service particulier, mais dans laquelle d'autres outils appelés *composants logiciels enfichables* peuvent être installés et utilisés. Chacun de ces composants met en oeuvre une fonctionnalité d'administration de base. Lorsqu'un ou plusieurs composants logiciels enfichables sont installés dans MMC, le résultat est alors appelé une *console*. Un jeu de consoles standard est livré avec tout ordinateur exécutant Windows, pour la plupart accessible depuis le groupe de programmes Outils d'administration.

Un composant enfichable peut posséder une ou plusieurs *extensions* (certaines elles-mêmes enfichables). La console Gestion de l'ordinateur, par exemple, qui ouvre l'accès à des outils majeurs d'administration du système, des services, et du stockage, est constituée de nombreuses extensions : Observateur d'événements, Planificateur de tâches, Utilisateurs et groupes locaux, et d'autres.

Par défaut, les fichiers correspondants à des composants enfichables portent l'extension .msc et se situent sous le répertoire %SystemRoot%\System32.

Modes d'accès d'une console MMC

Lorsque vous créez une console de composant logiciel enfichable personnalisée, vous avez le choix entre deux options d'accès : le mode Auteur, qui ouvre l'accès à toutes les tâches envisageables depuis une MMC, et le mode Utilisateur, qui comporte trois niveaux de restrictions concernant l'utilisation de la console. En définitive, quatre options différentes pour l'accès par défaut à la console sont ainsi disponibles.

- *Mode auteur* Les utilisateurs peuvent ajouter ou supprimer des composants logiciels enfichables, créer des fenêtres, afficher toutes les parties de l'arborescence de la console ainsi qu'enregistrer toutes les modifications apportées.
- *Mode utilisateur - accès total* Les utilisateurs peuvent se déplacer dans la console, créer et ouvrir de nouvelles fenêtres mais pouvoir les enregistrer, ni même ajouter ou supprimer des composants enfichables.
- *Mode utilisateur - accès limité, fenêtre multiple* Les utilisateurs peuvent créer de nouvelles fenêtres mais ne peuvent fermer aucune de celles déjà existantes - autrement dit les parties de l'arborescence qui étaient visibles lorsque le fichier de console a été enregistré.
- *Mode utilisateur - accès limité, fenêtre unique* Identique au mode précédent mais les utilisateurs n'ont pas le droit d'afficher plusieurs fenêtres.

Dans l'éventualité où vous rencontreriez une console bridée par l'un de ces modes, vous pouvez (moyennant des privilèges suffisants) forcer l'ouverture en mode auteur (voir section suivante) ou configurer les options idoines depuis la boite de dialogue Options de MMC.

Exécuter une console en mode Auteur

Les consoles MMC peuvent s'exécuter en mode Auteur ou dans trois variantes du mode Utilisateur. Le mode Auteur permet une personnalisation intégrale de la console de composant logiciel enfichable, en donnant notamment la possibilité d'ajouter ou supprimer des composants logiciels enfichables, de créer des fenêtres, et d'accéder à tous les menus ainsi qu'à toutes les options (comprendre les options des boîtes de dialogue Personnalisation de l'affichage et Options).

Par défaut, chaque console MMC s'exécute dans le mode où elle fonctionnait la dernière fois qu'elle a été enregistrée. Ceci dit, vous pouvez toujours exécuter une console dans le mode de votre choix.

Pour exécuter une console en mode Auteur, faites un clic droit sur le fichier la représentant dans une fenêtre de l'Explorateur Windows et choisissez Auteur dans le menu contextuel. Il est également possible afin de parvenir aux mêmes effets d'impliquer le commutateur de ligne de commande /a. Cela donne : nom.msc /a, où *nom* est le nom du fichier de la console. Exécuter une console pour un ordinateur distant

De nombreuses consoles MMC prédéfinies sont configurées pour fonctionner sur la station de travail locale par défaut, mais peuvent aussi - moyennant les autorisations appropriées - être employées pour gérer des ordinateurs distants. Dans un tel cas de figure, utilisez la syntaxe de ligne de commande suivante : console.msc /computer=nomordinateur.

Invite de commandes (Cmd)

Démarrer des programmes

Une particularité intéressante (quoique peu extraordinaire) de l'Invite de commandes est de pouvoir démarrer toutes sortes de programmes, qu'ils soient ou non dédiés à une utilisation console ou graphique. Il n'est de ce fait pas nécessaire de connaitre la nature d'une application pour la voir se réaliser : un programme en mode caractère s'exécutera dans la fenêtre Invite de commandes depuis laquelle il a été sollicité, un programme Windows dans sa propre fenêtre externe.

Dans les versions de Windows précédant XP, si vous exécutiez un programme Windows depuis une Invite de commandes, la session correspondante restait inaccessible jusqu'à ce que le logiciel ait terminé son exécution. Si, pour une raison quelconque, une de vos applications reposait sur ce comportement, exécutez-la au moyen de la commande start accompagnée du commutateur /wait.

Syntaxe de la ligne de commande Cmd

La syntaxe complète de la ligne de commande associée à Cmd est la suivante :

```
cmd [/a | /u] [/q] [/d] [/e:on | /e:off] [/f:on | /f:off] [/v:on | /v:off] [[/s] [/c | /k] chaînecommandes]
```

- **a | u** Système d'encodage employé pour la mise en forme de la sortie : ANSI (a) ou Unicode (u).
- **e:on|off** Contrôle les extensions de commande.
- **f:on|off** Recommandations concernant les caractères qui complètent les noms de fichiers et de dossiers. Cmd /f:on démarre une session d'Invite de commandes avec CTRL+D comme caractère pour compléter un chemin et CTRL+F pour compléter un nom de fichier, désactivant ainsi les paramètres enregistrés dans le Registre à cet égard (valeurs CompletionChar et PathCompletionChar sous HKCU\Software\Microsoft \Command Processor pour l'utilisateur courant, et même clés dans HKLM pour tous les utilisateurs, touche TAB par défaut). Cmd /f:off démarre une session sans caractère attribué, quels que soient les options stipulées dans le Registre.
- **c | k** Exécute une chaîne de commandes spécifiée en paramètre. La différence entre ces deux commutateurs tient au fait que l'un (/c) a pour effet de terminer la session d'Invite de commandes dès que la chaine de commandes s'est achevée, tandis que l'autre (/k) entraine un maintien de la session.
- **v:on|off** Contrôle l'expansion différée des variables d'environnement.

Commandes de base

Le tableau qui suit répertorie les différentes commandes et outils accessibles depuis l'invite de commandes. Si vous avez un doute sur la syntaxe à utiliser pour une commande particulière, n'hésitez pas à faire appel à la commande help.

ASSOC	**Affiche ou modifie les applications associées aux extensions de fichiers.**
ATTRIB	Affiche ou modifie les attributs d'un fichier.
BREAK	Active ou désactive le contrôle étendu de CTRL+C.
BCDEDIT	Définit les propriétés dans la base de données de démarrage pour le contrôle du chargement d'amorçage.
CACLS	Affiche ou modifie les listes de contrôles d'accès aux fichiers.
CALL	Appelle un fichier de commandes à partir d'un autre fichier de commandes.
CD	Modifie le répertoire ou affiche le répertoire actif.
CHCP	Modifie ou affiche le numéro de la page de code active.
CHDIR	Modifie le répertoire ou affiche le nom du répertoire actif.
CHKDSK	Vérifie un disque et affiche un rapport d'état.
CHKNTFS	Affiche ou modifie la vérification du disque au démarrage.
CLS	Efface l'écran.
CMD	Exécute une nouvelle instance de l'interpréteur de commandes de Windows.
COLOR	Modifie les couleurs du premier plan et de l'arrière-plan de la console.
COMP	Compare les contenus de deux fichiers ou groupes de fichiers.
COMPACT	Modifie ou affiche la compression des fichiers sur une partition NTFS.
CONVERT	Convertit des volumes FAT en volumes NTFS. Vous ne pouvez pas convertir le lecteur en cours d'utilisation.
COPY	Copie un ou plusieurs fichiers.
DATE	Affiche ou définit la date.
DEL	Supprime un ou plusieurs fichiers.
DIR	Affiche la liste des fichiers et des sous-répertoires d'un répertoire.
DISKCOMP	Compare les contenus de deux disquettes.
DISKCOPY	Copie le contenu d'une disquette sur une autre.
DISKPART	Affiche ou configure les propriétés d'une partition de disque.
DOSKEY	Modifie les lignes de commande, rappelle des commandes Windows, et crée des macros.
DRIVERQUERY	Affiche l'état et les propriétés du pilote de périphérique en cours d'utilisation.
ECHO	Affiche des messages ou active/désactive l'affichage des commandes.
ENDLOCAL	Stoppe la localisation des modifications d'environnement dans un fichier de commandes.
ERASE	Supprime un ou plusieurs fichiers.
EXIT	Quitte l'interpréteur de commandes (CMD.EXE).
FC	Compare deux fichiers ou groupes de fichiers et affiche les différences.
FIND	Recherche une chaîne de caractères dans un ou plusieurs fichiers.
FINDSTR	Cherche des chaînes dans les fichiers.
FOR	Exécute une commande sur chaque fichier d'un ensemble de fichiers.

ASSOC	**Affiche ou modifie les applications associées aux extensions de fichiers.**
FORMAT	Formate un disque devant être utilisé avec Windows.
FSUTIL	Affiche ou configure les propriétés du système de fichiers.
FTYPE	Affiche ou modifie les types de fichiers utilisés dans les associations d'extensions.
GOTO	Indique l'exécution d'un fichier de commandes pour une ligne identifiée par une étiquette.
GPRESULT	Affiche les informations de stratégie de groupe pour un ordinateur ou un utilisateur.
GRAFTABL	Permet à Windows d'afficher un jeu de caractères en mode graphique.
HELP	Affiche des informations sur les commandes de Windows.
ICACLS	Afficher, modifier, sauvegarder ou restaurer les listes de contrôle d'accès pour les fichiers et les répertoires.
IF	Effectue un traitement conditionnel dans un fichier de commandes.
LABEL	Crée, modifie ou supprime le nom de volume d'un disque.
MD	Crée un répertoire.
MKDIR	Crée un répertoire.
MKLINK	Créer des liens symboliques et des liens physiques
MODE	Configure un périphérique du système.
MORE	Affiche la sortie écran par écran.
MOVE	Déplace un ou plusieurs fichiers d'un répertoire à un autre.
OPENFILES	Affiche les fichiers partagés ouverts à distance par les utilisateurs.
PATH	Affiche ou définit le chemin de recherche des fichiers exécutables.
PAUSE	Interrompt l'exécution d'un fichier de commandes et affiche un message.
POPD	Restaure la valeur précédente du répertoire actif enregistrée par PUSHD.
PRINT	Imprime un fichier texte.
PROMPT	Modifie l'invite de commande de Windows.
PUSHD	Enregistre le répertoire actif puis le modifie.
RD	Supprime un répertoire.
RECOVER	Récupère l'information lisible d'un disque défectueux.
REM	Insère un commentaire dans un fichier de commandes ou CONFIG.SYS.
REN	Renomme un ou plusieurs fichiers.
RENAME	Renomme un ou plusieurs fichiers.
REPLACE	Remplace des fichiers.
RMDIR	Supprime un répertoire.
ROBOCOPY	Utilitaire avancé pour copier les fichiers et les arborescences de répertoires
SET	Affiche, définit ou supprime des variables d'environnement Windows.
SETLOCAL	Commence la localisation des modifications d'environnement dans un fichier de commandes.
SC	Affiche ou configure les services (processus en arrière-plan).
SCHTASKS	Planifie les commandes et les programmes à exécuter sur l'ordinateur.
SHIFT	Modifie la position des paramètres remplaçables dans un fichier de commandes.
SHUTDOWN	Permet un arrêt local ou distant correct de l'ordinateur.

ASSOC	**Affiche ou modifie les applications associées aux extensions de fichiers.**
SORT	Trie les entrées.
START	Ouvre une fenêtre séparée pour l'exécution d'un programme ou d'une commande spécifique.
SUBST	Associe un chemin d'accès à une lettre de lecteur.
SYSTEMINFO	Affiche les propriétés et la configuration spécifiques de l'ordinateur.
TASKLIST	Affiche toutes les tâches en cours d'exécution, y compris les services.
TASKKILL	Termine ou interrompt un processus ou une application en cours d'exécution.
TIME	Affiche ou définit l'heure du système.
TITLE	Définit le titre de la fenêtre pour une session CMD.EXE.
TREE	Affiche le graphisme de la structure de répertoire d'un lecteur ou d'un chemin d'accès.
TYPE	Affiche le contenu d'un fichier texte.
VER	Affiche la version de Windows.
VERIFY	Demande à Windows de vérifier si vos fichiers sont correctement écrits sur le disque.
VOL	Affiche le nom et le numéro de série d'un volume de disque.
XCOPY	Copie les fichiers et les arborescences de répertoires.
WMIC	Affiche les informations WMI dans l'interface de commande interactive.

Interprètes de commandes

Le rôle premier d'un interprète de commandes consiste à fournir une interface d'accès en *mode caractère* à tout ou partie des fonctionnalités du système d'exploitation. La mise en oeuvre de cet objectif s'effectue sur le plan concret en deux temps : (1) l'analyse syntaxique d'une commande par examen des mots et caractères la constituant, cela afin d'en isoler les options et les arguments, et (2) le chargement de ladite commande pour exécution au sein d'un processus tiers.

S'ils occupent une place moins centrale que dans d'autres systèmes d'exploitation (Unixoïdes surtout), il existe plusieurs interprètes de commandes populaires pour Windows. Parmi eux on trouve :

- *Cmd* Cmd est un logiciel d'interprétation des commandes Windows (et éventuellement DOS) affichant une interface utilisateur en mode texte. Processeur de commandes historique de Windows, ce logiciel fournit une interface pratique pour exécuter des tâches courantes, telles que commandes externes, programmes batch et d'autres exécutables. Avec l'enracinement de plus en plus marqué de PowerShell dans Windows, on assiste depuis quelques années à une diminution progressive de l'importance accordée à Cmd.

- *PowerShell* Suite logicielle intégrant à la fois une invite de commandes interactives et un puissant langage de scripts, PowerShell permet de piloter localement ou à distance l'ensemble des services offerts sur des systèmes Microsoft.

- *Bash* Bash (acronyme de Bourne-Again shell) est l'interprète par défaut sur de nombreux Unix libres, notamment les environnements GNU/Linux. Grâce à un partenariat entre Microsoft et Canonical, il est depuis Windows 10 possible de disposer d'un Bash Ubuntu natif, et par cet intermédiaire de profiter de certains avantages de Linux sans pour autant faire cohabiter les deux systèmes d'exploitation (*double boot*) ou utiliser des technologies de virtualisation.

- *Autres* Plusieurs logiciels conçues originellement pour les systèmes POSIX (tels que les systèmes GNU/Linux, BSD, et Unix) proposent une version compilée pour Windows - ou, a minima, laissent à des tiers le soin d'en fournir une. C'est, par exemple, le cas de Tcsh, Bash, et d'autres. Une alternative consiste à utiliser Cygwin, Gnu On Windows, ou toute autre application du même style.

A titre informatif, notez que les logiciels que nous venons de passer en revue ne sont que quelques-unes des formes de l'invite de commandes dans Windows. Les autres comportent le contrôle Exécuter (commande Run), la barre d'adresses de l'explorateur Windows, et même celle du navigateur Internet. En bien des points, ces invites de commande fonctionnent selon les mêmes conventions et les mêmes principes.

Le tableau suivant énumère plusieurs commandes équivalentes dans l'invite Cmd, PowerShell, et dans les shells Unix.

Cmd	Shell	PowerShell (cmdlet)	PowerShell (alias)	Description
cd	pwd	Get-Location	gl, pwd	Afficher le répertoire de travail courant
cd, chdir	cd	Set-Location	sl, cd, chdir	Changer le répertoire courant
cls	clear	Clear-Host	cls, clear	Effacer l'affichage
dir	ls	Get-ChildItem	gci, dir, ls	Afficher le contenu d'un répertoire
help	help, which	Get-Command	gcm	Obtenir une l'aide générale
help	man	Get-Help	help, man	Obtenir de l'aide aide sur une commande en particulier
kill	kill	Stop-Process	spps, kill	Arrêter un processus en cours d'exécution
move	mv	Move-Item	mi, move, mv	Déplacer un fichier ou répertoire
tasklist	ps	Get-Process	gps, ps	Afficher les processus en cours d'exécution

Fonctionnalités de Windows

Ajouter ou supprimer des fonctionnalités Windows

C'est en matière de fonctionnalités l'édition de Windows exécutée sur la station de travail qui détermine lesquelles sont fournies de base, et lesquelles sont complémentaires. Pour réviser cette liste et activer ou désactiver certaines d'entre elles, ouvrez le menu Programmes au Panneau de configuration et cliquez sur Activer ou désactiver des fonctionnalités Windows sous le titre Programmes et fonctionnalités.

La boite de dialogue Fonctionnalités de Windows indique celle présentes dans votre édition : une case coche signifie que la fonctionnalité est actuellement activée, une case blanche qu'elle est désactivée. Une case pleine (ni cochée, ni décochée) signifie qu'une partie seulement de la fonctionnalité est activée. Quelques fonctionnalités sont montrées sous forme d'une structure arborescente. Cliquez sur le signe plus à gauche de l'entrée de sorte à voir les filles d'une fonctionnalité parent. Certaines fonctionnalités ne peuvent être mises en sommeil sans que des problèmes en résultent, et un message s'affiche alors afin de vous avertir du risque encouru par une telle tentative. Les fonctionnalités liées à Internet Explorer, par exemple, ne devraient idéalement jamais quitter le système - peu importe que vous utilisiez ou non ce navigateur, ces dernières se voyant mutualisées entre plusieurs composants essentiels au bon fonctionnement de Windows.

Notez que les fonctions désactivées restent installées en vue d'une utilisation ultérieure, et n'imputent en cela en rien l'espace de stockage - du reste, compte tenu des tailles des disques durs actuels, cela n'a aucune espèce d'importance. Inhiber les moins utiles (selon vos besoins) contribue néanmoins à maximiser les performances du système en réduisant le nombre de processus.

Windows Update

Tant les utilisateurs que les administrateurs ont besoin de mécanismes efficaces et commodes d'emploi leur permettant de bénéficier des dernières améliorations et optimisations. Pour répondre à ces exigences, Windows inclue un système automatisé de mises à jour, sobrement nommé Update, qui prend en charge la montée à niveau

des logiciels s'exécutant sur la plateforme, dont le système d'exploitation, les pilotes de périphérique et quelques applications prédéfinies.

Update se connecte périodiquement à un serveur désigné - site hébergé par Microsoft (http://windowsupdate.microsoft.com) ou bien par l'entreprise - pour vérifier l'existence de nouvelles mises à jour. Selon la configuration, lorsque l'agent Update détermine que des mises à jour sont disponibles, soit il les télécharge et les installe automatiquement, soit il en informe l'utilisateur.

Windows Update classe les mises à jour logicielles selon leur niveau de criticité : importantes, recommandées et facultatives. Outre l'aspect informatif de cette nomenclature, l'agent Update prioritise les téléchargements de façon que les mises à jour soient appliquées par ordre d'importance. Cela permet aux mises à jour les plus critiques, par exemple un correctif de sécurité, d'être installées avant celles moins cruciales.

Les transferts réseau à l'initiative de Windows Update s'effectuent par l'intermédiaire de la technologie BITS (Background Intelligent Transfer Service). Ils ont de la sorte un impact négligeable sur la bande passante (plus précisément, sur l'impression que peuvent en avoir les usagers du même réseau) et peuvent être interrompus ou repris à volonté.

Pilotes de périphérique

Pour que Windows puisse fonctionner avec un matériel donné, il lui faut un pilote de périphérique adéquat. Au plus haut degré d'abstraction, les pilotes sont des programmes de contrôle qui s'intègrent nativement au système d'exploitation et orchestrent les tâches essentielles de communication relatives aux matériels de l'ordinateur, qu'il s'agisse de dispositifs scellés, par exemple un disque interne, ou de périphériques externes, tels que claviers, souris, imprimantes, et bien d'autres. Après installation d'un périphérique matériel, son pilote se charge automatiquement et s'exécute comme partie intégrante du système d'exploitation.

Les pilotes peuvent être propres à un dispositif ou à une famille de dispositifs (comme l'ensemble des imprimantes d'un même constructeur), ainsi qu'à une version spécifique d'un système d'exploitation. On parlera, par exemple, d'un pilote pour telle carte réseau fabriquée par tel constructeur et de ce modèle précis compatible ou non avec une version définie de Windows, Linux, ou autres systèmes.

En règle générale, seuls les pilotes de haut niveau font partie intégrante du système d'exploitation, par exemple le pilote NTFS pour Windows ou les pilotes ext2/3/4 pour Linux. Les autres pilotes, qui régissent des aspects aussi fondamentaux, mais exogènes, peuvent être intégrés de façon statique au noyau ou chargés à la demande sous forme de modules.

De nombreux systèmes d'exploitation, pour assurer la prise en charge d'un maximum de configuration, fournissent des versions génériques des pilotes, lesquels mettent en oeuvre les fonctions de base communes à l'ensemble des périphériques d'un même type, et couvrent ainsi un large panel d'utilisations. À titre d'exemple, il existe des pilotes génériques pour VESA (cartes vidéo), USB (périphériques de stockage), Wi-Fi (équipements réseau), et ainsi de suite. Bien que très utiles, ces pilotes, destinés uniquement à des fins générales, ne donnent de toute évidence pas accès à toutes les fonctionnalités des périphériques (par exemple, dans le cas d'une imprimante : recto-verso, double bac, etc.). C'est pourquoi il est peut-être important de mettre à jour le pilote générique d'un périphérique donné pour le remplacer par celui fourni par le constructeur. A de rares exceptions près, les ressources prises en charge par le biais de pilotes sont accessibles uniquement depuis le mode noyau. Le système d'exploitation, en tant que fournisseur d'un ensemble commun de services d'entrées/sorties, peut éventuellement permettre l'envoi de séquences de paramétrage à un périphérique depuis un programme utilisateur, ce via le mécanisme des appels système. Dans ce cas de figure, les pilotes servent tout à la fois à séparer et à unir les programmes de leur environnement d'exécution : s'il leur est impossible d'accéder de manière directe aux périphériques, ils peuvent en néanmoins en tirer parti, mais toujours sous arbitrage, et dans un contexte fortement régulé.

Panneau de configuration

Le Panneau de configuration est un composant de Microsoft Windows qui fait office de référentiel central pour l'accès aux outils de configuration du système et des applications.

Pour une clarté accrue de l'information, deux modes de visualisation sont proposés : la vue par catégorie et la vue classique. Il est possible de passer d'une option à l'autre depuis le côté supérieur droit de la fenêtre. Notez en outre la zone de recherche qui permet de rapidement se positionner sur l'applet de configuration de la fonctionnalité de votre choix.

Lors de la navigation parmi les menus du panneau de configuration, le volet gauche de la fenêtre s'adapte à l'emplacement actuel et propose des raccourcis vers des taches d'administration complémentaires.

Tableau 2.7. Applets de configuration

Applet (cpl)	Nom canonique	Description
Access.cpl	Microsoft.EaseOfAccessCenter	Options d'ergonomie
Admintools.cpl	Microsoft.AdministrativeTools	Outils d'administration
Appwiz.cpl	Microsoft.ProgramsAndFeatures	Programmes et fonctionnalités
Bthprops		Périphériques Bluetooth
Desk.cpl		Paramètres d'affichage
Firewall.cpl	Microsoft.WindowsFirewall	Pare-feu Windows Defender
Folders	Microsoft.FolderOptions	Options de l'Explorateur de fichiers
Fonts	Microsoft.Fonts	Polices
Hdwwiz.cpl	Microsoft.DeviceManager	Gestionnaire de périphériques
Inetcpl.cpl	Microsoft.InternetOptions	Options Internet
Intl.cpl	Microsoft.RegionAndLanguage	Options régionales et linguistiques
Joy.cpl	Microsoft.GameControllers	Contrôleurs de jeu
	Microsoft.Keyboard	Propriétés du clavier
Main	Microsoft.Mouse	Propriétés de la souris
Mmsys.cpl	Microsoft.Sound	Son
Nusrmgr.cpl	Microsoft.UserAccounts	Comptes d'utilisateur
Powercfg.cpl		Options d'alimentation
	Microsoft.DevicesAndPrinters	Périphériques et imprimantes
Sysdm.cpl		Propriétés système
Ncpa.cpl		Connexions réseau
Telephon.cpl		Options modems et téléphonie
Timedate	Microsoft.DateAndTime	Date et heure
Ups.cpl		Options d'alimentation
Wscui.cpl	Microsoft.ActionCenter	Sécurité et maintenance

Le panneau de configuration peut être utilisé en ligne de commande par l'intermédiaire de son application frontale (Control.exe) avec la possibilité de spécifier différentes options et paramètres. La syntaxe associée se présente sous une forme des plus classiques, soit `control /[option] [paramètre]`.

GUID

Pour repérer facilement différentes ressources au sein de l'écosystème, Windows emploie un ensemble de valeurs spéciales dite GUID (Globally Unique Identifier), lesquelles servent à la reconnaissance d'informations qui appellent à une identification sans ambiguïté, y compris documents, comptes d'utilisateurs et de groupes, comptes d'ordinateur, partitions disque, schémas d'alimentation, etc.

Concrètement, chaque GUID se présente sous la forme d'une séquence de 128 bits (16 octets), décomposée en cinq groupes (un groupe de 4 octets, trois groupes de 2 octets et enfin, 6 octets pour le dernier groupe) séparés par des traits d'union. Par exemple : 69bd62dd-3284-4e0e-b4ac-64af471cbc45.

Une propriété essentielle de tout GUID est d'être en principe unique à travers le monde. En réalité, l'algorithme à partir duquel sont issues des GUID n'offre en la matière aucune garantie. Les probabilités que deux GUID identiques puissent être générés demeurent toutefois statistiquement infinitésimales. De fait, bien que la

probabilité de collisions entre GUID existe (un même GUID désignant plusieurs objets différents), elle est suffisamment faible pour être négligeable/négligée.

Au-delà de leur unicité, les GUID ont aussi une finalité pratique clairement affirmée, puisqu'ils ils ne relèvent (et ne dépendent donc) pas d'une autorité centrale, pas plus qu'ils n'exigent de coordination entre les différents acteurs susceptibles d'émettre ce genre de valeurs.

Par nature immuables, les GUID sont générés et associés à un objet au moment de la création dudit objet. Si l'objet change de nom ou de place au sein de sa hiérarchie, le GUID est conservé.

Polices et fontes

Dans la perspective typographique, une police de caractères, ou police d'écriture, est un ensemble complet de caractères déterminés par une forme (un dessin) homogène. Les imprimeurs utilisent le terme fonte en vue de désigner spécifiquement des tailles de points, poids, graisse et styles pour une police donnée. Par exemple, Arial est une police, Arial gras 12 point une fonte. Dans le domaine informatique, les deux termes sont souvent confondus et employés de façon interchangeable.

Il existe plusieurs types de polices :

- Les polices TrueType s'affichent à l'écran telles qu'elles sont imprimées et dans toutes les tailles.
- Les polices Open Type présentent les mêmes caractéristiques que les polices TrueType mais elles sont plus complètes et il est possible de les faire pivoter
- Les polices Post Script, destinées aux professionnels, présentent une plus grande résolution à l'impression.

Services de terminal

Services de terminal fait dans la perspective Windows référence à la prise en charge de multiples sessions interactives sur une même machine. Grâce à une telle infrastructure, un utilisateur peut créer une session sur une machine distante et exécuter des applications sur le serveur. Le serveur transmet l'interface graphique au client, laquelle relaie les saisies de l'utilisateur.

La première session créée sur la station de travail locale est considérée comme étant la session console, ou session zéro, et renferme les processus qui hébergent des services. La première session de connexion correspond à la session numéro un, la seconde à la numéro deux, et ainsi de suite.

SDK et WDK

Sans doute l'avez-vous déjà entendu dire, ou lu à maintes reprises, mais un système d'exploitation n'est rien sans des logiciels capables d'exploiter au mieux ses capacités, raison pour laquelle Microsoft offre aux concepteurs de logiciels pléthore de méthodes et de moyens utiles pour créer (comprendre programmer) tout type d'application s'exécutant sur les systèmes d'exploitation Windows.

- *Software Development Kit (SDK)* Contient les fichiers d'en-tête et les bibliothèques de base nécessaires en vue créer des applications Windows, orientées bureau ou mobile. Les concepteurs de logiciels peuvent de la sorte utiliser le SDK Windows pour créer des applications utilisant le modèle de programmation natif (Win32/COM) ou managé.
- *Windows Driver Kit (WDK)* Réunit des outils pour créer, déployer, tester et déboguer des pilotes conformes à un modèle donné (WDF, KDMF, UMDF, et d'autres modèles hérités).

Convention d'appel

Les méthodes par lesquelles le système d'exploitation harmonise les relations entre différentes sous-routines, codifiant la manière dont elles reçoivent des paramètres et retournent un résultat, sont dites conventions

d'appel. Déterminant les séquences que le compilateur génère pour les appels de fonctions, ces conventions sont susceptibles de différer sur un ou plusieurs des éléments suivants :

- où les paramètres et valeurs de retour sont placés (dans les registres, sur la pile, ou un mélange des deux)
- l'ordre dans lequel s'effectue le passage des paramètres
- comment est attribuée la responsabilité du nettoyage de la pile d'appel (soit par la fonction appelante, soit par la fonction appelée)
- quels registres peuvent être préservés et lesquels sont librement utilisables

Ci-après un tableau récapitulatif offrant une vue rapide des différentes conventions d'appel les plus communément employées.

Architecture	Convention	Transmission des arguments	Responsabilité maintenance pile	Retour
x86	cdecl	Poussés sur la pile	Fonction appelante	EAX, EDX / pile
	fastcall	ECX, EDX / pile	Fonction appelée	EAX,EDX / pile
	stdcall	Poussés sur la pile	Fonction appelée	EAX,EDX / pile
x86_64	fastcall	RCX, RDX, R8, R9 / pile	Fonction appelée	RAX

Pile d'appels

D'un point de vue exclusivement structurel, chaque processus sous Windows est normalement organisé comme un ensemble de fonctions discrètes, qui s'appellent les unes les autres et, par là même, composent un tout indivisible. Lorsqu'une fonction prend fin, elle redonne le contrôle à la fonction l'ayant sollicitée (appelée dans ce contexte fonction appelante). Afin de mieux comprendre comment se concrétise ce flux, considérez l'exemple, du reste tout à fait fictif, d'une application Cipher.exe, intégrant une bibliothèque de fonctions (DLL), CipherFuncs.dll. Dans cette DLL réside une fonction nommée CipherText, chargée de chiffrer le texte transmis parmi ses paramètres d'entrées. Passées un certain nombre d'étapes préparatoires, CipherText appelle l'API Windows CryptEncryptMessage, située elle dans Crypt32.dll. Une fois que ladite API a accompli la tâche lui ayant été confiée, l'exécution revient au point où CipherText avait passé la main à CryptEncryptMessage, juste après l'appel.

La construction logicielle par laquelle le système enregistre les sollicitations auxquelles se livrent les diverses sous-parties d'un programme est dite pile d'appel (call stack). Lorsqu'une fonction est sur le point d'en appeler une autre (exigeant de ce fait un déroutement du flux d'exécution), elle place l'adresse mémoire de l'instruction suivante à exécuter au retour de la sous-fonction (autrement dit son adresse de retour) au sommet de la pile. Lorsque cette sous-fonction appelle encore une autre fonction, elle ajoute sa propre adresse de retour à la pile. Au retour d'une fonction, le système récupère l'adresse qui se trouve en haut de la pile et commence à exécuter le code à partir de ce point.

Une large majorité des utilitaires offrant la possibilité de visualiser la pile d'appel d'un programme adoptent la même convention, correspondant au schéma module!function+offset, où module est le nom du fichier exécutable hébergeant la fonction et offset le nombre d'octets (en hexadécimal) après le début de la fonction. Dans l'éventualité où le nom de la fontion n'est pas disponible, l'adresse est alors présentée sous la forme module +offset.

Au delà de fournir de précieux renseignements sur le comportement global d'un logiciel, la pile d'appels permet de retracer la séquence d'événements qui ont conduits à une situation donnée, et peut de la sorte s'avérer particulièrement utile pour mettre en lumière les causes d'un problème.

Au coeur de Windows

Une partie importante du contenu de ce livre provient d'observations concrètes menées à l'aide d'une approche par rétro-ingénierie du système d'exploitation. (La rétro-ingénierie informatique regroupe l'ensemble

des méthodes et des moyens liés à la compréhension d'un système logiciel, sans le bénéfice des spécifications originales. Elle a pour but d'analyser un système pour en créer une représentation à un plus haut niveau d'abstraction.) Maints aspects internes de Windows peuvent être mis en lumière (et les agissements exposés) à l'aide de toutes sortes d'utilitaires, tels que les outils intégrés à Windows ou les outils de débogage fournis par Microsoft. La présente section va présenter brièvement ces ressources, et le parti que l'on peut en tirer.

Pour vous encourager à explorer par vous-même les coulisses de Windows, cet ouvrage propose de nombreux exercices et mises en situation qui décrivent comment faire pour procéder à l'investigation concernant telle ou telle facette du système, et qui servent à faire remonter en surface nombre d'informations utiles pour en comprendre la logique. (Notez, sur le plan de la présentation de l'information, que ces passages sont identifiés au moyen de l'étiquette prédéfinie En pratique, ou lorsque cela est plus pertinent, entrent naturellement dans le flux textuel.) Voir concrètement comment fonctionne Windows étant tout le propos de ce livre, nous vous nous ne pouvons que vous recommander la lecture de ces passages, et encore plus de reproduire les manipulations qui y figurent. (En un mot : faites ces exercices !)

Le tableau suivant énumère les principaux outils utilisés dans ce livre, en donnant leur provenance.

Tableau 2.8. Utilitaires de visualisation des coulisses de Windows

Utilitaire	Nom de l'image	Origine
Startup Programs Viewer	AUTORUNS	Sysinternals
Global Flags	GFLAGS	Debugging tools
Débogueur noyau	WINDBG, KD	Debugging tools, Windows SDK
Moniteur de performances	PERFMON	Intégré à Windows
Moniteur de ressources	RESMON	Intégré à Windows
Process Explorer	PROCEXP	Sysinternals
Gestionnaire des tâches	TASKMGR	Intégré à Windows

Build libre et build controlé

Chaque itération majeure de Windows s'inscrit dans un modèle général qui donne communément lieu à deux déclinaisons. L'une, appelée *build libre*, correspond à la version commercialisée normale de Windows ; l'autre, appelée *build contrôlé*, est une variante spéciale orientée débogage dudit logiciel. Destinée avant tout aux concepteurs de pilote, cette version se démarque essentiellement par les perspectives qu'elle offre sur le plan du suivi et de la validation des composants exécutés en mode noyau. Cela se traduit par un nombre important de mesures, tests, contrôles, et d'autres interventions du même style, qui sont appliqués de façon à améliorer l'identification et la résolution des problèmes de niveau système.

À la différence de son homologue utilisé en environnement de production (build libre), calibré pour les performances et en ce sens allégé de toutes les stratégies susceptibles de les impacter, le build controlé résulte d'un processus de compilation des sources Windows duquel éclos une variété de fonctions de débogage et de trace, y compris la génération de rapports d'erreur, la vérification des résultats d'une opération, la détection des erreurs de logique, ou encore la validation des paramètres transmis d'une routine à une autre. Parallèlement à cela, bon nombre d'optimisations de compilation sont dans le contexte du build controlé à l'état de sommeil, cela afin de faciliter l'étude du code machine. Il n'y a pas, par exemple, post-traitement des binaires à des fins d'exécution plus rapide.

Par nature, le build contrôlé impose des conditions drastiques et des régulations sévères aux fonctions appelées depuis les composants mode noyau du système. Ainsi, si un pilote fait un appel invalide à une routine système qui contrôle les paramètres (par exemple, lors de la réclamation d'un spinlock à un niveau d'interruption inapproprié), le système détecte qu'une erreur est sur le point d'être commise et empêche la requête d'aboutir. (En d'autres circonstances, l'opération aurait certainement été menée à terme, avec tous les risque d'effondrement ultérieur que cela implique. Même si ce point est de plus est de plus en plus discutable, Windows, dans sa configuration initiale, tend à faire aveuglément confiance à n'importe quel élément faisant partie de l'espace système.) Notez que la plupart des tests présents dans la version contrôlée afin de

s'assurer de la justesse des valeurs alimentant tel algorithme ou telle structure de données reposent sur des scénarios éventuels. Il reste par conséquent envisageable qu'un paramètre se situe hors de la plage des valeurs considérées.

La déclinaison contrôlée de Windows emploie plusieurs méthodes de remontée des informations, parmi lesquelles la macro assert, les points d'arrêt et les messages de débogage. Toutes font intervenir la sortie de débogage afin de communiquer les résultat de leurs actions.

Une bonne partie des contrôles effectués dans le build controlé le sont par le biais de la macro assert, laquelle est communément utilisée afin de vérifier les hypothèses formulées depuis le code source d'un programme. Cette macro teste une condition (par exemple la validité d'un paramètre) ; si l'évaluation de l'expression retourne FALSE, cela donne lieu à l'appel de la fonction mode noyau RtlAssert, laquelle appelle DbgPrint pour envoyer le texte d'un message de débogage vers un tampon interne prévu à cet effet. Pour visualiser ces messages, vous pouvez soit attacher un débogueur noyau au système cible, soit employer la commande !dbgprint pendant une session de débogage local, ou encore utiliser l'utilitaire DbgView. La manière dont l'échec d'une assertion affecte le système dépend d'un certain nombre de facteurs. Dans les versions de Windows antérieures Windows Vista, si le système n'a pas été amorcé avec les options de débogage noyau et qu'aucun débogueur noyau n'est attaché, cela entraine automatiquement un effondrement du système (KMODE_EXCEPTION_NOT_HANDLED). Dans Windows Vista et versions ultérieures, si le système n'a pas été amorcé avec le débogueur noyau ou s'il n'y a pas de débogueur noyau actuellement attaché, l'échec d'assertion n'est pas signalé.

La plupart des conditions de point d'arrêt établies dans le build contrôlé le sont de telle manière à fournir un maximum d'informations sur les raisons pour lesquelles elles ont été rencontrées. Cela inclut des renseignements sur le problème qui est survenu, l'erreur qui s'ensuit, ainsi que sur la démarche ayant motivé l'interruption. Dans les rares cas où un point d'arrêt a été atteint sans qu'aucun diagnostic ne l'ai accompagné, un bref examen de la pile noyau (en utilisant par exemple la commande kb du débogueur) suffit en général à restituer avec exactitude les circonstances qui ont menés à cette situation.

Il n'est pas indispensable d'exécuter une installation complète du build controlé pour tirer parti de la version de débogage du système d'exploitation. Afin de nuancer les effets négatifs inhérents aux mesures opérationnelles intégrées à ce type de système (à savoir une forte empreinte sur les ressources machine et l'espace disque), Microsoft tient à jour une autre version parallèle, appelée *build controlé partiel*, dont la particularité est de ne comprendre que les versions contrôlées de l'image noyau et de la couche d'abstraction matérielle, tout le reste provenant la version commercialisée de Windows. Dans la pratique, une telle approche tend à combiner les avantages des deux environnements : rapide d'un côté, techniquement informatif de l'autre.

Le manque d'optimisation des binaires du build controlé, plus le fait qu'ils soit assujettis à des inspections minutieuses, rendent le système particulièrement moins véloce que sa contrepartie arrivant aux mains du grand public. Un tel phénomène peut ainsi dissimuler, ou au contraire mettre en évidence, des soucis de synchronisation multithread, fréquemment dus à des conditions de temporisation spécifiques. C'est la raison pour laquelle en effectuant vos tests sur la version contrôlée du système (ou au moins sur les versions contrôlées de l'image noyau et de la couche d'abstraction matérielle idoine), vous pourriez être surpris par des bogues n'apparaissant pas dans la version commercialisée. (S'il n'existe pas de barrage technique à faire exister et évoluer dans le build libre des contrôles aussi stricts que dans le build controlé, cela n'est pas réaliste du point de vue des performances.)

Vérification de la version (build) exécutée

La commande vertarget des débogueurs Windows standards permet de voir, entres autres informations, laquelle de la version de débogage ou de la version commercialisée est en cours d'exécution sur le système cible.

```
lkd> vertarget
Windows 8 Kernel Version 9200 MP (1 procs) Free x64
```

La propriété Debug de la classe WMI Win32_OperatingSystem indique TRUE s'il s'agit de la version contrôlée du système qui est exécutée, FALSE autrement.

Les informations présentées au niveau l'événement qui porte l'ID 6009, laquelle référence dénote que l'enregistrement a été créé au démarrage du système, incluent le type de l'image noyau qui a été chargée (mono processeur ou multi processeur, libre ou contrôlée).

DebugView

Compagnon d'aide idéal en matière de traque et de résolution des problèmes (*troubleshooting*) liés à la programmation, l'utilitaire DebugView permet de suivre en temps réel les informations de déboggage générées par les divers composants de Windows, incluant programmes en mode utilisateur et pilotes en mode noyau.

Flux de sortie de déboggage

Passerelles parmi d'autres entre le code machine et le développeur attaché à l'étude de celui-ci, Windows établit un certain nombre de flux au travers desquels les programmes peuvent faire entrer ou sortir des informations. L'un, consacré surtout à l'observation du comportement du système (pour voir s'il est conforme au comportement attendu), est le flux de sortie de débogage (debug output), lequel permet aux programmes d'émettre des messages d'erreurs et des diagnostics. Windows implémente ce canal sous forme d'un tampon interne, dont l'utilisation relie une source, un thread, à un destinataire, la plupart du temps un débogueur, mais il peut s'agir de tout autre outil externe avec suffisamment de contrôle pour afficher l'état et l'environnement de débogage.

Les programmes en mode utilisateur peuvent alimenter en informations la sortie de débogage au moyen de la fonction Windows OutputDebugString, laquelle relaie le contenu du message transmis comme paramètre vers le tampon interne. Pour les applications gérées, le framework .NET fournit les classes System.Diagnostics.Debug et System.Diagnostics.Debug, dont les méthodes passent la main à OutputDebugString. Notez que ces méthodes peuvent être sollicitées également depuis Windows PowerShell, par exemple :

```
[System.Diagnostics.Debug]::Print("Some debug output")
```

Les pilotes peuvent émettre des informations apparentées à la sortie de débogage en faisant appel aux routines DbgPrint ou DbgPrintEx. Les pilotes fonctionnant uniquement en mode noyau ont aussi à leur disposition un jeu de fonctions spécialement prévu pour les configurations de déboggage, constitué des macros KdPrint et KdPrintEx, qui sont sans effet dans un environnement de production.

Bien que Windows fournisse à la fois une version 8 bits (ANSI) et 16 bits (Unicode) de l'API OutputDebugString, respectivement OutputDebugStringA et OutputDebugStringW, toute sortie de débogage est traitée en interne selon le point de vue ANSI. En la matière, et en contrepied total du scénario classique des fonctions à double entrée, lorsqu'une application sollicite la version large de OutputDebugString (OutputDebugStringW), les chaînes en entrée sont converties en ANSI avant d'être traités par le système. Notez que cette conversion se faisant sur la base de paramètres régionaux, certains caractères peuvent ne pas s'afficher correctement.

Coup d'oeil sur l'interface DebugView

Comme la plupart des outils Sysinternals, DebugView ne requiert pas de processus d'installation, se rendant disponible pour l'exécution dès la fin du téléchargement. Vous pouvez également déclencher l'utilitaire directement à partir du site Web sans le télécharger. Le programme une fois lancé, DebugView démarre automatiquement la collecte des données significatives.

Ainsi que vous pouvez le voir au niveau de la figure suivante, les informations présentées par DebugView le sont sous forme de colonnes.

Interface principale de DebugView . image::debugview.png[effe, 400, scaledwidth="60%"]()

La première colonne affiche une liste de valeurs à partir de laquelle identifier un enregistrement de débogage parmi les autres. Notez que le schéma sur lequel DebugView se base pour attribuer à chaque enregistrement

un numéro de séquence est interne à l'application, et ne fait donc sens qu'au sein de celle-ci. Les valeurs sont générées dans un ordre croissant, même si des écarts entre certaines peuvent apparaitre, du fait des conditions en place, règles de filtrage notamment.

La seconde colonne correspond à un point donné du traitement des messages de débogage et affiche le moment où chacun a été enregistré, soit en temps écoulé ou en temps réel. Par défaut, DegugView indique le nombre de secondes écoulées depuis la capture du premier enregistrement, avec l'élément initial positionné toujours sur une valeur nulle (0.00). Cela peut être utile pour résoudre des problèmes liés à des conditions de timing spécifiques, souvent la cause de bogues de synchronisation multithread. Ce compteur est remis à zéro chaque fois que l'affichage est actualisé. Si vos préférences se tournent vers un mécanisme d'horodatage, avec association d'une date et d'une heure à un événement, activez l'option Clock Time à partir du menu Options. Choisissez Show Milliseconds, toujours dans le menu Options, si vous souhaitez que l'horodatage tienne compte des millisecondes. Vous pouvez également configurer l'affichage de l'heure avec des options de ligne de commande : /o pour afficher pour afficher l'heure d'horloge, /om pour l'heure d'horloge avec millisecondes, et / on pour le temps écoulé.

La troisième colonne affiche la collection des sorties de débogage pour les composants cibles : programmes mode utilisateur, programmes mode noyau, éventuellement les deux. Lorsqu'un enregistrement concerne le flux de communication mode utilisateur, l'identifiant du processus (PID) duquel est issu l'opération de débogage apparait entre crochets, suivie de la sortie elle-même. Dans le cas où vous ne souhaiterez pas voir les données d'identification relatives aux processus, désactivez l'option Win32 PIDs à partir du menu Options. Par défaut, DebugView affiche le texte de chaque message passée à une fonction de sortie de débogage sur une ligne distincte, que ce texte se termine ou non par un retour chariot. Si vous désactivez l'option Force Carriage Returns du menu Options, DebugView laisse se déverser les divers enregistrements dans un tampon interne, pour ne l'afficher à l'écran que lorsqu'un retour chariot est rencontré, ou si la mémoire allouée pour ce tampon atteint une taille limite. Remarquez que cette méthode de présenter les données, avec laquelle applications et pilotes peuvent générer plusieurs lignes en sortie, ne convient pas en toutes circonstances. Elle exige en tout cas de l'analyste une certaine discipline, puisque dans cette configuration, si plusieurs processus génèrent des informations de sortie, les données se mélangent alors de manière confuse. De plus, le PID apparaissant à ce moment dans l'application identifie non le processus source du message, mais le processus ayant émis le texte comportant un retour chariot ou qui a rempli la mémoire tampon.

Les résultats des opérations de débogage sont ajoutés en fin de la liste au fur et à mesure que celles-ci se produisent sur le système. La fonction de défilement automatique de DebugView, faisant en sorte que les informations les plus récentes soient aussi celles immédiatement visibles, permet de naviguer facilement entre les diverses entrées. Pour basculer en mode défilement automatique, faites Ctrl+A ou sélectionnez l'icone Autoscroll dans la barre d'outils.

Pour finir, sachez si vous le désirez que DebugView peut être affiché en avant-plan de toutes les autres fenêtres d'application, de façon à toujours avoir un oeil sur les résultats. Pour ce faire, sélectionnez Always on Top à partir du menu Options.

Enregistrement du flux de sortie mode utilisateur

DebugView peut enregistrer l'actualité des diagnostics générés depuis plusieurs sources de données : la session de travail de l'utilisateur interactif, la session de console physique sur la machine (également appelée session 0), et le mode noyau. Chacun de ces foyers d'informations peut être sélectionné à partir du menu Capture. Une fois le mode de fonctionnement choisi, la capture peut être démarrée via l'option Capture Events du menu Capture, via les touches de raccourci Ctrl+E ou en cliquant sur l'icône Capture de la barre d'outils. Lorsqu'elle est active, la sortie de débogage est capturée à partir des sources sélectionnées.

Par défaut, DebugView collecte seulement les activités de débogage dans la session courante parmi les multiples sessions interactives possibles sur une même machine. Une session se compose des processus et des autres objets système (station fenêtre, bureaux et fenêtres) représentant la session de travail d'un utilisateur. L'option Capture Win32 du menu Capture et de la barre d'outils (touches de raccourci Ctrl+W) permet de voir les messages envoyés par les programmes mode utilisateur sur la sortie de débogage (fonction Windows

OutputDebugString). Dans ce mode, des informations complémentaires relatives à l'identification du processus émetteur peuvent être demandées en sélectionnant l'option Win32 PIDs du menu Options.

En plus des fonctionnalités citées plus haut, en lien avec la session où un utilisateur exécute ses applications, DebugView permet également le suivi des opérations de débogage effectuées depuis la session globale, appelée aussi session 0, dans laquelle s'exécutent les services et où les objets de niveau globaux sont définis. L'élément de menu Capture Global Win32 fournit un tremplin vers l'activation/désactivation de l'enregistrement des données générés dans la session console. Notez que ce type d'action requérant de l'application l'élévation au statut d'administrateur, vous devez en conséquence disposer de droits d'accès appropriés pour utiliser cette option.

Capture du flux de sortie mode noyau

Au contraire de la plupart des outils du même style, lesquels ne permettent d'envisager que des scénarios de niveau applicatif mode utilisateur, DebugView permet en plus de surveiller le flux de sortie d'erreur apparenté au code mode noyau du système. Vous pouvez configurer DebugView pour afficher les résultats du débogage mode noyau en activant l'option Capture Kernel à partir de menu Options. Notez que cet usage requiert des droits d'administration, en particulier le privilège Load Driver.

Les composants s'exécutant dans l'espace noyau peuvent définir le niveau de gravité de chaque message qu'ils transmettent. Si vous souhaitez voir l'intégralité des sorties de débogage, choisissez l'option Enable Verbose Kernel Output dans le menu Capture. Quand cette option n'est pas activée, DebugView affiche seulement les messages dont le niveau de sévérité les classe comme erreurs.

S'il les en prive habituellement (mode par défaut), DebugView peut être configuré pour transmettre des résultats de débogage mode noyau à d'autres débogueurs. Pour activer ce mode, dit pass-through, activez l'option pass-through mode à partir du menu Capture ou cliquez sur l'icône Pass-Through dans la barre d'outils. Le mode pass-through vous permet de visualiser le contenu des informations de débogage dans un débogueur classique en meme temps que dans DebugView.

Contrôle à distance

Outre des procédures d'écoute relatives à la station de travail locale, DebugView dispose de fonctionnalités à distance qui le mettent en lien potentiel avec n'importe quelle machine du réseau. En ce sens, caractérisant une solution de visualisation intégrée et complète, l'utilitaire peut servir pour l'exploration de données en provenance de multiples sources, l'appareil local aussi bien que des ordinateurs distants.

Lorsqu'il est employé à des fins d'analyse dans un contexte autre que l'ordinateur local, DebugView s'exécute dans un mode de fonctionnement spécial, dit mode agent (agent mode), grâce auquel les enregistrements effectués par une instance de l'application sont relayés vers une instance centrale, cette dernière se chargeant seule de la présentation des données. Vous pouvez choisir de démarrer DebugView sur le système distant de manière manuelle ou automatique, moyennant dans le second cas la mise en conformité avec les normes et règles de sécurité de votre réseau.

Appuyez sur Ctrl-R ou choisissez l'option Connect du menu Computer pour ouvrir une boîte de dialogue via laquelle accéder à un autre ordinateur en réseau. Entrez le nom ou l'adresse IP de l'ordinateur distant, ou cliquez sur le bouton en regard de la zone de liste pour ouvrir la boîte de dialogue Rechercher un ordinateur. Dans la boîte de dialogue Rechercher un ordinateur, à l'aide du contrôle d'arborescence, recherchez l'ordinateur souhaité, puis cliquez sur OK. DebugView essaiera à ce moment d'installer et de démarrer un agent sur cet appareil ; s'il ne le peut pas, DebugView tente de se connecter à un agent déjà en cours d'exécution - cela supposant qu'un ai été démarré manuellement au préalable. Si une connexion est établie, DebugView affiche les données reçues depuis ce dispositif distant, en ajoutant le nom de l'ordinateur à la barre des titres et au menu Computer.

DebugView peut se connecter simultanément à un nombre quelconque de sources de données. Vous pouvez changer la visualisation de sorte à voir la sortie de débogage d'un ordinateur en le sélectionnant parmi la liste

proposée par le menu Computer, ainsi que le montre la figure x. Vous pouvez également basculer entre plusieurs ordinateurs en appuyant sur Ctrl+Tab. L'ordinateur actif est signalé par l'affichage de son nom dans la barre de titre de l'application, et par une icône en forme de flèche dans le menu Computer. Notez qu'il est aussi possible d'ouvrir chaque ordinateur dans une fenêtre séparée.

Enregistrement et journaux

DebugView implémente pour la conservation (sauvegarde) des données sources une méthodologie à deux voies : l'enregistrement à la demande, et l'enregistrement séquentiel (journalisation). Dans les deux cas, les informations de débogage ainsi enregistrées le sont en tant que fichier texte (extension .log), ce qui signifie qu'elles peuvent être partagées, archivées, ou simplement conservées pour utilisation ultérieure.

Vous pouvez enregistrer le contenu de la fenêtre principale de DebugView en choisissant Enregistrer ou Enregistrer sous à partir du menu Fichier. Pour visualiser le fichier résultant, sélectionnez l'option Ouvrir du menu Fichier, ou spécifiez le chemin d'accès au fichier sur la ligne de commande.

Pour enregistrer les informations de débogage au fur et à mesure qu'elles apparaissent, choisissez au niveau du contrôle Fichier l'entrée de menu Log To File, ou cliquez sur le bouton de la barre d'outils. La première fois que vous faites cela après le démarrage de DebugView, une fenêtre de réglages apparaît, menant aux divers paramètres susceptibles d'être introduits ou modifiés. Ces derniers sont :

- *Emplacement du journal* Désigne via le chemin d'accès un fichier à utiliser pour la sauvegarde. Lors du déploiement de contenus multiples, la partie de ce chemin menant au répertoire indique le dossier où iront se loger les fichiers résultants.
- *Taille de journal illimitée* Permet au fichier de croitre sans considération de volume sur le disque, entendu que les restrictions à cet égard au niveau du système de fichiers restent évidemment en vigueur.
- *Créer un nouveau journal pour chaque jour* Demande à DebugView de créer chaque jour un nouveau fichier pour le stockage du journal, avec la date du jour ajoutée au nom du fichier. Vous pouvez également sélectionner l'option pour effacer l'écran lorsque le journal pour une nouvelle journée est créé.
- *Limite de taille du journal* Quand cette option est choisie, le fichier journal est contrôlée par une taille limite jusqu'à laquelle il peut s'étendre. Du moment que cette valeur est atteinte, deux situations sont envisageables, selon que l'option Wrap soit ou non de mise. Avec l'option Wrap, les données reçus après atteinte de la taille maximale sont renvoyés au début du fichier journal. Autrement, l'enregistrement s'arrête.

Du moment que des choix ont été faits pour un journal, l'option de menu Log To File, ainsi que le bouton caractéristique dans la barre d'outils, sert de relais pour l'activation et la désactivation du rôle enregistrement. Pour établir un fichier différent ou modifier d'autres paramètres, choisissez l'option Log To File As à partir du menu File.

Lorsque plusieurs ordinateurs font objet d'une surveillance et que la journalisation dans un fichier est en vigueur, l'ensemble des informations est consigné à l'intérieur d'un seul et unique fichier. L'empreinte de chaque système est néanmoins facilement distinguable, avec comme moyen de repère un entête donnant le nom de l'ordinateur à partir duquel plusieurs résultats ont été ont été enregistrées.

Options de filtrage

Une autre méthodologie d'accès à l'information offerte par DebugView consiste à utiliser ses capacités en matière de filtre, lesquelles donnent à l'utilisateur les moyens de se repérer parmi une collection de données en choisissant un ou plusieurs critères.

Naviguez dans le menu Edit jusque l'élément Filter/Highlight, ou passez par le bouton appropriée de la barre d'outils, ou encore activez le raccourci clavier Ctrl+L, pour afficher la boite de dialogue consacrée aux règles de filtre et de mises en évidence.

Les règles de filtre permettent de vous concentrer sur un sous-ensemble de données spécifique en fonction de certains critères. A cela correspond dans DebugView les contrôles d'édition Include et Exclude, servant à spécifier les termes sur la base desquels afficher, ou au contraire ne pas montrer, les enregistrements remplissant les conditions spécifiées. Si plusieurs termes méritent une attention particulière, vous pouvez faire la liste de ces derniers en les séparant par un point-virgule. N'utilisez de caractères d'espacement que si vous souhaitez les voir faire partie du filtre. Notez que le caractère étoile est interprété comme représentant n'importe quelle suite, ou absence, de caractère, et que les règles de filtre sont insensibles à la casse. Les règles par défaut, puisqu'elles n'en exclue aucun, assurent à tout flux de données une visibilité au sein de l'application.

Remarquez que les filtres s'appliquent seulement aux données nouvelles (c'est à dire, les enregistrements effectuées après qu'un filtre ai été mis en place) et aux commentaires, ajoutées à l'aide de la fonction Append Comment. Les nouveaux enregistrements qui correspondent aux règles en vigueur sont affichés; ceux qui n'y correspondent pas sont supprimées et ne peuvent pas être affichés a posteriori, par exemple via un changement des règles. En outre, la modification des règles de filtre n'a pas d'incidence sur les informations déjà affichés par DebugView.

Les règles de mises en évidence, plutôt que de jouer comme les filtres sur la visibilité de l'information, mettent en place un protocole qui, avec la technique de présentation associé, marque l'importance d'une ou plusieurs données. Comme pour le filtrage, la mise en valeur suppose la spécification de critères. Les enregistrements qui correspondent aux conditions sont mis en surbrillance , contrairement aux enregistrements qui ne les remplissent pas, bien que ceux-ci restent visibles.

DebugView conserve les options de filtre et de tri appliquées au moment de la sortie de programme. La prochaine fois que vous démarrez DebugView, une boite de dialogue apparait comme première fenêtre ouverte à l'écran, vous demandant quoi faire de ces réglages. Vous pouvez choisir soit de les modifier, soit d'utiliser ceux en vigueur, ici en cliquant sur le bouton OK. Utilisez sur le bouton Load pour charger une configuration précédemment enregistrée, ou sur le bouton Reset pour enlever le filtre. Pour contourner cette boite de dialogue, ajoutez le commutateur /f à la ligne de commande.

Handle

L'utilitaire Handle recense les descripteurs ouverts par les processus sur un système. Cela inclut les descripteurs de fichiers et de répertoires, mais également les ports de communication, les clés du Registre, les threads, et d'autres encore. Tous les objets visibles de l'API Windows sont concernés.

Handle dispose de deux modes d'échantillonnage, à quoi correspond dans un cas une utilisation du logiciel à des fins de consultation (visualisation de tous les descripteurs ouverts), et, second cas, pour un usage à des fins de recherche. Par défaut, la commande Handle liste les valeurs de tous les descripteurs associés à des fichiers, et les noms de ceux-ci.

Quand il n'est pas utilisé pour une recherche avec des critères précis, Handle organise les donnes en autant de sections logiques que de processus pour lesquels il montre les informations. Une ligne pointillée sert de séparateur, après laquelle se trouvent le nom et l'ID du processus. Sous ce nom est présenté la valeur du descripteur (en hexadécimal), le type de l'objet auquel le descripteur est associé et, s'il en a un, le nom de l'objet.

Tasklist

L'utilitaire intégré tasklist affiche une liste des processus en cours sur un ordinateur local ou distant.

```
C:\>tasklist

Nom de l'image                 PID Nom de la sessio Numéro de s Utilisation
========================= ======== ================ =========== ============
System Idle Process              0 Services                   0          8 Ko
System                           4 Services                   0      2 856 Ko
Registry                       128 Services                   0     22 020 Ko
```

```
smss.exe                     508 Services                   0     1 200 Ko
csrss.exe                    880 Services                   0     5 904 Ko
...
```

La syntaxe associée se présente comme suit :

```
TASKLIST [/S système [/U utilisateur [/P [mot_de_passe]]]]
         [/M [module] | /SVC | /V] [/FI filtre] [/FO format] [/NH]
```

- **/FI filtre** Affiche un ensemble de tâches qui répond aux critères données.
- **/FO format** Contrôle la mise en forme de la sortie. Les valeurs valides sont TABLE (présentation sous forme de tableau), CSV (valeurs séparées par des virgules), et LIST (liste).
- **/M [module]()** Affiche les modules associés au nom d'exécutable ou de bibliothèque spécifié. En l'absence de nom, tous les modules chargés sont répertoriés.
- **/P mot_de_passe** Spécifie le mot de passe pour le contexte utilisateur donné.
- **/S système** Spécifie le système distant auquel se connecter. Les commutateurs /u et /p sont alors employés pour transmettre les informations d'authentification requises dans ce contexte.
- **/SVC** Affiche les services hébergés dans chaque processus.
- **/U utilisateur** Spécifie le contexte utilisateur sous lequel la commande doit exécuter.
- **/V** Affiche des informations de tâches détaillées. En plus des informations de base, l'emploi d'une telle option montre l'état du processus (Status), l'utilisateur pour le compte duquel il s'exécute (Nom d'utilisateur), la quantité totale de temps processeur utilisé depuis son démarrage (Temps processeur), et le titre de la fenêtre à laquelle il se rattache (si le processus concerné dispose d'une interface graphique).

Une particularité de TaskList, par rapport à d'autres logiciels du même type, tient à la possibilité de réduire sa sortie à des paramètres spécifiques en utilisant des filtres.

Tableau 2.9. Filtres utilisés dans la commande Tasklist

Nom du filtre	Opérateurs valides	Valeurs autorisées
CPUTIME	eq, ne, gt, lt, ge, le	Heure au format valide
IMAGENAME	eq, ne	Nom d'image
MODULES	eq, ne	Nom de DLL
MEMUSAGE	eq, ne, gt, lt, ge, le	Mémoire utilisée en kilo-octets
PID	eq, ne, gt, lt, ge, le	Valeur PID (entier positif)
SESSION	eq, ne, gt, lt, ge, le	Numéro de session
SESSIONNAME	eq, ne	Nom de session
STATUS	eq, ne	RUNNING, NOT RESPONDING, UNKNOWN
USERNAME	eq, ne	Nom de compte
SERVICES	eq, ne	Nom de service
WINDOWTITLE	eq, ne	Titre de la fen^tre

Tableau 2.10. Opérateurs de comparaison pour les filtres

Opérateur	Signification
eq	Égal
ne	Différent de
gt	Plus grand que
lt	Plus petit que
ge	Supérieur ou égal à
le	Inférieur ou égal à

Voici un exemple illustrant l'utilisation des filtres conjuguée à une commande visant à identifier les processus qui ne répondent pas.

```
tasklist /fi "status eq not responding"
```

WinDbg

Windows Debugger, ou WinDbg, est un puissant outil de débogage développé par Microsoft pour les systèmes d'exploitation Microsoft Windows. Il permet d'analyser aussi bien les composants de l'espace utilisateur que ceux de l'espace système, offrant ainsi une solution complète pour le débogage d'applications, de services, de pilotes, ou même du système d'exploitation.

Doté d'une interface graphique avancée, WinDbg offre des fonctionnalités variées : visualisation du code source, mise en place de points d'arrêt, inspection de variables, examen de l'état d'un processus, et bien d'autres.

En plus des fonctionnalités citées précédemment - sommes toutes classiques pour un logiciel de ce style -, WinDbg se distingue également par sa capacité à effectuer des analyses post-mortem, utiles pour examiner des instantanés de l'état d'un système ou d'un processus après une défaillance. Il permet de la sorte d'explorer des structures de contrôle sans interférer avec l'exécution du système.

Enfin, WinDbg propose un système efficace de gestion des symboles de débogage. Il peut charger automatiquement les fichiers PDB depuis un serveur public ou privé, en identifiant la version correcte en fonction des caractéristiques du fichier (CRC, architecture, date, etc.). Cette fonctionnalité élimine les difficultés liées à la coexistence de plusieurs versions binaires sur une machine cible, tout en facilitant la corrélation avec le code source original. Microsoft fournit un serveur public regroupant les symboles de la majorité de ses systèmes d'exploitation.

Directives de WinDbg

En plus des commandes d'extension, qui se rapportent à l'application en cours d'examen, WinDbg admet un certain nombre de *directives*, exerçant quant à elles leur influence sur la session de débogage dans son ensemble. La liste qui suit passe en revue une partie des directives disponibles. (Notez que toutes sont préfixées par un point, tandis que les commandes d'extension le sont par un point d'exclamation.)

- La commande *.load* charge une DLL d'extension du débogueur.
- La commande *.unload* décharge une DLL d'extension du débogueur.
- La commande *.reload* force l'actualisation des symboles déjà rencontrés lors de la session.
- La commande *.logopen* ouvre un fichier journal et enregistre dans ce dernier toutes les activités ayant lieu par la suite.
- La commande *.logclose* ferme le fichier journal.
- La commande *.kill* met fin à la session de débogage actuelle.
- La commande *.chain* répertorie les extensions de débogueur en cours de disponibilité.

```
0:000> .chain
Extension DLL search Path:
    ...
Extension DLL chain:
    DbgEngCoreDMExt: image 10.0.25921.1001, API 0.0.0,
        [path: C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\winext
\DbgEngCoreDMExt.dll]
    dbghelp: image 10.0.25921.1001, API 10.0.6,
        [path: C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\dbghelp.dll]
```

```
    exts: image 10.0.25921.1001, API 1.0.0,
        [path: C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\WINXP\exts.dll]
    uext: image 10.0.25921.1001, API 1.0.0,
        [path: C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\winext\uext.dll]
    ntsdexts: image 10.0.25921.1001, API 1.0.0,
        [path: C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\WINXP\ntsdexts.dll]
```

- La commande *.formats* affiche une valeur donnée dans plusieurs formats numériques différents, tels que hexadécimal, décimal, octal, binaire, et même temporel.

```
0:007> .formats 0xc0ffee
Evaluate expression:
  Hex:     00000000`00c0ffee
  Decimal: 12648430
  Decimal (unsigned) : 12648430
  Octal:   0000000000000060177756
  Binary:  00000000 00000000 00000000 00000000 00000000 11000000 11111111 11101110
  Chars:   ........
  Time:    Wed May 27 05:27:10 1970
  Float:   low 1.77242e-038 high 0
  Double:  6.24915e-317
```

Débogage noyau local

Au-delà des fonctionnalités attendues de ce genre de logiciel, WinDbg offre la possibilité d'ausculter l'état du système sur le même ordinateur que celui où le débogueur est exécuté. On parle en la circonstance de *débogage noyau local*.

Pour initier le débogage noyau local, assurez-vous d'abord que système soit en mesure de satisfaire une telle opération - à l'aide, par exemple, du commutateur debug dans bcdedit. Démarrez WinDbg (des privilèges d'administration sont requis), ouvrez le menu Fichier, choisissez Attach to Kernel, cliquez sur l'onglet Local puis sur OK. Vous devriez à ce moment voir quelque chose ressemblant à ce qui suit.

```
Microsoft (R) Windows Debugger Version 10.0.25921.1001 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

Connected to Windows 10 22000 x64 target at (Mon Feb 26 11:34:44.777 2024 (UTC - 8:00)), ptr64 TRUE

************* Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       srv*
Symbol search path is: srv*
Executable search path is:
Windows 10 Kernel Version 22000 MP (2 procs) Free x64
Product: WinNt, suite: TerminalServer SingleUserTS
Edition build lab: 22000.1.amd64fre.co_release.210604-1628
Kernel base = 0xfffff800`54400000 PsLoadedModuleList = 0xfffff800`55029710
Debug session time: Mon Feb 26 11:34:45.495 2024 (UTC - 8:00)
System Uptime: 6 days 0:52:11.354
```

Compte tenu des circonstances, certaines commandes du débogueur noyau ne fonctionnent pas en mode local, par exemple la définition de points d'arrêt (ce qui de toute manière n'aurait pas grand sens vu le contexte), la visualisation des registres processeur, ou encore la création de fichier de vidange sur incident. Un autre inconvénient à considérer en la matière se rapporte aux perpétuelles évolutions que traverse le système en cours d'examen (les registres, par exemple, changent constamment), ce qui en pratique est susceptible quelquefois de complexifier (voire totalement entraver) la procédure.

Commandes de manipulation des threads

- La commande ~ sans paramètres affiche la liste des threads. L'identifiant du thread, son état, et l'adresse du bloc d'environnement (TEB) qui s'y rapporte font partie des informations proposées.

```
0:000> ~
.  0  Id: 5b6c.5a58 Suspend: 1 Teb: 0000004d`7be4b000 Unfrozen
   1  Id: 5b6c.15f0 Suspend: 1 Teb: 0000004d`7be4d000 Unfrozen
   2  Id: 5b6c.1258 Suspend: 1 Teb: 0000004d`7be4f000 Unfrozen
   3  Id: 5b6c.55a4 Suspend: 1 Teb: 0000004d`7be51000 Unfrozen
```

- La commande `~n` affiche les informations concernant le thread spécifié, *n* étant dans ce contexte le numéro du thread. La priorité du thread, sa classe de priorité, ainsi que l'adresse de la fonction qu'il exécute, sont présentées.

```
0:000> ~0
.  0  Id: 5b6c.5a58 Suspend: 1 Teb: 0000004d`7be4b000 Unfrozen
      Start: notepad!wWinMainCRTStartup (00007ff6`1a5419a0)
      Priority: 0  Priority class: 32  Affinity: 3f
```

- La commande `~ns` modifie le contexte actuel pour le remplacer par celui du thread spécifié. L'invite de WinDbg est mise à jour de sorte à refléter la prise en compte de l'opération.

```
0:000> ~2s
ntdll!NtWaitForWorkViaWorkerFactory+0x14:
00007ffd`cee32fc4 c3              ret
0:002>
```

Commandes de registres

La commande *r* sans paramètres affiche les registres du thread en cours d'examen chargés de stocker des valeurs entières.

```
0:000> r
rax=0000000000000000 rbx=00007ffe27baa860 rcx=00007ffe27b0f814
rdx=0000000000000000 rsi=000000068798b000 rdi=00007ffe27ba6c08
rip=00007ffe27b4b784 rsp=000000068779f0c0 rbp=0000000000000000
 r8=000000068779f0b8  r9=0000000000000000 r10=0000000000000000
r11=0000000000000246 r12=0000000000000040 r13=0000000000000001
r14=0000014c038b0000 r15=0000000000000000
iopl=0         nv up ei pl zr na po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
```

Commandes d'affichage de la mémoire

- La commande *dw* (display word) affiche les données par groupes de 2 octets.

```
0:000> dw 0x7ffe0000
00000000`7ffe0000  0000 0000 0000 0fa0 e574 722c 075e 0000
00000000`7ffe0010  075e 0000 6b24 d72e 7400 01da 7400 01da
00000000`7ffe0020  a000 8711 0021 0000 0021 0000 8664 8664
00000000`7ffe0030  0043 003a 005c 0057 0049 004e 0044 004f
00000000`7ffe0040  0057 0053 0000 0000 0000 0000 0000 0000
00000000`7ffe0050  0000 0000 0000 0000 0000 0000 0000 0000
00000000`7ffe0060  0000 0000 0000 0000 0000 0000 0000 0000
00000000`7ffe0070  0000 0000 0000 0000 0000 0000 0000 0000
```

- La commande *dd* (display double-word) affiche les données par groupes de 4 octets.

```
0:000> dd 0x7ffe0000
00000000`7ffe0000  00000000 0fa00000 a486b84b 0000075f
00000000`7ffe0010  0000075f 0988f4af 01da7402 01da7402
00000000`7ffe0020  8711a000 00000021 00000021 86648664
00000000`7ffe0030  003a0043 0057005c 004e0049 004f0044
00000000`7ffe0040  00530057 00000000 00000000 00000000
```

```
00000000`7ffe0050  00000000 00000000 00000000 00000000
00000000`7ffe0060  00000000 00000000 00000000 00000000
00000000`7ffe0070  00000000 00000000 00000000 00000000
```

- La commande *dq* (display quad-word) affiche les données par groupes de 8 octets.

```
0:000> dq 0x7ffe0000
00000000`7ffe0000  0fa00000`00000000 0000075e`7e58e56c
00000000`7ffe0010  e35a725f`0000075e 01da7400`01da7400
00000000`7ffe0020  00000021`8711a000 86648664`00000021
00000000`7ffe0030  0057005c`003a0043 004f0044`004e0049
00000000`7ffe0040  00000000`00530057 00000000`00000000
00000000`7ffe0050  00000000`00000000 00000000`00000000
00000000`7ffe0060  00000000`00000000 00000000`00000000
00000000`7ffe0070  00000000`00000000 00000000`00000000
```

- La commande `da` affiche l'interprétation Ascii de la mémoire.

```
0:000> da 0x7ff7d4cf0000
00007ff7`d4cf0000  "MZ."
```

- La commande `du` affiche l'interprétation Unicode de la mémoire.

```
0:000> du 0x7ffe0030
00000000`7ffe0030  "C:\WINDOWS"
```

Commandes de niveau image

- La commande *!dlls* affiche une liste des modules chargés (essentiellement les DLL), y compris des informations telles que leur nom, adresse de départ, taille, etc.

```
0:000> !dlls

0x214c8112c40: C:\Windows\System32\notepad.exe
      Base   0x7ff67dfd0000  EntryPoint  0x7ff67dfd19a0  Size        0x0005a000    DdagNode     0x214c8112d80
      Flags  0x0000a2cc  TlsIndex    0x00000000  LoadCount   0xffffffff    NodeRefCount 0x00000000
             <unknown>
             LDRP_LOAD_NOTIFICATIONS_SENT
             LDRP_IMAGE_DLL

0x214c8112aa0: C:\WINDOWS\SYSTEM32\ntdll.dll
      Base   0x7ffe27a70000  EntryPoint  0x00000000  Size        0x00216000    DdagNode     0x214c8112be0
      Flags  0x0000a2c4  TlsIndex    0x00000000  LoadCount   0xffffffff    NodeRefCount 0x00000000
             <unknown>
             LDRP_IMAGE_DLL

0x214c8117940: C:\WINDOWS\System32\KERNEL32.DLL
      Base   0x7ffe25f70000  EntryPoint  0x7ffe25f825e0  Size        0x000c4000    DdagNode     0x214c8117a80
      Flags  0x000ca2cc  TlsIndex    0x00000000  LoadCount   0xffffffff    NodeRefCount 0x00000000
             <unknown>
             LDRP_LOAD_NOTIFICATIONS_SENT
             LDRP_IMAGE_DLL
             LDRP_DONT_CALL_FOR_THREADS
             LDRP_PROCESS_ATTACH_CALLED

0x214c8117f30: C:\WINDOWS\System32\KERNELBASE.dll
      Base   0x7ffe252f0000  EntryPoint  0x7ffe2532cb10  Size        0x003a6000    DdagNode     0x214c8118070
      Flags  0x0008a2cc  TlsIndex    0x00000000  LoadCount   0xffffffff    NodeRefCount 0x00000000
             <unknown>
             LDRP_LOAD_NOTIFICATIONS_SENT
             LDRP_IMAGE_DLL
             LDRP_PROCESS_ATTACH_CALLED

...
```

Utilitaires Sysinternals

Une bonne partie de la matière technique de ce livre provient d'utilitaires gratuits que vous pouvez télécharger depuis www.sysinternals.com (page qui vous redirigera sur http://technet.microsoft.com/sysinternals). Mark Russinovich, co-auteur des livres de la série Inside Windows, écrit la plupart de ces outils, d'abord de façon indépendante puis, Microsoft ayant acquis l'entreprise et ses actifs, conjointement au programme Microsoft TechNet.

Le site Web Sysinternals a été créé en 1996 par Mark Russinovich et Bryce Cogswell pour accueillir leurs logiciels de gestion, dépannage et diagnostic d'applications et de systèmes Windows, servant régulièrement à observer et à comprendre le comportement du système d'exploitation. Très appréciés des professionnels, nous ne pouvions manquer de donner à certains de ces utilitaires une mention spéciale. Les ressources disponibles sont classées en plusieurs grandes catégories :

- *Utilitaires de fichiers et de disques* Utilitaires permettant de consulter et de surveiller l'accès et l'usage des fichiers et disques ; parmi lesquels Diskmon, qui capture toute activité du disque dur, et NTFSInfo, qui permet d'afficher des informations détaillées sur les volumes NTFS.

- *Réseau* Outils réseau allant de moniteurs de connexion à des analyseurs de sécurité des ressources ; parmi lesquels PsFile, pour voir quels sont les fichiers ouverts à distance, et TCPView, qui affichera une liste détaillée de tous les points de terminaison TCP et UDP sur votre système, y compris les adresses locales et distantes et l'état des connexions TCP.

- *Processus et threads* Utilitaires permettant de voir ce que font les processus et les ressources qu'ils consomment ; parmi lesquels Autoruns, indiquant quels programmes sont configurés pour démarrer automatiquement lors du démarrage du système, Process Explorer, dont nous avons déjà parlé, et la suite Tools, qui inclut des utilitaires de lignes de commande pour répertorier les processus exécutés sur des ordinateurs locaux ou distants, exécuter des processus à distance, redémarrer des ordinateurs, vider des journaux d'événements, entre autres.

- *Utilitaires de sécurité* Utilitaires de configuration et de gestion de la sécurité ; parmi lesquels AccessChk, qui permet de visualiser le type d'accès à un objet que possède un utilisateur ou un groupe d'utilisateurs, ou LogonSessions, chargé de répertorier les ouvertures de session actives.

- *Informations système* Utilitaires permettant d'examiner l'utilisation et la configuration des ressources système ; parmi lesquels ClockRes, qui affiche la résolution de l'horloge système, LiveKd, qui s'appuient sur le moteur de débogage de Windows pour examiner un système actif, Winobj, qui permet d'accéder aux informations du Gestionnaire d'objets, et DebugView, qui permet d'afficher en temps réel les opérations de débogage effectuées par les divers processus de votre système.

Configuration et utilisation des symboles

Qu'il s'agisse de déplomber des programmes ou analyser le comportement du système, l'étape de configuration des symboles est cruciale. Utilisés par les débogueurs pour référencer et afficher fonctions et variables, les fichiers de symbole contiennent pléthore de renseignements utiles, et constituent une aide précieuse pour fomenter des hypothèses crédibles sur les mécanismes internes de Windows. Dans cette section, nous discutons de la façon d'utiliser correctement les fichiers de symboles, et de découvrir leur importance pour le débogage.

Généralités concernant les symboles

Les composants de système informatique (applications, librairies, pilotes, systèmes d'exploitation, etc.) doivent pour apparaitre sous forme d'unités compréhensibles du processeur faire l'objet de procédures de compilation et d'éditions des liens. Celles-ci, en plus de construire les modules exécutables correspondants, créent un certain nombre de fichiers supplémentaires, connus collectivement comme les fichiers de symbole. Les fichiers de

symboles contiennent une variété de données qui ne servent pas pour l'exécution du code, mais se révèlent fort pratique dans le processus de débogage. Voici en général ce que ce disent ces fichiers :

- Les noms et adresses des variables globales
- Les noms et adresses des fonctions et leurs signatures

Comme ils ne servent pas pour l'exécution, les symboles sont généralement séparés de l'image binaire de laquelle ils divulguent les informations. Cela rend les binaires plus petits et plus rapides. (Pour ce fait de rapidité, comprendre moins lourd à charger dans la mémoire.) En revanche, cela signifie que le débogueur puisse accéder aux fichiers de symboles associés aux images concernées par la session de débogage.

Pour diverses raisons, des problématiques de performance aux enjeux de la propriété intellectuelle, Microsoft a eu recours a plusieurs formats de symboles, dont COFF (Common Object File Format), CV (CodeView), et PDB (Program Database). Les versions récentes de Windows utilisent le format PDB.

Chemin des symboles

Le débogueur accède aux fichiers de symboles associés aux images concernées par la session de débogage en cours. Aussi, pour connaitre quels fichiers sont utiles, et lesquels ne le sont, le débogueur utilise deux types d'information : l'emplacement du chemin des symboles, représenté comme un ensemble de chemins d'accès, combiné avec les informations stockées dans les entêtes du module qui ont servi à valider les fichiers de symbole. Chaque chemin d'accès peut être représenté sous la forme d'un dossier local, un partage UNC, ou un serveur de symboles à la demande, comme décrit dans la section Serveur de symboles. Dans sa forme élémentaire, un chemin de symboles est une succession de chemin d'accès délimités par le caractère point-virgule (;). La commande. sympath (Set Symbol Path) modifie le chemin par défaut du débogueur hôte pour la recherche de symbole.

```
.sympath exe c:\sympath1 ; c:\sympath2
```

Outre les commandes de débogueur relatives au chemin des symboles, dont .sympath est la plus essentielle, le débogueur peut également être renseigné du même chemin d'accès via un couple de variables d'environnement système. Au démarrage, le débogueur interroge les emplacements définis dans _NT_ALT_SYMBOL_PATH et _NT_SYMBOL_PATH, généralement configurés par un fichier de commandes de débogage à l'aide de la commande SET. Par exemple :

```
set _NT_SYMBOL_PATH = c:\Sympath
```

Le chemin de symbole est créé par la concaténation des contenus de _NT_ALT_SYMBOL_PATH et _NT_SYMBOL_PATH, dans cet ordre. En règle générale, le chemin est défini par la seule variable _NT_SYMBOL_PATH. _NT_ALT_SYMBOL_PATH, s'il est facultatif, est utilisable dans divers cas particuliers, par exemple un qui aurait à négocier avec des versions privées de symboles partagés.

Une autre manière de configurer le chemin des symboles avant que naisse une instance de débogueurs est de le faire au moyen de paramètres d'entrée, à savoir le paramètre -y, que l'on ajoute à la commande d'appel. Par exemple :

```
C:\>windbg –y c:\symbols <image.exe>
```

Dernière méthode qui permette de configurer les symboles, aidée de la version graphique du débogueur. Dans l'espace de travail de WinDbg, cliquez sur File, sélectionnez Symbol File Path, puis indiquez le chemin des symboles.

Validation des symboles

Que le débogueur manipule des symboles corrects est un prérequis essentiel de toute activité de débogage. La première option est d'utiliser la commande lm (List Loaded Modules) qui, utilisée avec le paramètres l, Affiche seulement les modules dont les informations de symbole ont été chargé, et averti quand elles sont éronnées.

```
lkd> lm l
start             end                 module name
fffff802`e6a02000 fffff802`e714e000   nt         (pdb symbols)          c:\websymbols\ntkrnlmp.pdb
\7B3C9BFCDF6643ABAACE89E4C9C333812\ntkrnlmp.pdb
```

Autre solution, inspecter les résultats de la commande .reload en mode verbeux.

```
lkd> !sym noisy
noisy mode - symbol prompts on
lkd> .reload nt
DBGHELP: nt - public symbols
         C:\WinDDK\7600.16385.1\Debuggers\sym\ntkrnlmp.pdb\7B3C9BFCDF6643ABAACE89E4C9C333812\ntkrnlmp.pdb
```

Le débogueur fournit en plus une commande d'extension qui puisse vérifier la validité du fichier de symbole par rapport au fichier d'image. La commande prend en entrée soit le nom du fichier image exécuté soit l'adresse à laquelle le fichier image est mappé en mémoire, et évalue la qualité des symboles. Par exemple :

```
lkd> !chksym nt

ntkrnlmp.exe
    Timestamp: 5165E551
  SizeOfImage: 74C000
          pdb: ntkrnlmp.pdb
      pdb sig: 7B3C9BFC-DF66-43AB-AACE-89E4C9C33381
          age: 2

Loaded pdb is c:\websymbols\ntkrnlmp.pdb\7B3C9BFCDF6643ABAACE89E4C9C333812\ntkrnlmp.pdb

ntkrnlmp.pdb
      pdb sig: 7B3C9BFC-DF66-43AB-AACE-89E4C9C33381
          age: 2

MATCH: ntkrnlmp.pdb and ntkrnlmp.exe
```

Chargement différé des symboles

Les débogueurs Windows standards s'appuient à des fins d'optimisation sur un traitement différé des symboles, ne les chargeant en l'occurence qu'en cas de stricte nécessité, chaque fois qu'il se présente un symbole non reconnu. Certains symboles peuvent en conséquence n'être pas synchronisés avec la session en cours. La commande de base via laquelle tenir compte de cette situation est la commande .reload, qui contrairement à ce que son nom le laisse penser, ne recharge pas les informations de symboles, mais force en réalité le débogueur à supprimer tout ou partie de celles qu'il connait. La commande fait en fait savoir au débogueur que les fichiers de symboles concernés pourraient avoir changé, ou encore qu'un nouveau module doit être ajouté à la liste des modules. Les formes les plus usuelles de la commande .reload sont les suivantes :

- *.reload* Supprime les informations de symbole concernant l'ensemble des modules chargés. Toute tentative de résoudre un symbole entraine le rechargement du fichier de symboles à partir du disque.
- *.reload <module>* Supprime les informations de symbole du module spécifié. Toute tentative de résoudre un symbole recharge le fichier de symboles à partir du disque.
- *.reload /f <module>* Force le débogueur à charger le fichier de symboles associé à un module.
- *.reload nt* Recharge le fichier de symboles correspondant à la version du noyau exécuté par le système cible. Cette option ne concerne que le débogage en mode noyau.
- *.reload /user* Recharge les fichiers de symboles en mode utilisateur pour le processus actif. Cette option ne concerne que le débogage en mode noyau.
- *.reload <module>=address,size* Les commandes passées en revue jusqu'ici sur des informations stockées à la fois dans l'image du module concernée par la session de débogage et dans le bloc de contrôle du processeur.

Si pour une raison quelconque, ces données viendraient à être manquantes ou incorrectes, il est possible de forcer le chargement des symboles en spécifiant l'adresse et l'occupation mémoire du module.

Le chargement différé des symboles (actif par défaut) peut être activé ou désactivé à l'aide de la commande de la commande .symopt, accompagnée dans ce contexte de l'option SYMOPT_DEFERRED_LOADS (0x4).

Commande symopt

Un certain nombre d'options, modifiables à l'aide de la commande symopt, sont disponibles pour contrôler la façon dont les symboles sont chargés et utilisés.

Flag	Option	Actif par défaut
0x1	SYMOPT_CASE_INSENSITIVE	Oui
0x2	SYMOPT_UNDNAME	Oui
0x4	SYMOPT_DEFERRED_LOADS	Oui
0x8	SYMOPT_NO_CPP	Non
0x10	SYMOPT_LOAD_LINES	Non (KD/CDB), Oui (WinDbg)
0x20	SYMOPT_OMAP_FIND_NEAREST	Oui
0x40	SYMOPT_LOAD_ANYTHING	Non
0x80	SYMOPT_IGNORE_CVREC	Non
0x100	SYMOPT_NO_UNQUALIFIED_LOADS	Non
0x200	SYMOPT_FAIL_CRITICAL_ERRORS	Oui
0x400	SYMOPT_EXACT_SYMBOLS	Non
0x800	SYMOPT_ALLOW_ABSOLUTE_SYMBOLS	Non
0x1000	SYMOPT_IGNORE_NT_SYMPATH	Non
0x2000	SYMOPT_INCLUDE_32BIT_MODULES	Non
0x4000	SYMOPT_PUBLICS_ONLY	Non
0x8000	SYMOPT_NO_PUBLICS	Non
0x10000	SYMOPT_AUTO_PUBLICS	Oui
0x20000	SYMOPT_NO_IMAGE_SEARCH	Oui
0x40000	SYMOPT_SECURE	Non
0x80000	SYMOPT_NO_PROMPTS	Oui (KD/CDB), Non (WinDbg)
0x80000000	SYMOPT_DEBUG	Non

BCDEdit

Accompagnant la refonte de l'environnement de démarrage initiée avec la plateforme NT 6.0, l'utilitaire BCDEdit donne une vue sur le processus d'initialisation des systèmes Windows et ouvre la voie à différents scénarios le concernant.

Sur le plan fonctionnel, BCDEdit permet la configuration et le contrôle des fichiers de Données de configuration de démarrage (BCD, Boot Configuration Data), lesquels fournissent un magasin (*store*) servant à décrire les applications de démarrage et les paramètres de ces dernières. Les objets et les éléments contenus dans ce magasin remplacent à l'usage les fichiers Bootcfg.exe et Boot.ini des précédentes versions de Windows. BCDEdit peut être utilisé pour remplir de nombreuses fonctions, notamment pour créer de nouveaux magasins, pour modifier des magasins existants, pour ajouter des options de menu de démarrage, etc.

L'outil BCDEdit est fourni avec Windows dans le dossier %WINDIR%\System32 ; il n'est directement accessible que par le compte d'Administrateur principal de la machine ou par élévation des droits d'administration pour les autres comptes. Seul outil complet Microsoft à permettre d'entreprendre toutes les opérations de modification et de création de l'environnement de démarrage, BCDEdit reste limité aux types de données standard et est conçu essentiellement pour apporter des modifications courantes et peu nombreuses aux BCD. Pour des opérations plus complexes ou des types de données non standard, préférer WMI.

Opérations sur un magasin :

- */createstore* Crée un magasin des données de configuration de démarrage vide. Le magasin créé n'est pas un magasin système. Pour plus d'informations, consultez le volet \<\<_agir_sur_le_magasin\>\>.
- */export* Exporte le contenu du magasin système dans un fichier. Ce fichier peut être utilisé ultérieurement pour restaurer l'état du magasin système.
- */import* Restaure l'état du magasin système à l'aide d'un fichier de données de sauvegarde généré précédemment en utilisant l'option /export.
- */store* Spécifie un magasin autre que le magasin système actuel par défaut.
- */sysstore* Définit le dispositif de stockage du système.

Opérations sur les entrées d'un magasin :

- */copy* Crée des copies d'entrées du magasin.
- */create* Crée des entrées dans le magasin.
- */delete* Supprime des entrées du magasin.
- */mirror* Crée un miroir des entrées du magasin.

Contrôle de la sortie :

- */enum* Répertorie les entrées d'un magasin.
- */v* Mode détaillé.

Identificateurs bien connus

Pour faciliter la navigation parmi les entrées d'un magasin, l'éditeur BCD reconnait pour quelques-uns un nom convivial, plus simple à manipuler qu'un GUID. Les identificateurs bien connus suivants sont établis.

- *{badmemory}* Contient la liste globale de RAM défectueuse qui peut être héritée par toute entrée d'application de démarrage.
- *{bootloadersettings}* Contient la collection des paramètres globaux dont doivent hériter toutes les entrées d'application chargeur de démarrage Windows.
- *{bootmgr}* Indique l'entrée du Gestionnaire de démarrage Windows.
- *{current}* Représente un identifiant virtuel qui correspond à l'entrée de démarrage du système d'exploitation pour système d'exploitation en cours d'exécution.
- *{dbgsettings}* Contient les paramètres de débogage global pouvant être hérités par toute entrée d'application de démarrage.
- *{default}* Représente un identifiant virtuel qui correspond à l'entrée d'application du gestionnaire de démarrage par défaut.
- *{emssettings}* Contient les paramètres globaux EMS qui peuvent être hérités par toute entrée d'application de démarrage.
- *{fwbootmgr}* Sur les systèmes EFI, indique l'entrée de gestionnaire de démarrage micrologiciel,
- *{globalsettings}* Contient la collection des paramètres globaux dont doit hériter toute entrée d'application de démarrage.

- *{hypervisorsettings}* Contient les paramètres hyperviseur dont peuvent hériter toute entrée d'application de démarrage.
- *{legacy}* Indique le chargeur d'OS hérité Windows (Ntldr), utilisable pour démarrer un système d'exploitation Windows antérieur à Windows Vista.
- *{memdiag}* Indique l'entrée de l'application de diagnostic de la mémoire.
- *{ntldr}* Indique le chargeur d'OS hérité Windows (Ntldr), utilisable pour démarrer un système d'exploitation Windows antérieur à Windows Vista.
- *{ramdiskoptions}* Contient les options supplémentaires nécessaires au gestionnaire pour des périphériques de RAM disque.
- *{resumeloadersettings}* Contient la collection des paramètres globaux dont doit hériter toute entrée d'application de sortie de veille prolongée.

Dès lors qu'une entrée possède un identificateur bien connu, l'éditeur BCD le fait apparaitre dans la sortie de commande, excepté si le mode volubile (commutateur /v) a été choisi, auquel cas les identificateurs sont montrés sous leur forme complète.

Gestion de l'ordinateur

La console Gestion de l'ordinateur centralise l'accès à un ensemble d'utilitaires majeurs liés aux tâches fondamentales d'administration des systèmes Windows, locaux ou distants.

Chacune des méthodes suivantes peut être suivie de sorte à afficher la console de gestion de l'ordinateur : (1) ouvrez le menu technique (Win + X) puis cliquez sur le menu Gestion de l'ordinateur ; (2) depuis le groupe de programmes Outils d'administration, effectuez un double clic sur le raccourci Gestion de l'ordinateur ; (3) cliquez sur Gérer à partir du Poste de travail ; (4) depuis une invite de commande ou le contrôle Exécuter, saisissez la commande compmgmt.msc. Étant donné les fonctionnalités qu'il englobe, vous devez disposer des privilèges d'administrateur local pour exploiter tout le potentiel de Gestion de l'ordinateur.

Les outils de la console Gestion de l'ordinateur sont classés en trois grandes catégories : *Outils système*, *Stockage* et *Services et applications*. La première de ces rubriques sert de passerelle d'accès vers des outils généraux d'administration, tels que le service Planificateur de tâches, l'Observateur d'événements, ou encore les Dossiers partagés. La rubrique Stockage ne contient qu'un seul utilitaire : Gestion des disques, lequel fait figure d'assistant multi-terrain pour tout ce qui concerne partitions et volumes. Enfin, la rubrique Services et applications concentre les propriétés et les réglages des services Windows, et le contrôle WMI. Pour plus de détails sur les différents logiciels que met à disposition la console Gestion de l'ordinateur, reportez-vous au premier chapitre, section Outils d'administration.

Une fois donné le départ à une console Gestion de l'ordinateur, vous êtes alors en mesure d'administrer l'ordinateur sur laquelle cette console s'exécute (remarquez la mention (Local) au niveau de la première ligne de l'arborescence). Il est cependant envisageable de se servir de cette même interface en vue d'administrer à distance une autre station de travail. Pour ce faire, sélectionnez la racine de la console Gestion de l'ordinateur (Local) puis le menu Action - Se connecter à un autre ordinateur (ou utilisez le menu contextuel). Dans la boite de dialogue nouvellement apparue, nommée Sélectionner un ordinateur, saisissez le nom pleinement qualifié de l'ordinateur concerné, ou utilisez le bouton Parcourir pour rechercher la machine que vous souhaitez gérer à distance, puis cliquez sur OK. Si la connexion est correctement établie, le nom complet de l'ordinateur apparait entre parenthèses à la racine de la console. Si vous êtes détenteur de privilèges administratifs locaux sur le poste distant, vous pouvez contrôler et agir sur tous les éléments de ce dernier, à l'exception des périphériques.

Utilitaire Configuration du système (Msconfig)

L'utilitaire Configuration du système donne accès à presque tous les paramètres de démarrage.

Les options de configuration mises en avant dans la fenêtre de Configuration du système se divisent en cinq grandes catégories.

- *Général* Configuration générale de l'amorçage Windows : démarrage normal, en mode diagnostic (*safe mode*) ou sélectif.
- *Démarrer* Options du chargeur d'amorçage. Cet onglet fait en quelque sorte figure d'interface graphique pour l'utilitaire Bcdedit.
- *Services* Contrôle l'activation et le démarrage des services Windows. Il s'agit d'une version simplifiée de la console Gestion des services (services.msc).
- *Démarrage* Permet de visualiser et de contrôler les logiciels s'exécutant automatiquement lors de l'ouverture d'une session utilisateur.
- *Outils* Contient des lignes de commandes préconfigurées faisant miroir à diverses interfaces d'administration. Il ne s'agit pour la plupart que de raccourcis vers des éléments du Panneau de configuration.

Utilitaires d'administration

- *Contrôle WMI* Permet de configurer et contrôler l'infrastructure de gestion Windows (WMI).
- *DiskPart* Permet d'administrer disques, partitions et volumes depuis l'invite de commandes.
- *Dossiers partagées* Permet de visualiser, créer ou supprimer des répertoires partagées ainsi que les sessions en cours et les fichiers ouverts.
- *Gestion de l'ordinateur* Permet d'accéder à des outils majeurs d'administration du système, des services et du stockage.
- *Gestion des disques* Permet de visualiser et modifier la structure des partitions et volumes des disques dont est équipé l'ordinateur. Pour en savoir plus, consultez le chapitre Gestion du stockage.
- *Gestionnaire des taches* Permet d'afficher des informations d'utilisation concernant les ressources système.
- *Gestionnaire de périphériques* Permet de contrôler l'ensemble des périphériques et pilotes de matériel associés. Pour en savoir plus, consultez le chapitre Système d'E/S.
- *Performances* Permet de mesurer les performances du système et de connaitre les causes des problèmes éventuels à ce niveau.
- *Moniteur de ressources* Permet d'afficher des informations d'utilisation détaillées concernant les ressources système.
- *Ordinateur* Permet de visualiser rapidement les périphériques de stockage de la machine.
- *Observateur d'évènements* Permet d'accéder à l'ensemble des journaux d'événements (liste d'avertissement et erreurs) de l'ordinateur local ou de la machine sélectionnée.
- *Planificateur de tâches* Permet d'afficher, de créer et de gérer des tâches planifiées. Pour en savoir plus, consultez le chapitre Mécanisme de gestion.
- *Système* Affiche des informations de base sur l'ordinateur (caractéristiques techniques) et le système d'exploitation utilisé.
- *Services* Permet de gérer l'ensemble des services présents sur la station de travail. Pour en savoir plus, consultez le chapitre Mécanismes de gestion.

Variables d'environnement

Pour créer, supprimer ou modifier les valeurs des variables d'environnement depuis l'interface graphique, ouvrez l'application Propriétés système (SystemPropertiesAdvanced) et cliquez sur bouton Variables d'environnement situé en bas de la fenêtre.

Pour afficher une liste des variables d'environnement depuis la ligne de commandes, ouvrez une fenêtre d'invite de commandes et saisissez la commande `set`. Vous devriez alors voir quelque chose de similaire à ce qui suit.

```
C:\Users\arnaud>set
ALLUSERSPROFILE=C:\ProgramData
APPDATA=C:\Users\arnaud\AppData\Roaming
CommonProgramFiles=C:\Program Files\Common Files
CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files
CommonProgramW6432=C:\Program Files\Common Files
COMPUTERNAME=BONIFACE
ComSpec=C:\WINDOWS\system32\cmd.exe
configsetroot=C:\WINDOWS\ConfigSetRoot
DriverData=C:\Windows\System32\Drivers\DriverData
EFC_9256=1
FPS_BROWSER_APP_PROFILE_STRING=Internet Explorer
FPS_BROWSER_USER_PROFILE_STRING=Default
HOMEDRIVE=C:
HOMEPATH=\Users\arnaud
LOCALAPPDATA=C:\Users\arnaud\AppData\Local
LOGONSERVER=\\BONIFACE
NUMBER_OF_PROCESSORS=6
OneDrive=C:\Users\arnaud\OneDrive
OS=Windows_NT
Path=C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Program Files (x86)\VMware\VMware Player\bin\;C:\Program
 Files (x86)\Intel\Intel(R) Management Engine Components\iCLS\;C:\Program Files\Intel\Intel(R) Management Engine Components
\iCLS\;C:\Windows\system32;C:\Windows;
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
POWERSHELL_DISTRIBUTION_CHANNEL=MSI:Windows 10 Home
PROCESSOR_ARCHITECTURE=AMD64
PROCESSOR_IDENTIFIER=Intel64 Family 6 Model 158 Stepping 10, GenuineIntel
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=9e0a
ProgramData=C:\ProgramData
ProgramFiles=C:\Program Files
ProgramFiles(x86)=C:\Program Files (x86)
ProgramW6432=C:\Program Files
PROMPT=$P$G
PSModulePath=C:\Program Files\WindowsPowerShell\Modules;C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules
PUBLIC=C:\Users\Public
SESSIONNAME=Console
SystemDrive=C:
SystemRoot=C:\WINDOWS
TEMP=C:\Users\arnaud\AppData\Local\Temp
TMP=C:\Users\arnaud\AppData\Local\Temp
USERDOMAIN=BONIFACE
USERDOMAIN_ROAMINGPROFILE=BONIFACE
USERNAME=arnaud
USERPROFILE=C:\Users\arnaud
windir=C:\WINDOWS
```

Pour voir à quoi se rapporte une variable spécifique, utilisez la commande `set` suivie du nom de la variable.

La liste qui suit passe en revue certaines des principales variables d'environnement.

- **ALLUSERSPROFILE** Emplacement du profil *Tous les utilisateurs*. Exemple : C:\ProgramData.
- **APPDATA** Emplacement par défaut du stockage des données relatives aux applications. Exemple : C:\Users\(nom)\AppData\Roaming, où (nom) représente un nom d'utilisateur.

- **COMMONPROGRAMFILES** Emplacement des fichiers communs à diverses applications. Exemple : C:\Program Files\Common Files.
- **COMPUTERNAME** Nom de l'ordinateur.
- **COMSPEC** Chemin d'accès complet vers l'interprète de commandes. Exemple : C:\WINDOWS\system32\cmd.exe.
- **ERRORLEVEL** Code d'erreur de la dernière commande utilisée.
- **HOMEDRIVE** Volume disque où réside le dossier de l'utilisateur en cours. Exemple : C:.
- **HOMEPATH** Chemin d'accès au dossier personnel de l'utilisateur en cours. Exemple : \Users\(nom), où (nom) représente un nom d'utilisateur.
- **LOCALAPPDATA** Paramètres personnalisés des applications pour l'utilisateur en cours. Exemple : C:\Users\(nom)\AppData\Local.
- **LOGONSERVER** Nom du contrôleur de domaine ayant validé la session courante.
- **NUMBER_OF_PROCESSORS** Nombre de processeurs que compte l'ordinateur.
- **OS** Nom du système d'exploitation. Exemple : Windows_NT.
- **PATH** Liste de chemins d'accès vers les répertoires contenant des fichiers exécutables, séparés par des points-virgules. Pour lancer un programme présent dans l'un de ces dossiers, il n'est pas nécessaire de le faire précéder de son chemin d'accès. Exemple : C:\WINDOWS\System32\;C:\Program Files.
- **PATHEXT** Liste des extensions considérées comme exécutables. Exemple : .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC.
- **PUBLIC** Dossier public accessible à tous les utilisateurs de l'ordinateur. Exemple : C:\Users\Public.
- **PROCESSOR_ARCHITECTURE** : Architecture de processeur à laquelle répond l'ordinateur. Exemple : x86 ou AMD64.
- **PROCESSOR_IDENTIFIER** Description du processeur. Exemple : Intel64 Family 6 Model 158 Stepping 10, GenuineIntel.
- **PROCESSOR_LEVEL** Modèle du processeur.
- **PROCESSOR_REVISION** Numéro de version du processeur.
- **PROMPT** Paramètres d'invite de commandes pour l'interprète de commandes courant. Exemple : PG.
- **PSMODULEPATH** Emplacement des modules pour la suite PowerShell. Exemple : C:\Program Files\WindowsPowerShell\Modules.
- **SESSIONNAME** Nom de la session en cours. Exemple : Console.
- **SYSTEMDRIVE** Lecteur de la racine du système. Exemple : C:.
- **SYSTEMROOT** Emplacement du répertoire racine du système d'exploitation. Exemple : C:\WINDOWS.
- **TEMP / TMP** Répertoires temporaires par défaut employés par les programmes pour l'utilisateur actuellement connecté. Exemple : C:\Users\(nom)\AppData\Local\Temp.
- **USERDOMAIN** Nom du domaine auquel se rattache le compte de l'utilisateur actuellement connecté.
- **USERNAME** Nom de l'utilisateur actuellement connecté.

- **USERPROFILE** Emplacement du profil de l'utilisateur actuellement connecté. Exemple : C:\Users\(nom).
- **WINDIR** Emplacement du répertoire du système d'exploitation. Exemple : C:\WINDOWS.

Console Windows

La Console Windows est une forme d'interface utilisateur qui permet d'exécuter des logiciels conçus pour être pilotés à l'aide de commandes textuelles. Entrent dans cette catégorie, par exemple, les interpréteurs de commandes Cmd et Windows PowerShell, différentes commandes intégrées utiles pour l'administration du système, ainsi que pléthore d'utilitaires de gestion ou de diagnostic dédiés.

Chaque console Windows possède une mémoire tampon de sortie vidéo et une mémoire tampon d'entrée.

Dans la configuration par défaut, chaque processus exécutant une application fondée sur du texte a une console qui lui revient et à laquelle il est automatiquement associé. Les processus créés en tant qu'entités détachées (flag DETACHED_PROCESS de CreateProcess) ne s'accompagnent d'une console qu'à partir du moment où ils en créent une nouvelle (AllocConsole). Les applications dotées d'une interface graphique, par nature, ne présentent pas de lien avec une quelconque console. Un processus peut être associé seulement à une console unique. De ce fait, toute tentative d'association d'une console à un processus échoue (fonction Windows AllocConsole) si ce dernier est déjà pourvu en la matière. Un processus peut utiliser la fonction FreeConsole pour se désolidariser de sa console actuelle, puis donner lieu pour son propre usage à une nouvelle console (AllocConsole) ou opter pour la mise en relation avec une console déjà existante (AttachConsole).

Le tableau qui suit énumère les fonctions utilisables pour accéder à une console.

Tableau 2.11. Fonctions de gestion de la console

Fonction	Description
AddConsoleAlias	Définit un alias de console pour l'exécutable spécifié.
AllocConsole	Crée une nouvelle console pour le processus appelant.
AttachConsole	Rattache le processus appelant à la console à laquelle est associé un processus donné.
ReadConsole	Lit les caractères en entrée depuis le tampon d'entrée d'une console donnée.
SetConsoleCP	Définit la page de code en vigueur dans la console associée au processus appelant.
SetConsoleTitle	Définit le titre de la fenêtre console courante.
WriteConsole	Ecrit une chaîne de caractères dans le tampon de sortie d'une console donnée.

Observateur d'événements

Intermédiaire entre l'utilisateur et un panel d'actions (et de résultats de celles-ci) que consigne le système d'exploitation, l'Observateur d'événements est l'un des outils de diagnostic les plus vitaux de Windows. L'Observateur d'événements affiche des informations détaillées sur les événements significatifs de l'ordinateur, autant matériels (par exemple une défaillance dans un dispositif) que logiciels (par exemple un programme ne démarrant pas comme prévu), amenant l'outil à une place de choix pour contrôler la santé du système et régler les problèmes lorsqu'ils surviennent.

Accès à l'Observateur d'événements. Vous pouvez ouvrir l'Observateur d'événements à l'aide de l'une ou l'autre des méthodes suivantes.

- Depuis le contrôle Exécuter ou une fenêtre d'invite de commandes, entrez eventvwr.exe ou eventvwr.msc.
- Ouvrez la console Gestion de l'ordinateur (compmgmt.msc), développez Outils système et cliquez sur Observateur d'événements.

LiveKd

LiveKd permet d'utiliser les débogueurs noyau standard de Microsoft pour examiner un système au cours de son fonctionnement normal, sans être obligé de le préparer spécifiquement dans ce but (démarrage avec prise en charge du débogage noyau local, recours à un second ordinateur faisant office d'hôte, etc.).

LiveKd est compatible avec les versions x86 et x64 de Windows. Il a compte compte tenu des fonctions qu'il remplit besoin de droits administratifs, dont le privilège Déboguer des programmes.

Par défaut, LiveKd exécute le débogueur noyau ligne de commande (Kd). Pour exécuter le débogueur graphique (Windbg), spécifiez le commutateur -w. Pour exécuter un autre débogueur, spécifiez le commutateur -k. Des arguments de ligne de commande supplémentaires peuvent être fournis.

En interne, LiveKd s'appuie sur la mémoire physique en vue de simuler un fichier de vidange sur incident (dump), rendant de ce fait toutes les opérations autorisées dans ce contexte intelligibles auprès du débogueur. Procéder de la sorte n'est n'est cependant pas sans comporter quelque risque : le débogueur est-il ainsi susceptible de se trouver dans une situation où les structures de données sont en cours de modification par le système, et donc incohérentes. Chaque fois que vous sollicitez LiveKd, il commence par donner lieu à un nouvel instantané de l'état du système et démarre le débogueur pour lequel vous avez opté. Pour actualiser la vue, sortez du débogueur (commande "q"), qui vous demandera alors s'il doit redémarrer.

Driver Verifier

Une large majorité des problèmes susceptibles de frapper le bon fonctionnement du poste de travail sont issues de pilotes défectueux (mal conçus dès le départ ou mal entretenus par la suite). Partant de ce constat, Windows inclut un outil de dépannage puissant, appelé Vérificateur de pilotes, dont la principale fonction est de repérer parmi les actions des pilotes lesquelles sont de nature à compromettre la fiabilité du système, par exemple des appels de fonctions ou des accès mémoire illégaux.

Lorsque le Vérificateur de pilotes se trouve en face d'un comportement inapproprié, il crée de manière proactive une exception. Il devient ainsi possible d'analyser le fichier image mémoire généré et de la sorte découvrir le(s) pilote(s) auquel imputer l' anomalie.

Le vérificateur de pilotes est particulièrement impliqué dans la procédure d'amorçage du système, où il inspecte minutieusement chaque pilote. Il effectue en plus de cela la plupart des contrôles demandés par WHQL durant le processus de certification et de signature.

Pour commencer à utiliser le Gestionnaire du vérificateur de pilotes, ouvrez une fenêtre d'invite de commandes ou le contrôle Exécuter, saisissez verifier.exe puis validez. Dans la boîte de dialogue nouvellement affichée, sélectionnez Créer des paramètres standard. Cliquez sur Suivant ; après quoi vous devriez être en mesure de sélectionner les pilotes à vérifier. Les options suivantes sont accessibles :

- Sélectionner automatiquement les pilotes non signés
- Sélectionner automatiquement les pilotes conçus pour une ancienne version de Windows
- Sélectionner automatiquement tous les pilotes installés sur cet ordinateur
- Choisir des noms de pilotes dans une liste

Après un clic sur le bouton Suivant, le logiciel génère une liste des pilotes qui correspondent aux conditions spécifiées. Deux cas de figure peuvent alors être considérées : (1) Consulter la liste pour découvrir quels sont les pilotes identifiées selon la méthode choisie, puis cliquer sur Annuler ; (2) Cliquer sur Terminer pour quitter l'assistant et redémarrer l'ordinateur.

Pour désactiver le Vérificateur de pilotes, et ce faisant l'enjoindre à cesser toute vérification au démarrage de l'ordinateur, exécutez-le et sélectionnez Supprimer les paramètres existants dans la boite de dialogue initiale. Vous pouvez également saisir verifier /reset depuis une invite de commandes.

Tableau 2.12. Options de vérification

Hexadécimal	Option
0x00000001	Special Pool
0x00000002	Force IRQL Checking
0x00000004	Low Resources Simulation
0x00000008	Pool Tracking
0x00000010	I/O Verification
0x00000020	Deadlock Detection
0x00000040	Enhanced I/O Verification
0x00000080	DMA Verification
0x00000100	Security Checks
0x00000200	Force Pending I/O Requests
0x00000400	IRP Logging
0x00000800	Miscellaneous Checks
0x00040000	Systematic low resources simulation
0x00200000	NDIS/WIFI verification
0x00800000	Kernel synchronization delay fuzzing

Vérificateur de pilotes prend en compte de nombreuses options :

- *Détection de blocage* Surveille l'utilisation de différents objets exposés en tant que primitives de synchronisation (spinlocks, mutex, etc.) en recherchant des modèles de comportement susceptibles d'indiquer un verrou mortel.
- *Contrôle DMA* Contrôle l'utilisation adéquate des interfaces (tampons et registres) qui autorisent un accès direct à la mémoire (DMA, Direct Memory Access).
- *Pool spécial* Force les routines d'allocation (ExAllocatePool, IoAllocateIrp, etc.) à s'alimenter auprès d'un pool spécial (par opposition aux pools de mémoire noyau ordinaires) spécifiquement conçu pour la détection des débordements de tampons et des erreurs d'accès.
- *Suivi de pool* Veille à ce qu'un pilote, à l'issue de son exécution, ait restitué la mémoire allouée pour son compte.
- *Vérification d'E/S* Surveille l'accomplissement interne et l'impact externe des fonctions d'E/S définies par les pilotes, visant ici à relever de leur part toute éventuelle gestion non avisée.
- *Vérification IRQL forcée* Incrimine les pilotes qui accèdent à de la mémoire paginable à un IRQL trop élevé pour cette opération.
- *Simulation de ressources faibles* Évalue comment se comportent les pilotes face à une saturation des ressources système - et, partant, à un manque de mémoire noyau.
- *Vérifications de sécurité* Identifie les situations courantes susceptibles d'impacter négativement la sécurité du système.
- *Journalisation IRP* Enregistre ce que font les pilotes des IRP qui leur sont adressés.
- *Forcer les demande d'E/S en attente* Met à l'épreuve la capacité des pilotes à gérer correctement des E/S asynchrones.
- *Analyse du temps d'exécution* Chronomètre l'exécution des routines d'achèvement d'E/S et d'annulation des pilotes vérifiés et signale lesquelles excèdent les temps prescrits.
- *Retards de synchronisation de noyau* Propage à différents stades de l'exécution des pilotes vérifiés des délais aléatoires.

- *Vérification d'intégrité de disque* Surveille les accès au média de stockage en vue de déterminer s'il enregistre les informations conformément aux attentes.
- *Vérification SCSI* Supervise les interactions qui émanent de l'utilisation du bus SCSI.
- *Vérification NDIS/WIFI* Détermine si un pilote NDIS ou WIFI interagit correctement avec les autres composants du système d'exploitation, dont le noyau.
- *Conformité WDF* Examine la conformité des pilotes aux exigences essentielles imposées par le modèle de pilote KMDF.

S'il montre son utilité notamment dans le cadre de la conception de code mode noyau, Vérificateur de pilotes, étant donné l'étendue des fonctionnalités qu'il offre, est de surcroit un outil puissant entre les mains des administrateurs, qui l'emploiront selon le contexte pour remonter la piste d'un effondrement du système, débusquer des erreurs matérielles, etc.

La plupart des paramètres établis concernant la vérification des pilotes sont enregistrés dans le Registre sous HKLM\System\CurrentControlSet\Control\Session Manager\Memory Management. Parmi toutes les valeurs hébergées à cet emplacement, deux méritent une attention particulière : VerifyDriverLevel, qui contient un masque binaire représentant les types de vérification activés, et VerifyDrivers, qui renferme les noms des pilotes soumis à examen. Si vous avez choisi de procéder à un audit complet complet (tous les pilotes), VerifyDrivers fait alors mention d'un astérisque (caractère *). Notez que selon les paramètres pour lesquels vous avez opté, vous serez peut-être amené à réamorcer le système.

Durant la phase d'amorçage, le gestionnaire de mémoire consulte les valeurs de registre associées à la vérification des pilotes et détermine, partant de ces informations, quels pilotes sont concernés et quelles options sont en vigueur. (Notez que, si l'ordinateur a démarré en mode sans échec, tous les paramètres de Vérificateur de pilotes sont ignorés.). Par la suite, pour peu que la configuration appelle à l'examen d'au moins un pilote, le noyau vérifie le nom de chaque pilote qu'il s'apprête à charger par rapport à la liste des pilotes devant faire l'objet d'une procédure de contrôle. Pour chaque pilote dont le nom est répertorié, le noyau initie son chargement non plus à l'aide de la routine commune en la matière (NtLoadDriver), mais par le biais de la fonction spécialisée VfLoadDriver, laquelle se charge de remplacer les références que le pilote effectue à un certain nombre de fonctions du noyau ou de l'exécutif par des références à des versions équivalentes, mais mieux adaptées à la détection de bogues, du Vérificateur de pilotes.

Systeminfo

L'utilitaire en ligne de commande Systeminfo affiche un résumé de la configuration matérielle et logicielle de l'ordinateur.

Pour exécuter Systeminfo, ouvrez une fenêtre d'invite de commandes, saisissez `systeminfo` et validez par l'appui de la touche Entrée. En plus du format liste, Systeminfo offre deux autres modes de présentation conçus pour faciliter la lecture et le traitement ultérieur des résultats : tableau et CSV. Pour utiliser l'un de ces formats, ajoutez le commutateur /Fo à la commande. Le cas échéant, redirigez la sortie vers un fichier de sorte à conserver une trace des informations recueillies. Par exemple, la commande `systeminfo /fo csv > infos.csv` génére une sortie CSV et l'enregistre dans le fichier infos.csv.

Le commutateur /S permet d'obtenir des informations système pour un ordinateur distant. Utiliser les commutateurs /U et /P pour fournir, respectivement, le nom d'utilisateur et le mot de passe d'un compte autorisé sur l'ordinateur cible.

Obtention d'informations avancées sur le système (Msinfo)

Pour obtenir des informations détaillées concernant le système local ou d'autres ordinateurs (station distante), utilisez Informations système (Msinfo32.exe). Toutes les statistiques de configuration fournies le sont par l'intermédiaire du service WMI.

Pour exécuter Informations système, dans une fenêtre d'invite de commandes ou le contrôle Exécuter, saisissez msinfo32. La fenêtre principale de l'application s'ouvre alors sur la page Résumé Système, laquelle centralise maintes informations : version exacte du système, processeur, version du BIOS, etc. La navigation parmi les informations système s'effectue de la même façon que dans l'Explorateur Windows ou dans une console MMC. Cliquez sur une rubrique dans le volet de gauche pour afficher le contenu associé dans celui de droite.

Les informations affichées dans Informations système se répartissent en trois parties majeures :

- *Ressources matérielles* Fournit des informations détaillées sur les E/S, les requêtes d'interruption, les canaux d'accès direct à la mémoire (DMA) et les périphériques.
- *Composants* Fournit des informations détaillées sur les composants installés, allant des codecs multimédia aux ports USB en passant par les périphériques d'entrée et la gestion réseau.
- *Environnement logiciel* Fournit des informations détaillées sur la configuration courante du système d'exploitation.

MASM

Le logiciel Microsoft Macro Assembler, plus connu sous l'acronyme MASM, est un assembleur compatible avec les jeux d'instruction 32 et 64 bits. Il fut à l'origine développé par Microsoft pour le développement de leur système d'exploitation MS-DOS, et évolue depuis lors en parallèle des avancées technologiques des processeurs et de Microsoft Windows.

MASM supporte une grande variété de macros aidant à la programmation en langage assembleur ainsi que des idiomes de programmation structurée, incluant des constructions de haut niveau pour les boucles, les appels de procédures, les branchements, etc., permettant de programmer relativement aisément des applications fonctionnant sous Windows.

Vendu en premier lieu en tant que logiciel autonome, Microsoft rompit avec cette pratique commerciale pour la version 6.12 de MASM, préférant l'inclure aux différents kits de développement en provenance de Redmond. MASM 7.0 fut inclus avec Visual C .NET 2002, MASM 7.1 avec Visual C .NET 2003 et MASM 8.0 avec Visual C++ .NET 2005 (cette dernière version est la première à pouvoir assembler du code 64 bits). Les versions récentes de MASM sont incluses avec Microsoft Visual Studio.

Dans les premières versions de MASM, les fichiers objet générés l'étaient selon le format d'objet relogeable (OMF, *Relocatable Object Module Format*), utilisé pour créer les binaires vers MS-DOS et OS/2. Depuis la version 6.1, MASM est capable de produire le format standard des exécutables et bibliothèques pour les systèmes d'exploitation Windows 32 et 64 bits, à savoir PE/COFF.

MASM adopte la syntaxe Intel, du nom du fabricant de semi-conducteurs et processeurs qui en est à l'origine. (La syntaxe Intel est une forme de codage en assembleur majoritairement en vogue dans l'écosystème Windows. Elle s'oppose à la syntaxe AT&T, surtout utilisée sous UNIX, qui propose une sémantique différente. Par exemple, une instruction en norme Intel `mov eax, 0FFh` donnera en norme AT&T `movl $0xFF, %eax`. Les deux syntaxes ont leurs partisans et leurs détracteurs. Du fait de ses liens avec les systèmes Windows, et malgré que d'autres assembleurs puissent sembler plus attractifs, MASM reste encore aujourd'hui un assembleur phare de la communauté des programmeurs en assembleur. Utilisé dans les développements internes de Microsoft pour ses systèmes d'exploitation, MASM aussi l'un des éléments centraux de la chaîne de compilation des langages C et assembleur dans lesquels le code source de Windows est écrit.

Architecture des ordinateurs

Multiprocesseurs et multicoeurs

Une large majorité des ordinateurs actuels incorpore des fonctions plus ou moins avancées de parallélisation physique. À cet égard, Windows reconnait deux types de systèmes : les machines uniprocesseur, où une

seule séquence d'exécution (thread) est envisageable à un moment donné, et les machines multiprocesseurs, lesquelles rendent possible l'exécution parallèle et simultanée de plusieurs tâches. Cette seconde configuration englobe en réalité différents modèles d'architectures, à savoir :

- *Multiprocesseur* La machine héberge des processeurs physiques distincts.
- *Multicoeur* Plusieurs processeurs (coeurs) sont installées sur une même puce, reliées par un réseau d'interconnexion.
- *Hyperthreading* Un seul processeur physique peut gérer deux threads par cycle d'horloge.

Des combinaisons de ces différents schémas sont aussi possibles. Ainsi, une machine dotée de deux processeurs physiques, chacun muni de deux coeurs et chaque coeur étant hyperthreadé, assure au système d'exploitation huit processeurs virtuels.

Simultaneous Multithreading

Un processeur conventionnel n'est réellement capable d'accomplir qu'une seule tâche à la fois, et doit donc basculer alternativement d'un thread à un autre - donnant ainsi l'illusion du multitâche. Pour éviter une perte de temps liée à l'attente de nouvelles instructions, et réduire les délais qu'engendrent les commutations de contexte, les fondeurs de processeurs se sont impliqués dans l'amélioration du parallélisme (au niveau matériel, s'entend), et ont mis en avant des des procédés d'optimisation pour que les threads puissent partager les pipelines, les caches et les registres : on parle alors d'exécution SMT (Simultaneous Multithreading).

L'objectif général de toute conception SMT est de de limiter la sous-utilisation des structures internes du processeur. Dans une telle configuration, certaines structures sont dupliquées, comme les valeurs des registres ou la table de renommage, tandis que d'autres sont partagées, comme les caches ou les unités fonctionnelles (UAL, unités d'accès mémoire, etc.).

Pipeline

L'un des mécanismes de base qui régit les processeurs modernes est le *pipeline*.

Dans une microarchitecture standard (comprendre dépourvue de toute forme d'optimisation) l'exécution des instructions est accomplie de manière purement séquentielle, ce qui implique qu'une nouvelle instruction ne peut être prise en compte sans attendre que l'actuelle soit terminée. Par contraste, dans un processeur doté d'un pipeline, l'exécution des instructions est décomposée en plusieurs étapes fonctionnellement indépendantes (qui concernent des circuits séparés), et plusieurs instructions peuvent se trouvent simultanément en cours d'exécution.

Au plus fort degré d'abstraction, le principe du pipeline est comparable à celui d'une ligne (ou chaîne) de montage, où chaque unité s'emploie à une tâche spécifique, et participe de la sorte à un meilleur rendement. Chacune des étapes qui s'inscrit dans un processus pipeliné est appelé étage. Le nombre d'étages d'un pipeline est appelé sa profondeur. Afin de mieux cerner concrètement le sujet, voyons un exemple des plus notoires, le pipeline RISC originel, lequel assure l'exécution principale en 5 étapes :

- *Lecture de l'instruction (IF, Instruction Fetch)* Chargement du premier mot d'instruction de la mémoire principale vers le registre d'instruction.
- *Décodage (ID, Instruction Decode)* décode l'instruction et adresse les registres.
- *Exécution (EX, Execute)* Exécute l'instruction.
- *Mémoire (MEM, Memory)*
- *Écriture du résultat (WB, Write Back)* Modification de l'opérande destination pour les instructions arithmétiques ou logiques, ou pour ce qui est des instructions de contrôle, modification du registre d'instruction.

Notez que l'organisation en pipeline ne diminue en aucun cas le temps nécessaire pour achever une instruction (autrement dit, aucune n'est exécutée plus rapidement), mais augmente par contre le nombre d'instructions exécutées par cycle.

Le fonctionnement idéal d'un pipeline suppose un certain nombre de paramètres : (1) chacune des sous-opérations s'appuie sur des points différents du chemin de données ; (2) les instructions exécutées sont indépendantes les unes vis-à-vis des autres ; et (3) l'instruction devant être traitée après celle en cours est est la suivante en mémoire (ou, à tout le moins, peut être facilement déterminée). Dans les faits, il peut arriver qu'une de ces conditions, voire toutes, ne soit pas remplie, introduisant ce que l'on appelle alors un aléa. Partant des critères susmentionnés, trois sortes d'aléa se distinguent nettement :

- Aléa structurel
- Aléa de données
- Aléa de contrôle

Listes

Bon nombre de composants au sein de Windows emploient pour leur usage interne des listes. La liste qui suit passe en revue les fonctions courantes permettant leur manipulation.

- **InitializeListHead** Prépare pour l'utilisation une liste.
- **InsertHeadList** Insère une nouvelle entrée au début d'une liste.
- **AppendTailList** Insère une liste à la fin d'une autre.
- **InsertTailList** Insère une nouvelle entrée à la fin d'une liste.
- **IsListEmpty** Vérifie qu'une liste est vide.
- **RemoveHeadList** Supprime l'entrée située au début d'une liste.
- **RemoveTailList** Supprime l'entrée située à la fin d'une liste.
- **RemoveEntryList** Supprime une entrée spécifique parmi une liste.

Toutes ces fonctions se fondent sur une structure de données commune (voir définition ci-après), renfermant des pointeurs vers l'élément suivant et précédent d'une liste.

```
typedef struct _LIST_ENTRY {
    struct _LIST_ENTRY *Flink;
    struct _LIST_ENTRY *Blink;
} LIST_ENTRY, *PLIST_ENTRY;
```

Cartes binaire (bitmap)

Divers objectifs inhérents aux systèmes d'exploitation, au premier rang desquels la gestion des ressources, imposent fréquemment de pister l'état d'un grand nombre d'éléments discrets en indiquant simplement s'ils sont occupés ou libres : blocs/pages mémoire, identifiants, files de requêtes, clusters, etc. Pour répondre aux exigences d'efficacité que suppose une telle approche, Windows recourt à un mécanisme compact dit cartes binaires (*bitmaps*).

Un bitmap, sous sa forme la plus générale, est une séquence dans laquelle chaque bit individuel reflète l'état d'un élément donné : un bit armé signifie que l'élément correspondant est occupé, en cours d'utilisation, ou tout autre état approprié selon le contexte, tandis qu'un bit à 0 indique que l'élément est libre ou inactif, et se trouve par conséquent disponible.

```
typedef struct _RTL_BITMAP {
  ULONG SizeOfBitMap;
  PULONG Buffer;
} RTL_BITMAP, *PRTL_BITMAP;
```

Parmi cette définition, l'attribut `SizeOfBitMap` totalise le nombre de bits que le bitmap représente, et `Buffer` mémorise un pointeur vers la zone mémoire dans laquelle sont ces bits sont effectivement enregistrés.

Fonction	Description
RtlAreBitsClear	Vérifie que tous les bits dans un intervalle spécifié d'un bitmap sont à 0
RtlAreBitsSet	Vérifie que tous les bits dans un intervalle spécifié d'un bitmap sont à 1
RtlClearAllBits	Réinitialise l'intégralité des bits d'un bitmap à 0
RtlClearBit	Réinitialise à 0 un bit donné dans un bitmap
RtlFindClearBits	Cherche une séquence contiguë de bits à 0 dans un bitmap
RtlFindSetBits	Cherche une séquence contiguë de bits à 1 dans un bitmap
RtlInitializeBitMap	Initialise une structure `RTL_BITMAP` afin de la rendre prête à l'emploi auprès des fonctions subséquentes
RtlNumberOfClearBits	Renvoie le nombre total de bits à 0 dans un bitmap
RtlNumberOfSetBits	Renvoie le nombre total de bits à 1 dans un bitmap
RtlSetAllBits	Arme l'intégralité des bits d'un bitmap
RtlSetBit	Arme parmi un bitmap un bit donné
RtlTestBit	Teste l'état (0 ou 1) d'un bit donné

Chapitre 3. Architecture du système

Est esquissée en filigrane de ce chapitre l'architecture générale des systèmes d'exploitation de la gamme Microsoft Windows. Après une vue d'ensemble sur le sujet, nous cheminerons parmi les objectifs posés par Microsoft lors de la conception du logiciel, avec dans ce contexte les définitions des besoins basiques ou fondamentaux ayant servi de cadre guide aux spécifications de Windows NT dès 1989. Vous verrez par ce biais comment la construction du système a intégré avec brio divers principes et, sur un plan plus actuel, de quelle façon les honorent les versions modernes de Windows. Nous montrerons les grandes lignes de la structure interne du système : les composants clé, leurs interactions mutuelles et leurs raisons d'être.

Souhaitant éveiller chez le lecteur un sentiment d'ordre général, notez que ce chapitre privilégie une vue globale du thème discuté, sans excès de précisions ou trop-plein de détails techniques. L'ensemble est pour cette raison jalonné de liens vers les contenus plus en profondeur de ce livre, eux-mêmes menant à une multitude d'observations directes du système. Cela étant dit, commençons notre exploration par une étude du modèle de conception général de Windows.

Architecture générale de Microsoft Windows

En dépit de la diversité des méthodes pour les envisager (conception) et les réaliser (implémentation), tous les produits et systèmes informatiques se classent parmi un nombre extrêmement restreint de styles architecturaux. (L'architecture logicielle à l'instar de l'architecture traditionnelle peut se catégoriser en styles.) Nous verrons ainsi dans cette section le style architectural avec lequel Windows a été élaboré, émergeant à cet égard comme réponse à un ensemble d'impératifs divers, autant technique que stratégique. Nous indiquerons en quoi l'approche choisie surpasse ou s'avère plus appropriée que les autres pour atteindre les objectifs fixés, et verrons comment elle marque son empreinte sur des facteurs importants de la dynamique du système d'exploitation.

Fichiers système fondamentaux

La liste suivante met en évidence quels sont les noms de fichiers associés aux composants fondamentaux de Windows. Chacun de ces composants sera traité en détail dans ce chapitre ou dans ceux qui suivent.

- *Hal.dll* Couche d'abstraction matérielle.
- *Ntoskrnl.exe* Exécutif et noyau pour les systèmes mono processeur.
- *Ntkrnlmp.exe* Exécutif et noyau pour les systèmes multi processeur.
- *Ntkrnlpa.exe* Exécutif et noyau avec prise en charge de L'extension d'adresse physique (PAE, *Physical Address Extension*) ; systèmes 32 bits uniquement.
- *Kernel32.dll* DLL fondamentale du sous-système Windows.
- *Ntdll.dll* Fonctions de support internes et et mécanismes de diffusion vers les services système.
- *Winlogon.exe* Ouverture de session.
- *Csrss.exe* Processus exécutant le sous-système Windows.
- *Gdi32.dll* Fonctions liées à l'interface graphique.
- *User32.dll* Fonctions liées à l'interface utilisateur.
- *Services.exe* Processus exécutant le contrôleur de services.
- *Smss.exe* Sous-système gestionnaire de session.

- *Win32k.sys* Partie mode noyau du sous-système Windows.

Aperçu de l'architecture

Empruntant en la matière aux fonctionnalités de gestion des plateformes matérielles modernes, l'architecture de Windows s'articule autour de deux grands axes, et consiste ainsi donc de deux couches principales, l'une hébergeant les fonctions vitales du système d'exploitation (mode noyau), l'autre les programmes soumis à la supervision de ce dernier (mode utilisateur).

Comme mentionné au chapitre 1, les threads mode utilisateur sont exécutées dans un espace d'adressage de processus protégé, lequel les limite quant aux ressources auxquelles ils ont accès. Voici à quoi correspondent les quatre types fondamentaux de processus en mode utilisateur :

- *Processus de support système* Les processus de support système offrent l'infrastructure pour le soutien des utilisateurs, de leurs données et de leurs programmes. Dès que Windows a fini de démarrer, un certain nombre de ces processus, tels le processus d'ouverture de session et le gestionnaire de session, sont actif, cela afin que le système d'exploitation puisse fonctionner et l'utilisateur interagir avec lui. (Notez que ces processus ne sont pas des services Windows, au sens qu'ils ils ne reçoivent pas de requêtes du gestionnaire de contrôle des services, mais sont démarrés par le système lui-même.)

- *Applications utilisateur* Catalogue auquel s'intègrent les programmes des utilisateurs mais aussi divers programmes appartenant au système d'exploitation, les applications utilisateur s'exécutent en utilisant les services du système d'exploitation pour utiliser les ressources matérielles sous-jacentes. Bloc-notes (notepad.exe), célèbre éditeur de texte intégré à Windows, est un exemple d''application utilisateur ; Microsoft Word (winword.exe), Microsoft Excel (excel.exe), d'autres exemples. Windows divise les applications utilisateur en plusieurs catégories, cela en lien avec les fonctionnalités d'environnement du système d'exploitation : Windows 64 bits, Windows 32 bits, Windows 16 bits, MS-DOS 16 bits, POSIX 32 bits et OS/2 32 bits. (Notez que cette liste comprend des usages qui ne sont pas nécessairement couverts dans les versions modernes de Windows.)

- *Processus de service* Les processus de service, analogues des démons d'UNIX, abritent des services Windows, un genre particulier de processus servant généralement à l'incorporation (démarrage, pause et arrêt) de taches d'arrière-plan, ou qui doivent pouvoir être exécutées indépendamment des ouvertures de session des utilisateurs (non liés à l'utilisateur interactif donc). Le planificateur de taches, le spouleur d'impression, et bien d'autres logiciels sont mis en oeuvre en tant que services, ou contiennent des fonctionnalités exécutées à l'intérieur de tels entités.

- *Processus serveur de sous-système d'environnement* Implémentent une partie (l'autre partie étant constitué de services système en mode noyau) des fonctionnalités d'environnement du système, lesquelles donnent aux programmes des utilisateurs un contexte d'exécution adéquat (une application Windows 64 bits ne s'exécute pas de la même façon qu'une application Windows 32 bits, et encore moins qu'une application basée MS-DOS fondée sur du texte), met entre les mains des concepteurs d'application un ensemble de méthodes et de modèles à programmer, et donne au système d'exploitation sa personnalité graphique. Pour plus d'informations, consultez la section sous-systèmes d'environnement de ce chapitre.

Au contraire de la plupart des systèmes d'exploitation, qui donnent une proximité immédiate aux fonctions primitives fournies par le noyau et utilisées par les programmes s'exécutant dans l'espace utilisateur (appels système), les applications utilisateur sous Windows ne sollicitent pas directement les services natifs du système ; elles passent plutôt une ou plusieurs bibliothèques de liens dynamiques (DLL, Dynamic Link Library) qui, apparentées à un processus serveur de sous-système d'environnement, délivrent l'interface de programmation de ce système. Le rôle de ces DLL est de traduire l'appel d'une fonction en une requête vers le service système interne appropriée de Windows. Par exemple, les fonctions internes du processus du sous-système Windows sont rendues visibles aux programmes par des interfaces de programmation mises en œuvre par des fichiers DLL. Les trois principales bibliothèques sont User32.dll (manipulation de l'interface utilisateur), GDI32.dll (manipulation des dispositifs d'impression et d'affichage), et Kernel32.dll (utilisation des services concernant les processus et les threads, les fichiers, et autres).

Voici à quoi correspondent les quelques composants logés dans la portion système de Windows, exécutés en mode noyau :

- *Exécutif* L'exécutif Windows met en œuvre les services système fondamentaux, tels la gestion de la mémoire, la gestion des processus et des threads, la sécurité et les E/S.
- *Noyau* Le noyau Windows contient les fonctions système de bas, par exemple l'ordonnancement des threads, la gestion et la répartition des interceptions (trappes, interruptions et exceptions), et la synchronisation multi processeur. Il fournit également un ensemble de routines basiques et de structures de données élémentaires que les autres modules de l'exécutif peuvent invoquer.
- *Couche d'abstraction matérielle* La couche d'abstraction matérielle isole le noyau, les pilotes de périphérique et les services de l'exécutif des spécificités de la plateforme.
- *Pilotes de périphérique* Les pilotes de périphérique permettent aux applications de disposer d'une interface vers le système d'entrée/sortie de Windows.
- *Système de fenêtrage et de graphisme* Le système de fenêtrage et de graphisme réalise les opérations liées aux dessins des interfaces utilisateur graphique.

Approche en couches

Windows adopte un mode de fonctionnement orienté selon deux grandes lignes de force : l'approche par couche, dans laquelle le système d'exploitation comprend un certain nombre de divisions, et l'orientation objet, où chaque entité du système informatique, physique ou virtuelle, est représentée comme une combinaison de propriétés, de caractéristiques et d'états dans le système.

La division en couches consiste à regrouper les composants possédant une sémantique semblable (ou au moins analogue) de manière à créer un empilement de paquetages de composants ; les composants des couches supérieures dépendants fonctionnellement des composants des couches inférieures. Chaque couche dans cette perspective se superpose à une autre en apportant à l'ensemble de nouvelles fonctions plus élaborées, et reposant sur les fonctions plus élémentaires assurées par les couches sous-jacentes. La couche la plus basse d'un système d'exploitation est le matériel ; la plus haute, par conséquent la plus éloignée des rouages et des procédures bas niveau du système, est l'interface utilisateur.

Une couche est l'implémentation d'un objet abstrait qui encapsule des données et des opérations relatives à l'interaction avec ces données. Dans Windows, toutes les ressources partageables sont représentées par des objets : les processus, les threads, Les fichiers, les dispositifs physiques, les pilotes, et bien d'autres.

Le principal bénéfice à l'utilisation de l'approche en couches est la modularité, définie en génie logiciel comme le fait, pour un programme, d'être écrit en plusieurs parties relativement indépendantes les unes des autres. Les couches sont sélectionnées de telle sorte que chacune utilise seulement les dispositifs de programmation fournies par les couches de niveau inférieure. Ainsi, chaque couche peut individuellement être vérifiée. Quand la première couche fonctionne correctement, la seconde peut être inspectée, et ainsi de suite. Si une erreur est trouvée au cours du débogage d'une couche, il est légitime de penser que la faute est imputable, à cette couche, et uniquement à elle. (Au-delà du fait que probable ne veut pas dire certain, la complexité d'un système tel Windows peut mener dans l'analyse à de fausses conclusions.) En découpant le système en couches, on facilite par ce biais sa conception, sa réalisation, sa validation et sa maintenance.

L'abstraction est un autre avantage que procure l'approche en couches, cela dans la mesure où toutes se séparent de façon indépendante les unes des autres. Par définition, les opérations implémentés dans une couche le sont en faisant abstraction de l'implémentation de ces opérations dans les couches inférieures. Les éléments d'une couche, des objets, n'ont aucune connaissance de comment sont constitués les autres, et c'est uniquement à travers des interfaces bien définies et un modèle sémantique affirmé que les objets ont la possibilité d'agir entre eux. Chaque couche masque donc l'existence de certaines structures de données, d'opérations et de matériels aux couches supérieures.

Objectifs de conception

Un projet de l'envergure de Microsoft Windows revêt de multiples dimensions et couvre de nombreuses exigences de différents caractères. (Notez, sur la nature et la teneur des difficultés à satisfaire pléthore de contraintes, que ce n'est pas tant la multiplicité des composants, mais la diversité de leurs interrelations, qui caractérisent le plus la complexité, d'autant plus que certaines exigences sont antagonistes. Le résultat final, autrement dit le logiciel tel que nous le connaissons, est par conséquent un compromis entre des conditions générales commandées par les circonstances.) Afin que l'éventail des moyens résultants puisse être intégré de façon harmonieuse, les concepteurs de Windows NT adoptèrent les objectifs de conception suivants :

- *Compatibilité* Le système devait être capable d'exécuter les applications existantes développées pour les anciennes versions de Windows (gamme 16 bits) et pour MS-DOS. Il devait aussi pouvoir interagir avec d'autres systèmes comme UNIX ou OS/2.

- *Extensibilité* Le système devait pouvoir s'adapter en douceur aux évolutions matérielles et aux évolutions des besoins.

- *Fiabilité et robustesse* Le système devait savoir se protéger contre les dysfonctionnements internes et contre les altérations externes. Les applications ne devaient pas pouvoir nuire au système d'exploitation ni aux autres applications.

- *Performance* Le système devait répondre rapidement aux tâches demandées, être aussi rapide et réactif que possible sur chaque plateforme matérielle.

- *Portabilité* Le système devait pouvoir fonctionner sur différentes architectures et plateformes matérielles. Le support d'une nouvelle architecture devrait pouvoir être intégré sans effort rédhibitoire.

Fiabilité et robustesse

Premier éditeur mondial de logiciels et de solutions d'entreprise, Microsoft véhicule avec ses gammes de systèmes d'exploitation des années d'histoire et de progrès technologique (Windows 1.0, sortie en 1985, est l'aboutissement de quatre années de développement interne.) Cette longévité remarquable s'explique, en partie au moins, par l'enracinement dans Windows des objectifs et qualités requises en terme de fiabilité et de robustesse.

- *Protection contre les applications problématiques* Windows exploite une architecture mémoire autorisant une protection complète des divers codes s'exécutant sur le processeur. Le code du système d'exploitation est exécuté dans un mode privilégié du processeur (mode noyau), avec accès complet aux données, aux instructions et au matériel. Le code des applications est exécuté dans un mode non privilégié (mode utilisateur), avec accès limité au code et aux données de la partie privilégiée. Cette protection est l'une des raisons pour lesquelles les composants de Windows sont protégés contre les applications, et les applications contre les agissements potentiellement nuisibles (que ces derniers soient intentionnels ou accidentels) d'autres applications.

- *Restauration système* La fonctionnalité Restauration du système permet à un administrateur de poste d'enregistrer diverses informations cruciales pour Windows, essentielles à la bonne marche du système informatique. Un mécanisme de points de restauration permet de maintenir à jour les éléments récupérés. A l'installation de certains composants, ou autres changements de la configuration, Windows crée un point de restauration pour, en cas de problèmes, être en mesure de revenir à l'état initial du système avant prise en compte des modifications.

- *Maturité du code source* Riches de pratiques et de savoirs faire en constante amélioration, les diverses versions de Windows fournissent l'assurance d'une qualité logicielle maitrisée et régulière. Avec en parallèle de l'évolution du système d'exploitation la professionnalisation des métiers du test informatique, et partant l'industrialisation de leurs processus, Microsoft utilise une revue intensive du code, l'objectif étant, sur la base

de vérifications automatiques et manuelles, d'identifier les fichiers sources pouvant contenir des problèmes, cela afin d'améliorer la fiabilité (qualité et sécurité) du logiciel.

- *Capacités de résistance* Windows est soumis à des essais extensifs en condition de stress (stress testing), destinés à évaluer la robustesse du système dans des conditions de fonctionnement sévères ou inhabituelles (ressources mémoire venant à manquer, espace disque insuffisant, conditions de concurrence anormalement élevées, etc.). Le système a été testé dans les conditions d'usage les plus difficiles et montré ses capacités de résistance.

- *Système de fichiers NTFS* Si quantité de biens numériques lui sont confiés (programmes et données personnelles de l'utilisateur, processus et logiciels métiers, et bien d'autres), Windows le doit en ce qu'il s'appuie sur un système de fichiers robuste et moderne, NTFS (NT File System), lequel inclut en standard des fonctions relatives à la protection et à la sécurité des données. Dans la perspective NTFS, toutes les informations d'opérations exécutées sur le disque sont enregistrées dans un fichier journal. En cas de problème, NTFS utilise ce journal pour restaurer l'unité en panne. De plus, NTFS n'enregistre pas une quelconque action avant de s'être assuré que celle-ci s'est correctement déroulée (principe des transactions), cela en vue de garantir l'intégrité des données en cas d'arrêt brutal du système d'exploitation (coupure d'alimentation, effondrement du système, etc.).

- *Fiabilité perçue de l'interface graphique* utilisateur Windows étant du point de vue de l'utilisateur un système d'exploitation essentiellement graphique, la fiabilité perçue du logiciel, estimée selon les critères de l'utilisabilité (voir note plus bas) est également améliorée en rendant l'interface utilisateur plus facile à utiliser de par une meilleure ergonomie, des menus plus simples et divers perfectionnements dans la découverte intuitive de comment effectuer les tâches courantes. Exemple de fiabilité perçue, l'environnement personnalisé : Windows tient compte des habitudes de l'utilisateur pour construire dynamiquement une interface graphique façonnée selon l'activité antérieure; ainsi, les applications les plus utilisées sont mises en valeur, les autres objets mis au second plan.

- *Redémarrage automatique des composants en erreur* Windows intègre la capacité de relancer automatiquement les composants du système d'exploitation qui auraient cessé de fonctionner. De plus, le système consigne dans plusieurs emplacements du système de fichiers (journaux, Registre, etc.) les raisons pour lesquelles un composant est défectueux non conforme à une commande donnée, ou susceptible de provoquer une situation non attendue. Divers utilitaires accompagnent cette démarche, par exemple l'Utilitaire de résolution des problèmes, qui recherche les problèmes courants et vérifie que tous les nouveaux périphériques et matériels connectés à l'ordinateur se comportent correctement.

- *Vérificateur de pilotes (et consorts)* Les systèmes Microsoft intègrent une gamme d'outils, standards ou optionnels, via laquelle améliorer la stabilité du système d'exploitation et déterminer la qualité des composants sur le plan des fonctionnalités. À titre d'exemple, les utilisateurs (les plus techniques, concédons-le) peuvent utiliser la fonctionnalité d'évaluation des pilotes pour vérifier qu'un système d'exploitation Windows en cours d'exécution contient le bon ensemble de pilotes, et par voie de conséquence repérer ceux potentiellement à problème. Un administrateur sera de cette façon capable d'identifier précisément et immédiatement les pilotes susceptibles provoquer l'instabilité de l'ordinateur, au lieu de devoir se contenter d'indices éventuellement trompeurs obtenus après-coup l'effondrement du système. (Le dysfonctionnement d'un composant en mode noyau de Windows est la source la plus répandue de défaillance système.) Dans un autre registre, mais toujours au chapitre de la fiabilité et de la robustesse du système d'exploitation, divers utilitaires sont incorporés à Windows dans le but de surveiller la bonne marche des processus et des services (Gestionnaire de tâches), de juger de la disponibilité des ressources (Moniteur de ressources), d'examiner la manière dont l'exécution des programmes affecte les performances de l'ordinateur (Moniteur de fiabilité et de performances), et bien d'autres usages.

- *Protection des fichiers Windows* Les fichiers système critiques de Windows sont protégés en étant sauvegardés automatiquement ; lorsqu'un tel fichier est modifié ou est supprimé accidentellement, Windows le remplace par une copie valide. La protection de ces fichiers permet d'éviter des problèmes au niveau des programmes et du système d'exploitation. Le chapitre Mécanismes système revient plus en détail sur la fonctionnalité Protection des fichiers Windows.

Haute performance

Importante, déjà, en ce qui concerne des logiciels de moindre ampleur, la question de la performance d'un système informatique est centrale quant à son adoption et à son utilité. Microsoft Windows fut ainsi conçu pour offrir de hautes performances pour les systèmes de bureau, les systèmes serveur, terminaux mobiles, et pour les grands environnements multithreadés et multiprocesseurs.

Notez, sur le plan de la langue, que le terme *performance* n'a pas la même connotation compétitive en anglais qu'en français. Au sein d'un contexte francophone, la notion se réfère ainsi le plus souvent à une représentation subjective de la réussite, ou du moins aux voies d'accès à celle-ci. En anglais, le terme performance peut désigner l'action elle-même, une acceptation qui renvoie donc dans ce cas à un processus, sans forcement être synonyme de bons résultats.

La notion de performance apparaît de façon explicite dans de nombreux procédés mis en avant par Windows :

- *Entrées/sorties asynchrones* Les E/S asynchrones permettent à une application d'émettre une requête d'E/S, puis de continuer à travailler pendant que le périphérique effectue les traitements impliqués (moyennant quoi le thread applicatif doit synchroniser son exécution avec l'achèvement de la requête).

- *Mise en cache sophistiquée de données du système de fichiers* Le gestionnaire de cache de Windows utilise une méthode basée sur les blocs virtuels (par opposition à un cache basé sur des blocs logiques), qui permet des lectures anticipées intelligentes (il peut prédire l'endroit où l'appelant risque de faire la lecture suivante) et des accès haut débit au cache.

- *Mise en lot des requêtes* Le cache Windows fait de la régulation des écritures : on laisse un temps s'accumuler dans le cache les modifications adressant l'intérieur de fichiers (on met autrement dit en lot les E/S sous-jacentes) avant de les écrire toutes en une seule fois sur le disque. Cette gestion rend possible de s'abstraire des limites et contraintes inhérentes au matériel disque, et favorise la réactivité d'ensemble de l'environnement.

- *Considérations matérielles* Windows emploie pour ses techniques de gestion mémoire et mémoire cache le principe de localité (utilisation des instructions et données situées dans la zone mémoire proche des données et instructions accédées récemment, localité spatiale ; réutilisation des instructions et donnes utilisées dans le passé, localité temporelle). L'algorithme afférent tient compte, et s'adapte en fonction, des spécificités matérielles impliquées dans ce principe (taille de page, taille des lignes de cache, etc.). Le code de gestion des interceptions profite s'il est présent du dispositif appel système rapide, alternative moins coûteuse que les interruptions dans le domaine. En matière de verrouillage et synchronisation, les codes d'acquisition et de libération sont optimisés en langage bas niveau, à la fois pour des raisons de vitesse (performance) et pour pouvoir bénéficier des installations offertes par l'architecture processeur sous-jacente. Ce sont là divers exemples montrant comment, à partir de considérations faites sur le matériels, Windows développe, enrichit et rend optimal ses stratégies d'exploitation.

- *Protocoles de verrouillage extensibles et à faible granularité* Les mécanismes de synchronisation offerts par le noyau sont implémentés sous forme de primitives extensibles. Les besoins en la matière étant multiples, plusieurs jeux de fonctions coexistent, conscients qu'à chaque situation de synchronisation correspond une réponse appropriée. Par exemple, là où des spinlocks ordinaires conviendraient peu (le terme spinlock vient du fait que le noyau « boucle » (spins) jusqu'à ce qu'il ait le verrou), vous pourriez utiliser des spinlocks en file (qui offre une meilleure évolutivité) ; et s'ils ne convenaient pas du tout des pushlocks, des mutex, des ressources éxécutif, etc.

- *Système de priorités adaptatives* Windows emploie pour l'ordonnancement des tâches un modèle piloté par priorités : les threads peuvent être préemptés par d'autres threads de priorité supérieure. Le système peut en conséquence répondre rapidement à des événements extérieurs, favoriser ou au contraire défavoriser certains éléments de l'exécution.

- *Optimisation* Quand bien même C et C++ restent favoris partout ailleurs, parce que gage de portabilité, c'est l'assembleur qui est utilisé lors de procédures sensibles en matière de performances. On trouve de

l'assembleur dans le code de gestion des appels système, dans la réalisation de certains protocoles de verrouillage, et même dans certaines bibliothèques mode utilisateur.

- *Mécanisme LPC* LPC (Local Procedure Call) est une technologie de communication inter processus pour transmission rapide de messages entre processus d'un même environnement (sur le même ordinateur). Sollicité intensivement dans la communication vers et depuis les sous-systèmes internes de Windows, les mécanismes LPC font beaucoup pour la réactivité du système, et s'utilisent dans des endroits des plus critiques, tels que par exemple le processus serveur d'authentification de sécurité locale, le gestionnaire de session ou celui des services.

- *Graphiques gérés par le noyau* Avant Windows NT 4, routines d'interface utilisateur et code graphique (gestionnaire de fenêtres et services de graphisme donc) étaient exécutés dans le contexte d'un processus mode utilisateur. Par la suite, l'essentiel de ce que réalisaient ces éléments fut déplacé dans le noyau (dans le fichier Win32k.sys). On réduisait ce faisant le nombre de basculements de contexte vers un processus serveur séparé, opération coûteuse en cycles processeur et ressources mémoire, et améliorait alors les performances globales du système.

- *DirectX* Collection de bibliothèques destinées à la programmation d'applications multimédia sur les plates-formes Microsoft (Xbox, Windows), DirectX offre de hautes performances graphiques pour les ordinateurs personnels. La technologie DirectX permet un gain de rapidité en accédant directement au calculateur 3D des cartes graphiques modernes, et à toutes les versions de Windows depuis Windows 95 de bénéficier de capacités multimédia performantes.

- *SuperFetch* Destinée à améliorer les performances de démarrage et le temps de réponse des applications, la fonction SuperFetch permet, selon une analyse des activités passées de l'ordinateur, de pré charger en mémoire les composants les plus sollicités, donc les plus susceptibles de l'être encore. Par exemple, dans le cas d'une utilisation bureautique poussée, SuperFetch détectera un recours intensif à Microsoft Word et Microsoft Excel (en supposant bien sûr une préférence pour ces produits) et chargera automatiquement en mémoire les divers modules relatifs aux processus exécutant ces programmes.

Internationalisation

Conçu pour une utilisation sur le territoire mondial, où de multiples langues, cultures et singularités coexistent, Windows définit compte tenu de ces particularités divers mécanismes pour la prise en compte de l'internationalisation et de la régionalisation, deux parties d'un même ensemble visant à permettre à une application de présenter son contenu dans des langues et des formats adaptés à ses utilisateurs. (Pour plus d'informations sur ces sujets, voir encadré plus loin.)

Parmi les mesures grâce auxquelles Windows s'adapte facilement à différentes langues et cultures :

- *Support des Langues Nationales* L'API de support des langues nationales (NLS, National Language Support) héberge divers services internationalisés pour ce qui est de la langue, de la culture et des conventions écrites. L'architecture NLS permet de sauvegarder et manipuler les données en fonction de paramètres de langue préconfigurés. Elle assure que les programmes, messages d'erreur, ordre de tri, date, heure, monétaire, numérique, et calendrier s'adaptent automatiquement aux paramètres préconfigurés de la langue et locale. Elle fournit également le support pour les dispositions de clavier (*keyboard layouts*) et des polices spécifiques à une langue.

- *Unicode* Pour répondre à l'hétérogénéité des différentes représentations de données, Windows utilise Unicode, un format spécial permettant des échanges de textes dans différentes langues, à un niveau mondial. Les applications Windows sont totalement compatibles Unicode, ce qui signifie que chaque caractère est représenté par un nombre unique, indépendamment de la plateforme, du programme ou de la langue. Codage de caractères natif de Windows, Unicode est l'un des moyens par lesquels le système a pu s'exporter facilement, et s'imposer sur le marché international. Pour de plus amples informations sur Unicode, consultez la section Unicode (chapitre 1).

- *Fichiers ressources* Les chaines de texte système, plutôt que d'être déclarées de façon statique (autrement dit codées en dur), sont conservées dans des fichiers ressources qui peuvent être remplacés de sorte à adapter l'environnement à différents langages. Un certain nombre de variables liées au pays ou à la culture d'appartenance sont établies relativement à ces aspects, cela afin de mieux correspondre aux préférences des individus ou groupes d'individus multilingues. .Internationalisation et régionalisation

Les définitions en ce qui concerne l'internationalisation et la régionalisation varient. Ici, nous offrons quelques descriptions de niveau général sur la façon dont ce livre a tendance à utiliser ces termes.

L'internationalisation (souvent abrégé en i18n - la lettre initiale i du mot internationalization suivie du nombre de lettres intermédiaires et le tout terminé par un n final) est un terme général faisant référence au fait de préparer un logiciel afin qu'il puisse s'adapter à des langues et à des cultures différentes. Une des techniques de base de l'internationalisation consiste à séparer, dans le code source d'un programme, ce qui est indépendant de la langue et de la culture de ce qui en est dépendant ; cela, généralement, dans des fichiers séparés, qui pourront alors être traduits sans devoir modifier le code de ce programme.

La régionalisation est l'adaptation d'un logiciel à destination d'une culture, ou d'une langue particulière. Le terme localisation (l10n), transposition du faux ami anglais localization, est souvent utilisé. La régionalisation implique principalement la traduction de l'interface utilisateur d'un logiciel , mais elle ne se limite pas à cet aspect, puisque l'adaptation peut également concerner les attentes culturelles et linguistiques propres à la langue et au pays de l'utilisateur, par exemple la représentation des chiffres, le type de virgule, l'utilisation judicieuse de couleurs et de symboles adaptés à la culture cible, les différentes lois et réglementations dans le pays visé, etc. avant qu'un logiciel ne soit régionalisé, il faut qu'il ait été internationalisé, c'est-à-dire qu'il ait été écrit pour pouvoir être traduit. A cet égard, plus L'internationalisation est bien conçue, plus la régionalisation est techniquement facile à effectuer. Le consortium Unicode travaille sur une normalisation de ces paramètres régionaux, le projet Common Locale Data Repository.

Fontions	**Services**
GetSystemDefaultLangID	
GetSystemDefaultLCID	NtQueryDefaultLocale
GetSystemDefaultUILanguage	NtQueryInstallUILanguage
GetThreadLocale	
GetUserDefaultLangID	
GetUserDefaultLCID	NtQueryDefaultLocale
GetUserDefaultUILanguage	NtQueryDefaultUILanguage
GetUserGeoID	NtQueryValueKey
IsValidLocale	
IsValidLocaleName	
SetThreadLocale	
SetThreadUILanguage	
SetUserGeoID	NtSetValueKey

Attributs	**Structures**	**Autres références**
CurrentLocale	TEB	GetThreadLocale, SetThreadLocale

Extensibilité

L'extensibilité fait référence à la capacité d'un système d'exploitation à suivre les avancées technologiques, tant en matière de dispositifs physiques que relativement aux infrastructures et solutions logicielles. Les mesures influentes en ce domaine visent, sur le plan structurel, à permettre l'introduction facile de nouvelles fonctions et l'évolution en douceur des existantes. Celles-ci incluent :

- *Structure modulaire* Afin que les changements futurs soient facilités, Windows a été conçu selon des principes modulaires qui ouvrent la voie à la greffe de nouvelles fonctions et au remplacement de certaines par

d'autres plus évoluées. L'exécutif Windows fonctionne en mode noyau ou protégé et offre les services système fondamentaux, par exemple la gestion des processus (gestionnaire de processus) ou l'ordonnancement des threads (noyau). Chaque composant au sein de la couche exécutive communique avec un ou plusieurs de ses homologues via les interfaces de programmation adéquates. Au-dessus de la couche exécutive, plusieurs sous-systèmes serveur s'exécutent en mode utilisateur. Parmi eux se trouvent les sous-systèmes d'environnement, lesquels agissent comme support pour les programmes Windows et les programmes écrits pour d'autres systèmes d'exploitation, tels MS-DOS ou UNIX. Grâce à cette structure modulaire, de nouvelles surfaces autour desquelles concevoir des supports additionnels peuvent par ce biais être intégrées sans affecter la couche exécutive.

- *Pilotes chargeables dans le système d'E/S* Windows met en oeuvre pour la prise en charge des pilotes un mécanisme de modules chargeables dans le système d'E/S. Orientés par une approche souple de l'ajout de matériel, les pilotes de périphérique sous Windows ne manipulent pas directement le matériel, mais sollicitent la couche d'abstraction matérielle pour faire l'interface avec le matériel. De nouveaux périphériques, tels que claviers et souris, et pilotes, tels pilotes de système de fichiers, peuvent ainsi être ajoutés.

- *Exécution distribuée* Windows supporte l'exécution distribuée au moyen d'appels de procédures distantes (RPC, Remote Procédure Call). Le protocole RPC permet d'appeler des fonctions sur un système distant de façon quasi transparente, comme s'il s'agissait d'appels locaux, et est utilisé dans de nombreux services Windows. On trouve par exemple du code RPC dans les processus gérant le Registre, les services, les mécanismes de sécurité locale et d'authentification des utilisateurs (LSA), etc.

- *Gestion en réseau* Windows intègre de façon native de nombreuses fonctionnalités réseau, permettant à un ensemble d'équipements (ordinateurs et périphériques tiers) reliés entre eux afin d'échanger des informations. Le système supporte un grand nombre de normes de câblage correspondant à des technologies différentes (ethernet, liaison sans fil, modem...), reconnait autant de protocoles réseau (NetBEUI, TCP/IP, Appletalk, TokenRing...), et rend accessible pléthore de services réseau (SMB, DNS, NTP, DHCP, Wins...). L'ensemble de ces intermédiaires donne à Windows un caractère qui offre une grande interopérabilité dans le cadre de réseaux hétérogènes à tout niveau.

Compatibilité

Si chaque nouvelle mouture de Windows apporte son lot de fonctionnalités nouvelles et d'améliorations envers de futures intégrations plus satisfaisantes, Microsoft a toujours été salué pour son attention aux problèmes de compatibilité, fréquents dans le domaine de l'informatique, qui connaît une évolution rapide du matériel et des logiciels. Les méthodes grâce auxquelles les déclinaisons les plus récentes de Windows sont compatibles avec les anciennes versions sont majoritairement les suivantes :

- *Profils système* Windows intègre une architecture modulaire et extensible qui offre toute la souplesse requise pour s'adapter à différents profils de système d'exploitation. Il en résulte la prise en charge d'applications dont le fonctionnement se trouve réglé par des dispositions d'origine et de nature diverses, que celles-ci émanent directement d'un autre système d'exploitation, par exemple MS-DOS, sur lequel nous reviendrons plus tard, OS/2, ou de normes techniques prédéfinies, par exemple POSIX. Cela permet permet d'effectuer des améliorations sur un profil système sans affecter la compatibilité avec les applications des autres profils.

- *Mode de compatibilité* Pour obtenir une meilleure compatibilité avec les anciennes versions de Windows, les systèmes Microsoft permettent aux utilisateurs de spécifier les applications devant être prises en charge selon un mode de fonctionnement spécifique, appelé mode de compatibilité, se rapportant aux paramètres issus d'une version précédente de Windows et, sur un plan plus global, aux conditions délivrées par une version individuelle de Windows lors de l'exécution des programmes. En interne, Windows introduit pour ce faire une couche intermédiaire qui modifie les API Windows pour mieux approximer le comportement attendu par les anciennes applications. Ce composant peut, par exemple, faire paraître Windows 8 compatible fonctionnalités pour fonctionnalités avec Windows XP, et donner le change à certaines applications qui s'attendent (souvent sans fondements) à voir une certaine version de Windows, comptent sur des bizarreries d'implémentation relatives aux API, mésestiment les traitements internes effectués à l'intérieur des services natifs, ou plus simplement font des assertions erronées quant à l'incidence qu'ont certaines routines et variables noyau. En

maintenant un support pour toutes les applications - y compris celles contenant des erreurs de programmation latents qui deviennent apparents à cause des changements dans l'implementation - Microsoft parvient à conserver toujours sur son écosystème un paysage riche des logiciels des versions précédentes.

- *Assistant Compatibilité des programmes* La plupart des programmes conçus pour des versions antérieures de Windows sont compatibles avec la nouvelle version (voir point précédent). Toutefois, après mise à niveau vers une version de Windows plus récente que celle pour laquelle ils ont été conçus, il se peut que certains programmes réagissent de manière erratique, effectuent des actions inappropriées, voire ne fonctionnent plus du tout. Pour ces cas, Windows utilise l'assistant Compatibilité des programmes pour procéder automatiquement à des modifications associées à des problèmes de compatibilité connus. Si l'assistant détecte un problème de compatibilité connu, il signale le problème et fournit des solutions possibles pour le résoudre. Vous pouvez alors autoriser l'assistant à reconfigurer automatiquement l'application, ou modifier vous-mêmes les réglages afférents.

- *Maturité du code* La maturité du code rend pérenne les interfaces exposées dans le noyau, les services de l'exécutif et les bibliothèques de sous-système (les routines normalement invisibles aux autres composants n'ont de toute façon aucun impératif à ce niveau, puisqu'elles ne devraient normalement jamais être appelé par des composants tiers). Depuis la rédaction du cahier des charges de Windows NT jusqu'à nos jours, les concepteurs d'applications et de pilotes tiers sont en face globalement des mêmes spécifications. Seuls quelques rares prototypes de fonctions ont été revus, notamment pour l'amélioration du multitraitement symétrique (multi processeur), ou pour l'infrastructure de soutien du 64 bits. Lors du passage d'une architecture 32 bits à une nouvelle 64 bits, des types de données additionnels intégrèrent les diverses interfaces Windows, afin de supporter ce nouveau type d'adressage, et le firent de manière assez délicate : en s'adaptant à une architecture cible, 32 ou 64 bits. Ainsi, une valeur ULONG_PTR est une valeur 32 bits quand elle est manipulée par un compilateur 32 bits, une valeur 64 bits quand elle vue par un outil idoine 64 bits. À titre indicatif, vous pouvez être certain que lorsque vous croisez un type PTR dans une déclaration de fonction, il s'agit d'une manœuvre pour permettre aux développeurs de concevoir du code portable et compatible.

- *Compatibilité des pilotes* Outre la réduction des problèmes de compatibilité des programmes utilisateur, Windows tend à diminuer également ceux relatifs aux pilotes. Ainsi, chaque modèle de conception de pilotes livre avec Windows définit en standard des méthodes compatibles entre plusieurs versions du système d'exploitation. De plus, Les pilotes conformes à ces règles sont conçus pour être compatibles « vers le haut » (compatibilité ascendante), de sorte qu'un pilote peut fonctionner sur une version de Windows plus récente que celle pour laquelle il a initialement été écrit (ce faisant, le dit pilote ne peut bien entendu pas profiter des fonctionnalités introduites avec la nouvelle version).

- *Support pour les applications héritées* Témoins d'un héritage technologique qui, s'il tend à disparaître, reste encore vivant, les produits de la gamme Windows fournissent un support également aux applications dont la réalisation (jeu d'instruction utilisé, sémantique des appels système, etc.) est héritée du lignage des systèmes Microsoft, incluant MS-DOS, Windows 16 bit et, si ce dernier n'a pas le caractère désuet des deux autres, Windows 32 bits sur architecture processeur 64 bits. Les versions 32 bits de Windows intègrent divers mécanismes pour maintenir la compatibilité avec les applications 16 bits ; à titre d'exemple un dispositif chargé de convertir les appels aux interfaces 16 bits (depuis longtemps dépréciée) en appels 32 bits équivalents. Dans la même veine, les versions 64 bits de Windows incorporent un environnement de compatibilité permettant d'exécuter une application 32 bits sur un système Windows 64 bits. WOW64 offre une couche de conversion des appels de l'API 32 bits en appels 64 bits natifs, et prend en charge de nombreuses différences entre Windows 32-bit et Windows 64-bit, en particulier celles impliquant des changements structurels.

Portabilité

Un système d'exploitation est portable s'il fonctionne sur différentes architectures et plateformes matérielles et s'adapte facilement sur de nouvelles, avec relativement peu de changements. Windows est conçu pour l'être.

La première version de Windows NT était compatible avec les architectures x86, omniprésente dans les ordinateurs personnels, stations de travail et serveurs informatiques, et MIPS, surtout en vogue (à l'époque)

dans les supercalculateurs SGI (système CISC d´un côté, RISC de l´autre). La compatibilité avec Alpha AXP de Digital Equipment Corporation fut ajoutée dans la foulée. Pionnière de l'industrie informatique à partir des années 1960 jusqu'aux années 1990, la société DEC fut rachetée par Compaq en 1998, qui fusionna avec Hewlett-Packard en 2002. Dans le milieu des années quatre-vingt dix, Windows NT 3.51 fut le premier représentant d'une courte lignée disponible pour l'architecture PowerPC. PowerPC, parfois abrégé PPC, est une gamme de microprocesseurs dérivée de l'architecture RISC POWER d'IBM, et développée conjointement par Apple, IBM et Freescale (anciennement Motorola). Le marché ayant évolué depuis lors, Microsoft profita du début des travaux sur Windows 2000 pour procéder à un élagage massif parmi les architectures prises en compte, et rompit de la sorte toute attache avec les processeurs MIPS, Alpha et PowerPC, ne laissant sur le terrain que x86. Windows XP, mis en circulation en 2001 à la fois comme mise à jour du système de bureau Windows 2000 et en remplacement de Windows 95/98, est la première version de Windows à disposer de déclinaisons 64 bits : une pour les processeurs à base de x86_64, l'autre pour Itanium. Sans grande surprise face au nombre réduit de stations de travail dotés d'un tel équipement, Microsoft décida de mettre un terme à son soutien des processeurs Itanium, Windows Server 2008 R2 étant la dernière version à pouvoir se charger d'eux. En 2001, poussé par l'explosion des ventes dans le milieu de l'informatique embarquée, comprenant entre autre la téléphonie mobile et les tablettes, la firme de Redmond annonça que les déclinaisons futures de ses systèmes d'exploitation seraient capables de fonctionner sur ARM, chose faite dans Windows RT, une version du système d'exploitation Windows 8 pour les appareils ARM. Pour finir, et concédons le, résumer un paragraphe quelque peu rébarbatif, voici une liste (sans doute non exhaustive, sachant que certaines éditions du système ne virent le jour qu'à des fins de test et n'ont jamais été commercialisées), des diverses architectures avec lesquelles Windows est, ou a été, compatible : x86-32, x86-64, MIPS, Alpha, PowerPC, IA-64 (comprendre Itanium), et, dernièrement, ARM.

Windows assure la portabilité sur différentes architectures et plateformes matérielles de deux grandes façons.

- *Couche d'abstraction matérielle* Windows dispose d'une structure en couches dans laquelle le code dépendant du matériel est isolé au sein d'une bibliothèque spéciale nommée *couche d'abstraction matérielle* (HAL, *Hardware Abstraction Layer*). Pour renforcer la portabilité, les couches supérieures du noyau reposent sur les interfaces HAL plutôt que sur les couches matérielles inférieures.

- *Langage de haut niveau* La majeure partie de Windows est écrite en C, un langage qui adopte une vision très proche de la machine, mais dont la portabilité reste un des avantages les plus importants. L'assembleur n'est en l'occurrence employé qu'en de rares occasions, toujours dans des domaines très spécifiques, et là encore avec une certaine réserve. En règle générale, les primitives Windows exprimées en assembleur le sont soit parce qu'elles accèdent directement au matériel (par exemple, le gestionnaire d'interception) ou réussissent de la sorte à procurer un avantage manifeste en ce qui concerne les performances (par exemple, le basculement de contexte). S'il est évidemment surtout possible de rencontrer de l'assembleur dans le noyau et dans la couche d'abstraction matérielle, quelques autres endroits du système lui assurent une présence plus ou moins appuyée, y compris la portion bas niveau du sous-système Windows, pour le transfert vers un périphérique d'affichage, ou la bibliothèque de support Ntdll, pour le démarrage des processus.

Sécurité

Compte tenu de son orientation en tant que système d'exploitation d'entreprise, Windows a dès le départ été pensé en termes de sécurité, intégrant pour cela un ensemble de moyens visant à conserver, rétablir et garantir l'intégrité de l'environnement et des informations que ce dernier héberge. Chaque version de Microsoft Windows assume en la matière les rôles essentiels, et apporte le catalogue des fonctionnalités nécessaires. Celles-ci incluent :

- *Normes de sécurité* Windows répond en matière de sécurité à diverses normes indiquant le degré de protection du système d'exploitation. Il est classé C2 dans la norme TCSEC (Trusted Computer Security Evaluation Criteria) et niveau F-C2 / E3 dans la norme ITSEC (Information Technologie Security Evaluation Criteria). Pour plus d'informations sur le sujet, consultez la section Sécurité et normalisations.

- *Journaux d'événements* Les journaux d'événements, des fichiers spéciaux dans lesquels Windows enregistre des informations détaillées sur les événements significatifs du système informatique, ouvrent la voie à un système de surveillance permettant de détecter les tentatives d'action non autorisée ou d'intrusion.

- *Chiffrement* Windows prend en charge le chiffrement local des données avec EFS (Encryption File System). Cette technologie, qui permet d'enregistrer des fichiers chiffrés au-dessus de NTFS, est la protection la plus renforcée fournie par Windows pour aider à sécuriser vos informations personnelles, les protégeant des attaques de personnes ayant un accès direct à l'ordinateur.

- *Contrôle d'accès* Windows intègre le souci de la sécurité par un modèle de contrôle d'accès fin et précis. Les dispositifs majeurs à cet égard incluent la protection discrétionnaire (quels sujets ont accès à quels objets et comment), la protection de la mémoire (chaque programme s'exécute dans son propre espace d'adressage virtuel privé), l'audit (détection et enregistrement des événements relatifs à la sécurité), l'authentification lors de l'ouverture de session, l'identification auprès d'une autorité certifiée et, finalement, la protection contre la réutilisation d'objets (interdiction à un utilisateur d'accéder à des ressources non initialisées ou qu'un autre utilisateur a supprimé).

- *Protection des objets* Aisément adaptable à des ressources de nature très différente, le modèle de sécurité de Windows permet de décrire les relations entre les diverses entités partageant des objets, mutualisant de cette façon le contrôle d'accès (voir point précédent) et l'audit. Renvoyant à la notion de propriété et de droits d'accès aux ressources du système, les propriétaires des objets (fichiers, imprimantes ou autres) accordent ou refusent l'accès à autrui. Chaque objet peut être doté d'une sécurité individuelle ou d'une sécurité de groupe. Les objets ont différents types de permissions, qui permettent d'autoriser ou d'interdire leurs accès. Quand un utilisateur ouvre une session, lui est attribué un ensemble de données d'identification (credentials). Quand l'utilisateur essaie d'accéder à un objet, le système compare son profil de sécurité avec celui de l'objet concerné (liste de contrôle d'accès) et vérifie que l'utilisateur est légitime dans son action envers l'objet. Si cette vérification est positive, le système légitime l'accès.

- *Solutions logicielles intégrées* Windows intègre en standard diverses solutions dédiées à la sécurité, dont un antivirus et un pare-feu. L'Outil de suppression de logiciels malveillants (MRT, Malicious Software Removal Tool) est mis à la disposition des utilisateurs pour leur permettre de débusquer et d'éliminer les menaces les plus courantes sur un PC Windows infecté. L'application pare-feu livrée avec Windows vous permet d'empêcher les utilisateurs ou logiciels non autorisés d'accéder à votre ordinateur depuis un réseau ou Internet. Le pare-feu peut également empêcher votre ordinateur d'envoyer des éléments logiciels nuisibles à d'autres ordinateurs. Antivirus et pare-feu sont de base configurés pour bloquer les risques liés aux principales menaces.

- *Centre de sécurité* Le centre de sécurité, ou centre de maintenance, vérifie le statut de l'ordinateur en matière de sécurité et affiche des avertissements en cas de risques potentiels.

- *Tolérance aux pannes* Windows intègre un ensemble de fonctionnalités permettant de fiabiliser le fonctionnement du système. Parmi ces techniques, lesquelles visent à améliorer les performances, la sécurité ou la tolérance aux pannes, figurent par exemple Active Directory (AD), File Replication Service (FRS), Distributed File System (DFS) ou encore Redundant Array of Independent Disks (RAID).

- *BitLocker* Solution de chiffrement utilisée afin de mieux protéger l'utilisateur contre le vol, BitLocker permet le chiffrement intégral de volumes, qu'il s'agisse d'un volume système (lecteur sur lequel réside Windows) ou d'un volume de données.

- *Secure Boot* Fonctionnalité liée à UEFI, Secure Boot permet lors de la procédure d'amorçage du système de vérifier que chaque élément chargé est fiable.

- *PatchGuard* Diminue la surface d'attaque du système en le protégeant de toutes modifications. Pour plus d'informations, voir \<<kernel-patch-protection>\>.

Pour replacer les choses dans le contexte qui les a vu naître, notez que la mise en conformité de Windows avec divers processus concernant la sécurité est pour une large part un aspect concomitant des objectifs de conception visant à la robustesse et à la fiabilité du système. De nos jours, la sécurité informatique est une industrie à part entière, avec des idées ayant qui leur vie propre et leur propre discernement.

Pour une vision plus large de la façon dont est gérée la sécurité dans Windows, reportez-vous au chapitre Sécurité.

Modèle de système d'exploitation

Microsoft Windows est une série de systèmes d'exploitation multiplateforme, multitâche, multi-utilisateur et multiprocesseur à base de micronoyau.

- *Multiplateforme* Windows est conçu pour fonctionner sur plusieurs plateformes et différentes architectures matérielles. Voici, sur le sujet, une liste d'architecture de processeur qui sont, ou ont été, supportés au fil des années : x86, Alpha, MIPS, PPC, Itanium, AMD64, ARM. Microsoft étoffant ou élaguant la prise en charge entre famille de processeurs en fonction des impératifs du marché (pour des raisons commerciales), les versions récentes de Windows connaissent l'architecture x86, tirent profit des capacités étendues de l'architecture x64 (les versions 64 bits de Windows peuvent solliciter une mémoire plus importante que leurs homologues 32 bits), et gèrent les architectures ARM, dominants dans le domaine de l'informatique embarquée.

- *Multitâche* Windows a de la capacité de gérer plusieurs programmes simultanément en leur attribuant tour à tour un pourcentage de temps processeur pour que ces programmes puissent s'exécuter. Chaque application occupe un processeur pendant un laps déterminé à l'avance ou jusqu'à ce qu'une autre application ait une priorité supérieure à l'application en cours. L'emploi du temps du processeur (autrement dit, quelle application doit s'exécuter à un instant t) est fait par un composant du système d'exploitation, l'ordonnanceur, qui contrôle et pilote l'exécution des divers programmes entre tous les utilisateurs.

- *Multi-utilisateur* Windows permet à plusieurs utilisateurs d'utiliser l'ordinateur simultanément. Il met en ce sens en place les technologies essentielles du contrôle d'accès (respectivement, mécanisme d'authentification, d'autorisation et de traçabilité), procède aux vérifications de sécurité allant de pair avec l'intégrité des données vitales du système d'exploitation, des utilisateurs et de leurs programmes, et assure l'éventail des dispositifs nécessaires pour protéger des regards, ou au contraire partager, les biens confiés au système d'exploitation.

- *Multiprocesseur* Windows sait gérer la présence dans l'environnement matériel de plusieurs processeurs. Il fonctionne sur des systèmes multi processeur symétriques en utilisant au mieux les multiples processeurs.

- *Micronoyau hybride* L'architecture de Windows montre les signes d'une structure interne modulaire en adoptant de la technologie à micronoyau hybride : le système n'est stricto sensu ni monolithique, ni tout à fait un micronoyau. Les principaux composants du système (par exemple le noyau, le gestionnaire de mémoire ou le gestionnaire de processus) sont exécutés dans un mode privilégié du processeur, avec accès aux données système et au matériel, et partagent un même espace d'adressage (ce qu'ils ne feraient pas dans un micronoyau pur et dur). Le code des applications est exécuté dans un mode non privilégié du processeur, avec accès limité aux données système et sans possibilité d'accéder directement au matériel. Dans Windows, comme les programmes applicatifs des utilisateurs sont pris en charge par des systèmes client/serveur et ne sauraient fonctionner sans eux, on trouve au système des allures de micronoyau. Windows est encore parfois considéré comme un micronoyau en raison du grand nombre d'objectifs de conception le rapprochant d'un de ces systèmes (en particulier la séparation des personnalités du système d'exploitation à partir d'un design général du noyau).

Systèmes multi-utilisateur

Est dit multi-utilisateur tout système d'exploitation capable de simultanément et indépendamment exécuter plusieurs applications appartenant à plusieurs utilisateurs. Simultanément, dans ce contexte, tient au fait que le système est capable de gérer en harmonie de multiples identités numériques, et de faire en sorte que les accès concurrents vers les mêmes ressources (processeur, mémoire, disques durs…) ne soient pas source de litiges. Chaque logiciel d'application, de n'importe quel utilisateur, est régi de manière indépendante et ne peut détériorer l'environnement d'un autre logiciel ; chaque tâche peut s'accomplir sans se préoccuper de ce que d'autres tâches, d'autres utilisateurs, effectuent.

Les systèmes multi-utilisateurs doivent pour parvenir à leurs fin prendre en compte plusieurs mécanismes :

- Un mécanisme d'identification, chargé de prendre connaissance de l'identité d'un utilisateur.

- Un mécanisme d'authentification, où le système valide les informations d'ouverture de session d'un utilisateur.

- Un mécanisme d'autorisation, qui accorde à un individu un droit particulier d'utilisation du système.
- Un mécanisme de protection, assurant l'intégrité du noyau et la non-interférence des applications entre elles.
- Un mécanisme de comptabilité, astreignant les tâches utilisateur à une utilisation raisonnable de la machine.

Sous-systèmes d'environnement

Reflet de comment Windows apparaît aux concepteurs de logiciels, et éventuellement aux utilisateurs (via le dessin et l'expérience graphique), un sous-système d'environnement désigne une étendue dans laquelle processus et bibliothèques mode utilisateur se voient offrir un contexte d'exécution spécifique. Chaque sous-système présente sa propre réalisation d'un environnement de système d'exploitation (on parle en ce sens de personnalité), pour lequel il rend visible une combinaison de caractéristiques, comportements et interfaces de programmation. Cela permet à Windows d'exécuter des programmes développés pour d'autres systèmes d'exploitation, tels que Windows 16 bits, MS-DOS ou UNIX.

À l'origine, il était prévu que Windows soit pourvu de trois sous-systèmes : Windows, POSIX et OS/2, que la liste suivante présente à grands traits (nous reviendrons sur eux en détail plus tard).

- *Windows* A pour rôle de faire fonctionner les applications spécifiquement conçues pour Microsoft Windows, y compris les programmes à interface graphique et ceux à interface caractère (mode console).
- *POSIX* Propose un sous-ensemble de fonctions avec lesquelles exécuter des utilitaires et des applications UNIX.
- *OS/2* Reproduit le comportement et les mécanismes du système d'exploitation OS/2, résultat d'une création conjointe entre IBM et Microsoft.

Des constituants de cette liste, le premier, et le plus important est évidemment toujours en vigueur. Le second, OS/2, a suivi un chemin similaire à son modèle : marquant le désengagement de Microsoft sur le projet, le support de OS/2 a disparu à partir de Windows 2000. Concernant POSIX, ses fonctionnalités ont été incorporées à un ensemble d'outils d'interopérabilité, les Windows Services for UNIX, qui étendent et remplacent à cet égard un sous-système POSIX jugée trop minimaliste.

Conçu à partir d'une optique générale du système d'exploitation, et des fonctionnalités offertes par lui, le rôle d'un sous-système d'environnement est d'exposer aux applications un certain sous-ensemble des services système de l'exécutif Windows. Chaque sous-système rend visible un sous-système différent des services natifs de Windows. Cela signifie qu'une application exécutée par-dessus un certain sous-système peut faire des choses que ne peut pas faire une application exécutée par-dessus un autre sous-système. Ainsi, une application Windows ne peut pas utiliser la fonction *fork*, laquelle est apparentée à POSIX.

L'architecture en sous-systèmes de Windows interdit de mélanger des appels de fonction provenant de différents environnements au sein d'une même application. En d'autres termes, une application ne peut solliciter les services que du sous-système pour lequel elle est prévue. Une seule bannière sous laquelle se rassembler étant possible, une application Windows ne peut appeler que des services exportés par le sous-système Windows, une application POSIX ne peut appeler que des services exportés par le sous-système POSIX, etc.

Développé autour d'une perspective client/serveur, chaque sous-système est porté par un processus serveur s'exécutant en mode utilisateur, duquel les applications sollicitent les services, et par un ensemble de librairies dynamiques, desquelles ils appellent les fonctions. En interne, ces librairies transcrivent les requêtes en composants compréhensibles d'un sous-système de plus bas niveau, dit natif, seul habilité à interagir avec le noyau (Windows donne pour bonne conduite des programmes le fait de ne pas s'impliquer directement avec les appels système).

Bibliothèques de sous-système

Premiers fournisseurs des technologies sur lesquelles s'appuient les applications mode utilisateur, chaque sous-système réalise ses fonctionnalités dans une ou plusieurs bibliothèques. Les bibliothèques de sous-système

exportent l'interface que peuvent appeler les programmes liés au sous-système. Ainsi, les DLL du sous-système Windows (telles Kernel32.dll, Advapi32.dll, User32.dll et Gdi32.dll) implémentent les fonctions de l'API Windows. La DLL du sous-système POSIX implémente les fonctions de l'API POSIX.

Quand une application appelle une fonction ayant corps dans une DLL de sous-système, il peut se passer l'une des trois choses suivantes :

- La fonction est entièrement implémentée dans la DLL de sous-système. A ce niveau, aucun autre traitement supplémentaire n'est requis, il n'y pas de message envoyé au processus du sous-système d'environnement, ni non plus d'appel aux services système de l'exécutif Windows. En pareil cas, la fonction s'exécute et rend la main à l'appelant, retournant quelquefois au passage un résultat. Exemples : *GetCurrentProcess*, qui retourne la valeur définie pour le pseudo handle associé au processus en cours (systématiquement -1), ou *GetCurrentProcessId*, laquelle fonction retourne l'ID du processus appelant, qui ne change pas pour un processus en cours d'exécution et est en interne lu depuis une zone de données directement accessible en mode utilisateur.

- La fonction requiert le concours des services de l'exécutif Windows, et exige à ce titre un accès aux services noyau. A ce cas de figure correspond, par exemple, les fonctions Windows ReadFile et WriteFile, qui impliquent des appels aux services d'E/S Windows NtReadFile et NtWriteFile, respectivement.

- La fonction nécessite la coopération du processus de sous-système d'environnement. A ce moment, on adresse via fonctionnalité d'appel a une procédure locale (LPC) une requête client-serveur au processus de sous-système. La DLL de sous-système attend alors une réponse avant de rendre la main à l'appelant.

Pour finir sur une note plus globale, chaque fonction provenant d'une DLL de sous-système peut, d'après ce qui vient d'être exposé, être amenée à s'interfacer avec un ou plusieurs autre(s) composant(s) de nature diverse : services natifs, librairies tierces, voire d'autres processus. A de maints égards, cette façon de faire illustre, par les multiples interactions qu'elle suppose, à la fois la dimension en couches de Windows, et le côté coopératif des divers composants.

Informations de démarrage de sous-système

Les informations de démarrage de sous-système sont hébergés sous les auspices du gestionnaire du session (Smss) au sein de la valeur de registre HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\SubSystems.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Subsystems
    (par défaut)    REG_SZ    mnmsrvc
    Debug    REG_EXPAND_SZ
    Kmode    REG_EXPAND_SZ    \SystemRoot\System32\win32k.sys
    Optional    REG_MULTI_SZ
    Required    REG_MULTI_SZ    Debug\0Windows
    Windows    REG_EXPAND_SZ    %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1024,20480,768
 Windows=On SubSystemType=Windows ServerDll=basesrv,1 ServerDll=winsrv:UserServerDllInitialization,3 ServerDll=sxssrv,4
 ProfileControl=Off MaxRequestThreads=16
```

La valeur Required énumère les sous-systèmes chargés au démarrage du système. La valeur a deux deux chaînes : Windows et Debug. La valeur Windows contient l'emplacement et le nom du fichier du sous-système Windows, Csrss.exe (Client/Server Run-Time Subsystem). La valeur Debug, généralement vide, est employée dans l'optique de tests internes. La valeur Optional indique que le sous-système Posix sera chargé à la demande. La valeur Kmode indique les données de chemin d'accès du fichier abritant la portion mode noyau du sous-système Windows, Win32k.sys.

Environnement MS-DOS (NTVDM)

Reproduction des conditions du système d'exploitation de type DOS développé par Microsoft pour les compatibles PC, l'environnement MS-DOS est supporté dans Windows par le biais d'une application spécifique

appelée machine virtuelle DOS (VDM, *Virtual DOS Machine*), qui fournit un contexte d'exécution adéquat aux applications conçues pour ce système. Par définition, tous les programmes écrits pour MS-DOS sont des programmes 16 bits.

Les technologies enracinées dans la machine virtuelle DOS dépendent du mode virtuel 8086 du processeur Intel 80386, lequel permet d'exécuter des logiciels conçus pour le processeur 8086 en mode réel dans un environnement piloté par interruptions. Chaque machine DOS s'appuie sur une unité d'exécution d'instruction (*instruction-execution unit*) interne pour exécuter ou émuler des instructions, et a des pilotes virtuels pour l'écran, le clavier et les ports de communication.

Les machines DOS virtuelles sont apparues avec Windows 2.1 386 et sont présentes dans toutes les versions 32 bits subséquentes de Windows. Dans la famille Windows NT, laquelle marque l'abandon des technologies héritées de MS-DOS et de Windows 3.x, elles sont toutefois reléguées à émuler DOS et ne ne peuvent plus interagir avec l'API Windows.

L'exécutable sous-jacent à la couche d'émulation DOS dans Windows NT et supérieurs porte le nom de ntvdm.exe et démarre automatiquement quand l'utilisateur sollicite une application 16 bits dans un environnement 32 bits. Les versions 64 bits de Windows ne possèdent pas cet émulateur.

Sous-systèmes et création de processus

Chaque module exécutable (.exe) est lié à un sous-système. A cet effet, l'attribut Subsystem dans l'entête facultative de l'image détermine pour quel environnement le fichier est prévu. Les valeurs définies à cet égard incluent :

- *IMAGE_SUBSYSTEM_UNKNOWN* Le sous-système pour lequel se destine l'image n'est pas connu. Le fichier ne peut être chargé.
- *IMAGE_SUBSYSTEM_NATIVE* Sous-système non requis. Cette option est généralement réservée aux composants du système Windows, incluant pilotes de périphérique et processus système natifs.
- *IMAGE_SUBSYSTEM_WINDOWS_GUI* Pour une application Windows en mode graphique. L'application ne nécessite pas de console, car elle crée probablement ses propres fenêtres d'interaction avec l'utilisateur.
- *IMAGE_SUBSYSTEM_WINDOWS_GUI* Pour une application Windows en mode caractère. Le système d'exploitation fournit automatiquement une console aux applications console.
- *IMAGE_SUBSYSTEM_POSIX_CUI* Pour une application de type UNIX s'exécutant avec le sous-système POSIX dans Windows.
- *IMAGE_SUBSYSTEM_WINDOWS_CE_GUI* Le fichier est prévu pour être exécuté dans l'environnement Windows CE, lequel n'est pas reproduit dans Windows.

Lors de l'exécution d'une image, le code de création de processus (fonction Windows *CreateProcess*) examine le code de type de sous-système dans l'en-tête de l'image de façon à informer le bon sous-système de l'existence du nouveau processus. Si le fichier exécutable est une application Windows native, il est exécuté sans détour. Autrement (par exemple, s'il s'agit d'une application MS-DOS ou POSIX), du code additionnel se greffe aux opérations en place pour trouver une *image de support* permettant d'exécuter le fichier. Les applications non natives ne pouvant être exécutées directement, Windows utilise comme intermédiaire l'une des quelques images de support spéciales qui ont la charge d'executer réellement les programmes non Windows. Le tableau suivant associe quelles applications sont pris en charge par quel sous-système.

Tableau 3.1. Associations entre sous-systèmes et processus de support

Type d'application	Processus de support
Fichier exécutable POSIX	Posix.exe
Application MS-DOS	Ntvdm.exe

Application Win16	Ntvdm.exe
Procédure de commandes (.bat ou .cmd)	Cmd.exe

Pour la plupart des programmes Windows, deux types de sous-système de l'image sont utilisées : Windows GUI (IMAGE_SUBSYSTEM_WINDOWS_GUI dans le tableau x), correspondant aux programmes à interface graphique, et Windows CUI (IMAGE_SUBSYSTEM_WINDOWS_CUI) pour les programmes à interface caractère. L'option /SUBSYSTEM de l'Editeur de liens (link.exe) permet de spécifier quel environnement est apparenté à un fichier executable. La sélection de cette option affecte le symbole de point d'entrée (ou la fonction de point d'entrée) que l'éditeur de liens sélectionne (WinMain, DllMain, DriverEntry, etc.)

Composants fondamentaux du système

Après nous être intéressé à l'architecture générale de Windows, nous allons dans cette section revenir plus en détail sur le rôle joué par chacun des composants clé du système d'exploitation.

Ntdll.dll

Ntdll.dll est une bibliothèque spéciale de support autour de laquelle s'articulent différents composants Windows exécutées en mode utilisateur. Ntdll contient principalement deux types de fonctions :

- Des fonctions points d'entrée vers les services système de l'exécutif.
- Des fonctions de support internes sollicitées par les sous-systèmes, les DLL de sous-système et autres images natives.

Le premier groupe de fonctions assure l'interface vers les services système de l'exécutif pouvant être appelés depuis le mode utilisateur. Il existe des centaines de ces fonctions, pour la plupart accessibles depuis l'API Windows. Par exemple, Ntdll exporte NtCreateProcess, qui constitue en l'occurence l'interface commune à les fonctions Windows de création de processus. Autres exemples : NtReadFile, service interne sur lequel s'appuie ReadFile (Kernel32.dll) pour lire des données à partir d'un fichier ; NtSetEvent, employé par SetEvent pour définir le signal associé à un objet de synchronisation type événement, et ainsi de suite. Chacune de ces fonctions contient le code spécifique qui, après vérification de certains paramètres, force une transition vers le mode noyau pour invoquer le dispatcher de service système, lequel passe finalement le relais au service système proprement dit.

Ntdll renferme aussi maintes fonctions de support internes, par exemple le chargeur d'image (fonctions commençant par Ldr), le gestionnaire de tas, le dispatcher d'APC, les mécanismes de communication du processus du sous-système Windows (fonctions commençant par Csr), les routines générales de la bibliothèque d'execution (fonctions commençant par Rtl).

Exécutif

Au-dessus du noyau et des pilotes de périphériques se trouve la partie supérieure du système d'exploitation appelée exécutif, formée d'un petit nombre de composants eux-mêmes constitués d'une multitude de procédures collaborant à la réalisation d'une activité ou d'une tâche donnée. L'exécutif est écrit en C ; il est de la sorte indépendant de l'architecture, prend en compte un large éventail de plateformes, et peut facilement être porté sur de nouvelles.

La liste suivante énumère quels sont les grands composants de l'exécutif Windows. Chacun d'eux sera traité en détail dans un chapitre ultérieur.

- *Gestionnaire d'alimentation* Coordonne les événements système résultants d'un changement de l'état d'alimentation de l'ordinateur et génère des notifications d'E/S de gestion électrique aux pilotes de périphériques concernés.

- *Gestionnaire d'E/S* Fournit le cadre pour implémenter les pilotes de périphériques, de même que certains services spécifiques à la configuration, l'accès ou la réalisation d'opération sur les périphériques.
- *Gestionnaire de Plug and Play (PnP)* Détermine quels sont les pilotes requis par la prise en charge d'un certain matériel, par exemple une carte réseau, et charge ces pilotes.
- *Gestionnaire de processus* Traite le cycle de vie des diverses entités exécutables gérés par le système d'exploitation, y compris les processus, les threads, et les travaux (ensemble de processus administrable comme un tout).
- *Gestionnaire de mémoire* Implémente la mémoire virtuelle et les procédés auxquels cette technique ouvre la voie, par exemple les fichiers mappés en mémoire ou la stratégie d'optimisation copy-on-write. En outre, le gestionnaire de mémoire fournit la prise en charge sous-jacente au gestionnaire de cache.
- *Gestionnaire d'objets* Crée, gère et supprime les objets de l'exécutif et abstrait les types de données qui servent à représenter les ressources système du genre processus, threads et fichiers.
- *Gestionnaire de cache* Fournit des services globaux de cache aux différents pilotes de systèmes de fichiers (NTFS et autres), et optimise de la sorte les entrées/sorties disque en conservant en mémoire centrale des parties de fichiers lues ou à lire.
- *Gestionnaire de configuration* Met en oeuvre et assure le suivi du Registre Windows, lequel héberge une collection de différents paramètres liés au système d'exploitation, ainsi qu'aux logiciels et aux utilisateurs.
- *Moniteur de références de sécurité* Contrôle et fait appliquer les stratégies de sécurité au niveau de l'ordinateur local (connexion authentifiée, mise à zéro de la mémoire allouée, protection de l'espace d'adressage, contrôle d'accès, etc.), assurant de cette façon la protection et l'audit des ressources du système.
- *Gestionnaire WMI* Permet aux composants de l'exécutif de publier des données de configuration et de performances et de recevoir des commandes du service WMI mode utilisateur.
- *Prefetcher logique* Accélère le démarrage du système et l'accès aux programmes les plus utilisés.
- *Système LPC* Assure la communication entre processus serveur et processus client sur une même machine.
- *Gestionnaire d'errata* Fournit des solutions de contournement pour les périphériques matériels présentant une configuration non standard ou non conforme à leur bon fonctionnement.
- *Windows Hypervisor Interface Library (WinHv)* Établit un pont entre les pilotes d'un système d'exploitation virtualisé et l'hyperviseur lui-même, ce qui permet aux pilotes de solliciter l'hyperviseur à l'aide des conventions d'appel standard de Windows.
- *Gestionnaire de transaction noyau* Prend en charge les aspects transactionnels de NTFS et du registre.
- *Event Tracing for Windows* Fournit des routines d'assistance pour le suivi des événements.
- *Windows Diagnostic Infrastructure (WDI)* Intègre des scénarios de diagnostic destinées à résoudre les problèmes externes qui affectent le comportement de Windows.
- *Event Tracing for Windows* Fournit des routines d'assistance pour le suivi des événements.
- *Windows Diagnostic Infrastructure (WDI)* Intègre des scénarios de diagnostic destinées à résoudre les problèmes externes qui affectent le comportement de Windows.
- *Driver Verifier* Soumet les pilotes à un ensemble de contraintes et de tests, en vue notamment de valider leur qualité.
- *Windows Hardware Error Architecture* Définit un cadre commun de gestion des erreurs matérielles.

Gestionnaire de processus

Le gestionnaire de processus de Windows fournit les services nécessaires à la création, la suppression et l'utilisation des processus, des threads et des travaux. Il a pour rôle la définition des critères et des attributs latents au support de ces dispositifs dans le système, et ouvre la voie à leur transposition sous forme algorithmique, à leur mise en oeuvre, et enfin à leur programmation.

Vu la complexité de la mission à réaliser et la diversité des fonctionnalités à pourvoir, le gestionnaire de processus fonctionne en interaction avec de nombreux autres composants de la couche exécutive : gestionnaire de mémoire, pour fournir à chaque processus un espace d'adressage virtuel privé ; moniteur de références de sécurité, afin que tout processus soit systématiquement contrôlé lors des opérations qui doivent l'être ; *prefetcher* logique, de façon à accélérer le démarrage de certains processus ; etc.

Le gestionnaire de processus représente la partie la plus visible des techniques utilisées pour la gestion des flux d'exécution. La prise en charge sous-jacente des processus et des threads est implémentée dans le noyau Windows ; l'exécutif ajoute des fonctionnalités supplémentaires relatives à ces objets de bas niveau.

En plus des mécanismes généraux de supervision des entités sous sa coupe, le gestionnaire de processus a aussi pour tâche la mise au point de procédés connexes, tels la mise en file d'attente et la livraison des appels de procédure asynchrones (APC, *asynchronous procedure calls*) aux threads.

Bien que les systèmes Windows tiennent compte d'une certaine forme de hiérarchie entre les processus, le gestionnaire de processus n'a aucune connaissance des relations entre processus pères ou fils. Cette subtilité est traitée par le sous-système d'environnement particulier duquel le processus hérite ses caractéristiques. Le gestionnaire de processus n'est également pas impliqué dans l'ordonnancement des processus, hormis en ce qui concerne le positionnement des priorités et affinités des processus et des threads lors de leur création.

Sur le plan de la sémantique et de la programmation système, les noms des routines qui fournissent une interface directe avec le gestionnaire de processus et de threads sont couramment assortis du préfixe Ps, par exemple *PsCreateSystemThread* ou *PsGetCurrentProcess*.

Couche d'abstraction matérielle (HAL)

Comme mentionné en début de chapitre (section Objectifs de conception), Windows a dès le départ été conçu pour fonctionner sur différents types de machines. En creux de cette capacité d'adaptation, HAL (Hardware Abstraction Layer), qui agit tel un intermédiaire entre le système d'exploitation et les équipements de l'ordinateur.

Sur le papier un ensemble de prescriptions d'ordre structurel et technique, HAL se présente concrètement sous la forme d'un module noyau chargeable (Hal.dll) qui fournit l'interface bas niveau vers la plateforme matérielles, masquant ainsi les détails d'implémentation dépendants du matériel tels que les interfaces d'E/S, les contrôleurs d'interruption et les mécanismes de communication multi processeur.

Vous pouvez évaluer les relations qui unissent les images du noyau (Ntoskrnl.exe) et de la couche d'abstraction matérielle (Hal.dll) en examinant leurs tables d'importation et d'exportation respectives via l'utilitaire Dependency Walker (Depends.exe). Vous remarquerez alors que Ntoskrnl est lié à Hal, qui lui-même est lié à Ntoskrnl. Chacun, en d'autres termes, sollicite et bénéficie des fonctions que l'autre rend visible.

Pour voir quel HAL est en cours d'exécution sur la station de travail, démarrez une session de débogage puis entrez la commande lm vm hal.

```
lkd> lm vm hal
Browse full module list
start             end                 module name
fffff801`b7b54000 fffff801`b7be0000   hal        (deferred)
    Image path: hal.dll
    Image name: hal.dll
    Browse all global symbols  functions  data
    Image was built with /Brepro flag.
```

```
    Timestamp:        4252FF42 (This is a reproducible build file hash, not a timestamp)
    CheckSum:         000892FF
    ImageSize:        0008C000
    Translations:     0000.04b0 0000.04e4 0409.04b0 0409.04e4
    Information from resource tables:
```

Au lieu d'accéder directement au matériel, les composants internes de Windows assurent la portabilité en sollicitant les routines HAL dès lors qu'ils manipulent des informations liées à la plateforme. De ce fait, un pilote qui utilise correctement HAL pour accéder aux périphériques ne nécessite aucune modifiction (autre que la recompilation) pour lui permettre de s'exécuter sur les les différentes plateformes matérielles reconnues par Windows.

Les ressources pour lesquelles HAL fournit une abstraction sont les suivantes :

- Adressage des périphériques
- Architecture d'E / S
- Gestion des interruptions
- Opérations DMA
- Horloges et minuteries système
- Gestion de la configuration

Noyau

Au plus large degré d'abstraction, le noyau Windows consiste en un base de primitives grâce auxquelles d'autres composants du système (services de l'exécutif, sous-systèmes d'environnement, etc.) assurent leur fonction. Ses rôles centraux incluent la gestion du matériel, l'ordonnancement et la synchronisation de threads, la prise en charge des interceptions (interruptions et exceptions), et la synchronisation multi processeur.

L'une des responsabilités les plus importantes du noyau est d'offrir l'interface de haut niveau vers la plateforme matérielle. Il rend à cet égard visible un vaste ensemble de données et d'interfaces sémantiquement identiques sur toutes les architectures. Comme c'est le cas des diverses fonctions de support de l'exécutif mentionnées dans la section qui précède, un certain nombre de fonctions du noyau sont documentés.

Autre rôle crucial attribué au noyau : offrir des accès mutuellement exclusifs sur l'ensemble des processeurs. Il emploie pour ce faire diverses méthodes, notamment spin locks et opérations inter verrouillées.

Un aspect intéressant à relever en ce qui concerne le noyau Windows est qu'il évite au plus possible les décisions stratégiques, les conférant en réalité à l'exécutif.

Le noyau est écrit essentiellement en C, l'assembleur étant dans ce contexte cantonné aux tâches qui exigent l'accès à des registres et instructions spécialisés du processeur.

Pour obtenir des informations concrètes en ce qui concerne le noyau, saisissez dans une fenêtre de débogueur noyau la commande lm vm nt.

```
lkd> lm vm nt
Browse full module list
start             end                 module name
fffff801`b7201000 fffff801`b7b54000   nt         (pdb symbols)          c:\symbols\ntkrnlmp.pdb
\DD0D1D2022E0401BB9551D9DD9ECDF581\ntkrnlmp.pdb
    Loaded symbol image file: ntkrnlmp.exe
    Image path: ntkrnlmp.exe
    Image name: ntkrnlmp.exe
    Browse all global symbols  functions  data
    Timestamp:        Thu Sep 20 05:40:30 2018 (5BA316AE)
    CheckSum:         008B3C39
```

```
    ImageSize:        00953000
    Translations:     0000.04b0 0000.04e4 0409.04b0 0409.04e4
    Information from resource tables:
```

Sur le plan des dépendances externes, Ntoskrnl est en premier lieu lié à Hal.dll, correspondant à la couche d'abstraction matérielle. Il l'est également à Bootvid.dll, pilote vidéo d'amorçage servant à afficher l'écran de démarrage, à Ci.dll, module d'intégrité du code, à Clfs.sys, pilote de système de fichier journal commun (CLFS, Common Log File System), à Kdcom.dll, qui contient du code d'infrastructure de débogueur noyau, et enfin à Pshed.dll, pilote d'erreurs matérielles spécifiques à une plateforme.

Processus système fondamentaux

Quelques-uns des dispositifs de base de Microsoft Windows, y compris la gestion des utilisateurs, la prise en compte de la sécurité ou encore le suivi des services, sont le fait d'une collection d'acteurs individuels, organisés sous la forme de processus venus compléter le jeu de fonctionnalités exposées depuis le système d'exploitation. Chaque système Windows met à ce titre en oeuvre les processus que voici. (Deux d'entre eux, Inactif et System, ne sont pas des processus à part entière, vu que leur déroulement n'implique pas stricto sensu de module exécutable.)

- *Processus inactif du système* (System Idle Process) Occupe le processeur tandis qu'il ne peut l'être autrement.
- *Processus System* Héberge la majorité des threads système mode noyau.
- *Processus d'ouverture de session* (*Winlogon.exe*) Gère l'ouverture et la fermeture des sessions interactives.
- *Gestionnaire de session locale* (*Lsm.exe*) Gère l'ouverture de session locale.
- *Serveur d'authentification de sécurité locale* (*Lsass.exe*) Gère les mécanismes de sécurité locale et d'authentification des utilisateurs via le service Winlogon.
- *Gestionnaire de contrôle de service* (*Services.exe*) Permet l'interaction avec les services et les processus hébergés dedans.
- *Sous-système Windows* (*Csrss.exe*) Implémente le code mode utilisateur du sous-système d'environnement Windows. Gère les fenêtres et les éléments graphiques de Windows.
- *Gestionnaire de session* (*Smss.exe*) Administre les processus et autres objets système (station fenêtre, bureaux et fenêtres) utilisés dans le but de représenter la session d'un utilisateur.
- *Processus d'initialisation de la session 0* (*Wininit.exe*) Gère le démarrage de Windows (session 0).

Pour mieux comprendre les relations entre les protagonistes susmentionnés, il est à propos de s'intéresser à la place qu'occupe chacun au sein de l'arborescence des processus. Utilisez pour ce faire un utilitaire permettant de montrer les liens parent/enfant entre processus, par exemple PsList, ou de manière plus graphique, Process Explorer.

```
C:\>*pslist.exe -t*

pslist v1.3 - Sysinternals PsList
Copyright (C) 2000-2012 Mark Russinovich
Sysinternals - www.sysinternals.com

Process information for HP:

Name                             Pid Pri Thd  Hnd      VM      WS    Priv
Idle                               0   0   8    0      64       4       0
  System                           4   8 170 1535  138556   32808     884
    smss                         392  11   2   49 4194303     208     352
csrss                            532  13  15  476 4194303    1780    1608
wininit                          624  13   2   86 4194303     680    1036
```

```
services                        700   9   5  425 4194303    5104    3416
  svchost                       440   8  25 1151 4194303   14608   17632
    dasHost                    5684   8   4  312 4194303   10812    5516
  svchost                       904   8  27  753 4194303   12936    8352
  .
  .
  .
```

Tous les processus dont nous venons de parler fonctionnent dans le contexte du compte système local, lequel est le compte de service qui bénéficie des plus hautes accréditations sur la machine. Windows les considère de ce fait comme des parties approuvées du système d'exploitation.

Processus inactif du système

Préfigurant la résolution à une figure classique des questions soulevées en matière d'ordonnancement, le processus inactif du système se différencie des autres processus par sa seule fonction, à savoir occuper le processeur lorsqu'il ne peut l'être par aucune autre tâche. Il fait de la sorte référence à la vacance des ressources du système (en premier lieu celle du processeur), lesquelles sont par conséquent disponibles pour qui les réclame.

Contrairement aux autres processus, le processus inactif du système travaille lorsque les autres processus ne font rien. Le degré auquel ce processus emploie les ressources machine se rapporte donc en réalité à un certain facteur d'inaction au sein de celles-ci. Ainsi, une valeur de 90% comme taux d'utilisation du processeur pour le processus inactif signifie que 90% des ressources processeur ne sont pas utilisées.

Une autre particularité que présente le processus inactif (de même par ailleurs que celui baptisé System) est de ne pas avoir d'auxiliaire de type image (.exe) à lui être spécifiquement dédié. Les instructions et les données sous-jacentes sont en fin de compte hébergés au niveau du fichier système spécifique au noyau (Ntoskrnl.exe ou équivalent). Comme les processus sont généralement identifiés par les noms des images associées, le nom affiché pour le processus inactif (ID 0) diffère d'un outil à l'autre. Pour résoudre cette ambiguïté, partez du fait que l'ID du processus inactif est toujours initialisé à la valeur zéro.

Winlogon

Le Processus d'ouverture de session Windows (\\Windows\System32\Winlogon.exe) gère les ouvertures et fermetures de session interactive, le déclenchement et l'arrêt des écrans de veille, et l'arrêt/redémarrage de la machine. Il est en outre responsable de la présentation de l'écran d'accueil sur la station de travail, lequel invite généralement à saisir les informations d'identification d'un utilisateur.

Les parties identification et authentification du processus d'ouverture de session sont mises en oeuvre de façon exogène par l'intermédiaire d'interfaces spécialisés, qui font dans ce contexte office de courtiers entre l'utilisateur et l'autorité de sécurité locale (Lsass). Dans les systèmes d'exploitation Windows NT4, Windows 2000, XP et Windows 2003, c'est au premier lieu le module GINA (Msgina.dll) qui offre des services sécurisés en la matière. À partir de Windows Vista, ce rôle est confié à LogonUI, lequel agit en interaction avec divers fournisseurs d'informations d'authentification.

Lors de son démarrage, Winlogon crée 3 bureaux, un bureau sécurisé dans lequel les interactions sécurisées auront lieu, un bureau appelé à être actif quand l'écran de veille se déclenchera, et un autre qui deviendra celui de l'utilisateur interactif. Il enregistre alors la combinaison clavier SAS (Secure Attention Sequence), de sorte à être informé chaque fois qu'un utilisateur interagit avec cette séquence de touches.

La séquence SAS, par défaut CTRL + ALT + SUPPR sert à protéger l'ouverture de session. Lors de la phase d'amorçage, la portion mode noyau du sous-système Windows (Win32k.sys) réserve ladite combinaison de touches afin que, chaque fois que le gestionnaire de saisie clavier de Windows (mis en œuvre dans un thread spécifique au sein de Win32k.sys) l'intercepte, il en résulte l'envoi d'un message RPC vers le processus Winlogon (qui est à l'écoute de notifications d'authentification). Les touches entrées au clavier qui correspondent à une touche de raccourci enregistrée ne sont jamais envoyés à un processus autre que celui souscripteur, et seul

le thread ayant enregistré une touche de raccourci peut annuler son inscription. Le système est partant de là conçu de telle façon que la combinaison SAS ne peut être interceptée par un programme autre que le processus d'ouverture de session. Cela est notamment utile contre les programmes de capture de mot de passe, ou tout autre logiciel malveillant basé sur l'usurpation d'identité.

Quel que soit la méthode ayant débouché sur son entrée en scène (ouverture de la session initiale ou SAS), Winlogon affiche à l'écran le bureau sécurisé, lequel présente selon le contexte l'écran d'ouverture de session, la boîte de dialogue Sécurité de Windows, menant à une variété de mesures tels que fermer la session, lancer le gestionnaire des tâches, verrouiller la station, arrêter le système, ou encore les fenêtres associées au consentement de contrôles de comptes utilisateurs. L'écran de veille, géré comme un cas à part, dispose en conséquence de son bureau, toujours sous la supervision de Winlogon.

Apres saisie des données d'identification relatives à un utilisateur (lesquelles sont généralement associées au couple nom et mot de passe), celles-ci sont relayés à LSASS pour validation. Par la suite, Winlogon donne lieu au premier processus créé dans le contexte utilisateur, Userinit, qui est justement le sujet de la section suivante.

Userinit

Premier des processus exécutés dans le contexte de sécurité d'un utilisateur spécifique, Userinit a pour principal objectif de façonner la session de travail selon un ensemble de préférences fonctionnelles et esthétiques. C'est le processus d'ouverture de session (Winlogon), à la suite de l'examen de la clé de registre userinit sous l'emplacement HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon, qui fait appel au processus Userinit (\Windows\System32\Userinit.exe). Entre autres opérations effectuées à l'échelle de ce processus, le chargement du profil utilisateur et le démarrage de l'interface graphique de Windows.

Etroitement liée à l'architecture de scripts incorporée à Windows, elle-même fortement couplée aux paramètres de stratégie, Userinit contrôle les scripts d'ouverture et de fermeture de session, ainsi que les scripts de démarrage et d'arrêt de l'ordinateur. Il itère à cet effet les entrées des arborescences HKCU\Software\Policies\Microsoft\Windows\System\Scripts et HKLM\Software\Policies\Microsoft\Windows\System\Scripts. (Les paramètres de configuration ordinateur définissent un script à l'arrêt et au démarrage de la machine indépendamment de l'utilisateur. Les paramètres de configuration utilisateur appliquent le script à l'ouverture et à la fermeture de session sur n'importe quel ordinateur.) Chaque fois que nécessaire, Winlogon déclenche une nouvelle instance du processus Userinit, lequel se base sur l'emploi de la fonction ShellExecute pour l'exécution des scripts.

Hormis ceux déployés par la stratégie de groupes, Userinit gère plusieurs autres emplacements de registre et répertoires afin de démarrer automatiquement des processus. Entrent dans cette catégories les clés Run et RunOnce situées sous HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion et sous HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion, ainsi que les dossiers %SystemDrive%\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup et %SystemDrive%\Users\%UserName%\AppData\Roaming\Microsoft\Windows\Start Menu.

Autre mission prise en charge par Userinit, le démarrage de l'environnement de bureau Windows, soit celui associé de manière naturelle aux systèmes d'exploitation Microsoft, soit une alternative comme LiteStep. Durant la phase d'initialisation, Userinit crée un processus pour exécuter ce qui est indiqué par la valeur HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell. Si cette valeur n'existe pas (ce qui est le cas par défaut), il réitère la même procedure pour HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell, qui indique par défaut Explorer.exe.

Un autre rôle que joue Userinit se situe dans l'application des quotas. Ainsi, si la stratégie de groupe fait mention de restriction en ce qui concerne les profils utilisateur, Userinit donne le départ au processus Proquota, chargé en l'occurence de faire respecter le quota du profil de l'utilisateur courant.

Après avoir mené à bien les différentes tâches que nous venons de décrire, Userinit se termine, ce qui a pour effet de couper tout lien de parenté évident entre ce processus et ses fils. De ce fait, Explorer.exe a la particularité d'être toujours montré comme privé de parents au sein des différents utilitaires de visualisation de processus.

Wininit

Le processus d'initialisation de la session console est en charge d'un certain nombre d'étapes préliminaires au déroulement d'autres processus exécutés sous la session console (session 0), y compris le gestionnaire de contrôle des services (services.exe), le serveur d'authentification de sécurité locale (lsass.exe) et le gestionnaire de session locale (lsm.exe).

C'est le thread principal du processus Smss qui démarre Wininit. Lors de l'amorçage du système, Smss détermine l'emplacement du fichier image à exécuter pour la prise en charge de la session 0 au moyen de la valeur S0InitialCommand sous HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager, laquelle mène par défaut à %SystemRoot%\system32\wininit.exe.

SMSS (*Session Manager Subsystem*)

Premier des processus mode utilisateur créés dans le système, le gestionnaire de session a pour tâche principale de préparer l'ordinateur en vue de son utilisation par un tiers. C'est le thread système mode noyau effectuant la phase finale de l'initialisation de l'exécutif qui crée le processus Smss (\Windows\System32\Smss.exe).

Le gestionnaire de session est responsable d'un certain nombre d'étapes importantes du démarrage de Windows, parmi lesquelles la création des variables d'environnement, l'ouverture de fichiers de pagination supplémentaires, la prise en charge des opérations différées de niveau fichier, et d'autres. Il participe de surcroît à la mise en place du sous-système Windows, du processus d'ouverture de session, et du processus d'initialisation de la session 0, lequel crée le reste des processus système.

La plus grande partie des données de registre concernant le gestionnaire de session se trouvent sous HKLM \SYSTEM\CurrentControlSet\Control\Session Manager. Les valeurs de premier niveau situées sous cette clé servent principalement à la constitution de paramètres initiaux pour l'initialisation de Smss. Les sous-clés hébergées à partir de cet emplacement consolident, quant à elles, pour la plupart, des opérations qui dépassent le strict cadre du gestionnaire de session, par exemple la taille du fichier d'échange ou les informations d'activation (WPA).

Smss offre toute la logistique nécessaire afin que Windows soit un système à multiples sessions. Pour établir une plus grande séparation entre les processus mode utilisateur et le système d'exploitation, Smss crée au démarrage deux sessions, et installe dans chacune une copie de lui-même. La première instance de Smss est exécutée au sein de la session console (appelée session 0), laquelle est exclusivement réservée aux services et aux processus système. La seconde instance fonctionne dans la première session à disposer d'une connexion interactive, la session 1.

L'une des premières actions de large ampleur à se concrétiser dans le cadre du gestionnaire de session est la détection des éventuels programmes et services à exécuter lors de l'amorçage du système, mais avant le sous-système Windows. Le gestionnaire de session procède à cet effet à l'examen de la valeur BootExecute sous HKLM\SYSTEM\CurrentControlSet\Control\Session Manager, servant en l'occurrence à identifier les logiciels qui se situent à ce niveau, et donne le départ aux programmes définis par ce biais. En principe, la valeur BootExecute contient une commande pour exécuter la vérification des systèmes de fichiers, correspondant au module autochk. (Autochk est une version spéciale de Chkdsk prévue uniquement pour des disques NTFS et exclusivement pour l'amorçage.)

Une fois que sa configuration interne a définitivement pris forme, le gestionnaire de session charge la partie mode noyau du sous-système Windows, à savoir Win32k.sys. (A titre informatif, c'est là que Windows bascule l'affichage du mode texte au mode graphique.) Il sollicite ensuite dans le but de les voir démarrer le processus d'initialisation de la session 0 (Wininit), le processus Windows coté utilisateur (Csrss) et le processus d'ouverture de session (Winlogon). Si l'un ou l'autre des processus Csrss et Wininit se termine de manière inattendue, le noyau fait s'effondrer le système, Windows dépendant étroitement d'eux. Dans le cas où ce dysfonctionnement concerne Winlogon, le système met brutalement fin à la session associée.

Smss évalue quels sont les processus de sous-système à charger automatiquement au démarrage (en principe, seulement Csrss, dans la mesure où les autres sous-systèmes sont prévus pour fonctionner à la demande) en

se référant aux informations stockées sous HKLM\SYSTEM\CurrentControlSet\Control\Session Manager. Plus spécifiquement, Smss consulte à cet égard la valeur Required, laquelle redirige généralement vers une autre valeur, nommée Windows, qui contient la position du l'exécutable csrss.exe dans le système de fichiers. La mise en marche de Csrss donne aux sessions la possibilité de gérer les applications mode utilisateur, qui emploient dans la plupart des cas de figure l'API Windows.

La prise en charge des sessions s'effectue également par le biais de Smms, à la fois celle de la session non interactive, et de celles qui le sont. Une *session* se compose des processus et des autres objets système (station fenêtre, bureaux, fenêtres, etc.) qui représentent la session ouverte par un utilisateur sur une station de travail. Quand Smss crée la première des sessions interactives (ID 1), ou quand Smss recoit une demande de création de session, le processus du gestionnaire de session (Smss.exe) commence par faire apparaitre une instance de lui-même au sein de cette session, à laquelle il passe le relais. Commence à partir de là le scénario préparatoire à l'emploi de toute nouvelle session, ce qui inclut le chargement d'un exemplaire propre à la session de Win32k.sys, la création de l'espace de noms de gestionnaire d'objets propre à la session, et la création des instances spécifiques à la session des processus Winlogon et Crsss - des processus Wininit et Crsss en ce qui concerne la session 0.

Ainsi que nous l'avons dit, l'une des premières tâches de Smms est de démarrer le sous-système Windows. À ce titre, le fait que l'un entre en scène avant l'autre constitue une particularité originale. Dépourvu en la circonstance des API Windows, dont l'exposition est justement assurée par le sous-système Windows, non fonctionnel à ce stade, Smss se présente sous les traits d'une application native, et n'utilise par conséquent que des API fondamentales de l'exécutif.

Processus système initial

Le processus système initial (baptisé System) est le réceptacle d'un genre particulier de threads, exécutés uniquement en mode noyau, auxquels il offre pour le compte du système d'exploitation des territoires communs, notamment l'espace d'adressage. (Plus de détails sur de tels entités, consultez la section Threads système du chapitre Processus et threads).

La variable noyau PsInitialSystemProcess mémorise l'adresse du bloc EPROCESS du processus System. La routine PsIsSystemProcess s'appuie sur ledit enregistrement pour vérifier qu'un processus est ou non le processus System.

Sous-système Windows

Environnement principal du système d'exploitation homonyme, le sous-système Windows exécute les applications, gère le clavier, la souris, l'écran et les dispositifs graphiques. Occupant une place spéciale parmi les autres environnements, le sous-système Windows remplit un rôle double, en ce sens qu'il fournit un support aux applications spécifiquement conçues pour Windows, et en plus, via gestion de divers aspects de l'expérience graphique, endosse l'identité visuelle des plateformes développées par Microsoft. Autre particularité majeure, il est à la base de tous les autres sous-systèmes.

Les composants clé du sous-système Windows sont les suivants :

- Le processus du sous-système d'environnement (Csrss) administre l'état des applications exécutées sous son contrôle - soit, en réalité, l'ensemble des programmes exécutés en mode utilisateur. Il gère les fenêtres console (texte), la création et suppression des processus et des threads, les portions du support des autres sous-systèmes, et diverses autres fonctions.

- Le pilote de périphérique mode noyau (Win32k.sys) héberge le gestionnaire de fenêtres, qui contrôle le fenêtrage (fenêtres, menus, etc.), les affichages écran et les saisies clavier-souris (et autres périphériques). Il contient également GDI (Graphics Device Interface), est une bibliothèque de fonctions pour périphériques de sortie graphique.

- Les DLL de sous-système, telles Kernelbase.dll, Kernel32.dll, User32.dll et Advapi32.dll, implémentent les fonctions de l'API Windows et mettent en œuvre des interactions générales avec le processus du sous-système

d'environnement. Elles traduisent les fonctions de l'API Windows en appels aux services système, exécutés en mode noyau, de Ntoskrnl.exe, et Win32k.sys.

Le processus de support via lequel apparaît le sous-système Windows s'appelle en interne Client/Server Run-Time Subsystem, ceci en référence au mode de communication à travers duquel il était initialement prévu que les applications communiquent. Dans la conception originelle de Windows, tous les sous-systèmes devaient être exécutés en tant que threads d'un processus de sous-système d'environnement commun au système, Csrss.exe. On fit ensuite des sous-systèmes POSIX et OS/2 des processus à part entière, sans pour autant modifier le nom de fichier du processus du sous-système Windows.

Le sous-système Windows classe les applications en deux catégories : les applications graphiques et celles fondées sur du texte (mode console). Aux applications graphiques est offert un ensemble d'objets et de contrôles standards (fenêtres, boutons, etc.) autour desquels construire et orienter le dialogue homme-machine. Une application à interface caractère, à laquelle le sous-système Windows fournit une représentation graphique, a sa sortie interactive dirigée vers une fenêtre texte. Chaque fois qu'une routine de sortie est appelée, le sous-système Windows appelle une routine pour afficher le texte dans la console associée. Cette façon de faire fonctionne pour toutes les applications en ligne de commande.

Du fait de sa proximité avec les services de l'exécutif natif du système, et de son implication avec l'affichage graphique, le sous-système Windows est indispensable au bon fonctionnement de Windows. Sa présence étant obligatoire, les informations de démarrage le concernant le veulent toujours actif, alors que les autres sous-systèmes sont configurés pour être exécutés à la demande. Pour plus d'informations sur comment Windows démarre un sous-système, consultez la section Démarrage de sous-système.

Graphics Device Interface (GDI)

Jonction entre un système d'exploitation essentiellement graphique et des dispositifs d'affichage et systèmes d'impression de nature diverses, Graphics Device Interface (GDI) est une interface de Microsoft Windows pour la représentation d'objets graphiques ainsi que pour leur transmission aux périphériques de sortie, typiquement un écran ou une imprimante.

La fonction principale de GDI est le dessin d'éléments visuels primitif, à savoir pixels, lignes, polygones et bitmaps. (Un pixel est le plus petit élément constituant une image graphique ou une surface d'affichage ; une ligne est un ensemble de points arrangés de façon contiguë ; un polygone est un ensemble de lignes, placées bout à bout de manière à former une figure contiguë et fermée ; un bitmap est un ensemble de pixels arrangés dans une configuration rectangulaire, le tout constituant une forme, une image ou une texture reconnaissable.) GDI contient des fonctions de dessin de traits, de textes et de figures et des fonctions de gestion des palettes. En revanche, il n'est pas chargé de l'affichage des fenêtres, des menus et autres, cette tâche étant assignée au gestionnaire de fenêtres.

L'une des particularités remarquables du système GDI est son indépendance vis à vis des dispositifs physiques, cela au sens d'être capable de s'adapter à une gamme très étendue de combinaisons matérielles, incluant par exemple cartes graphiques et imprimantes. Avec cet objectif d'indépendance en vue, GDI fournit des mécanismes au travers desquels créer du contenu graphique sans considération pour le type, donc les caractéristiques, du matériel utilisé. Il devient alors simple, avec les mêmes instructions, de dessiner à l'écran, ou d'imprimer sur du papier. GDI connait également des moyens de stockage standardisés pour les images sur disque ou sur écran.

S'il est certain que l'émancipation de GDI en regard du matériel est un atout, une conséquence directe de cette indépendance est le manque relatif de performances au niveau de la vitesse d'exécution. En outre, GDI ne sait pas produire correctement des animations et ne supporte aucune fonctionnalité 3D, contrairement aux API DirectX et OpenGL, conçues pour exposer les fonctions matérielles 3D des cartes graphiques.

GDI est un composant mode noyau (win32k.sys), élevées au rang d'entités privilégiées afin de s'adresser directement au matériel et diminuer les temps de réaction de l'IHM. Complétant la politique globale du noyau quant à la couche matérielle, le système GDI offre des capacités de représentation abstraite des périphériques cibles et assure la cohérence du dessin sur des interfaces multiples.

SYSTEM_INFORMATION_CLASS

Tableau 3.2. SYSTEM_INFORMATION_CLASS

ID	Classe	Type de données associé
0x0000	SystemBasicInformation	SYSTEM_BASIC_INFORMATION
0x0001	SystemProcessorInformation	SYSTEM_PROCESSOR_INFORMATION
0x0002	SystemPerformanceInformation	SYSTEM_PERFORMANCE_INFORMATION
0x0003	SystemTimeOfDayInformation	SYSTEM_TIMEOFDAY_INFORMATION
0x0004	SystemPathInformation	
0x0005	SystemProcessInformation	SYSTEM_PROCESS_INFORMATION
0x0006	SystemCallCountInformation	
0x0007	SystemDeviceInformation	SYSTEM_DEVICE_INFORMATION
0x0008	SystemProcessorPerformanceInformation	SYSTEM_PROCESSOR_PERFORMANCE_-INFORMATION
0x0009	SystemFlagsInformation	SYSTEM_FLAGS_INFORMATION
0x000A	SystemCallTimeInformation	
0x000B	SystemModulesInformation	RTL_PROCESS_MODULES
0x000C	SystemLocksInformation	RTL_PROCESS_LOCKS
0x000D	SystemStackTraceInformation	
0x000E	SystemPagedPoolInformation	
0x000F	SystemNonPagedPoolInformation	
0x0010	SystemHandleInformation	SYSTEM_HANDLE_INFORMATION
0x0011	SystemObjectInformation	SYSTEM_OBJECT_INFORMATION, SYSTEM_-OBJECTTYPE_INFORMATION
0x0012	SystemPageFileInformation	SYSTEM_PAGEFILE_INFORMATION
0x0013	SystemVdmInstemulInformation	
0x0014	SystemVdmBopInformation	
0x0015	SystemFileCacheInformation	SYSTEM_FILECACHE_INFORMATION
0x0016	SystemPoolTagInformation	SYSTEM_POOLTAG_INFORMATION
0x0017	SystemInterruptInformation	SYSTEM_INTERRUPT_INFORMATION
0x0018	SystemDpcBehaviorInformation	SYSTEM_DPC_BEHAVIOR_INFORMATION
0x0019	SystemFullMemoryInformation	SYSTEM_MEMORY_INFORMATION
0x001A	SystemLoadGdiDriverInformation	SYSTEM_GDI_DRIVER_INFORMATION
0x001B	SystemUnloadGdiDriverInformation	
0x001C	SystemTimeAdjustmentInformation	SYSTEM_QUERY_TIME_ADJUST_-INFORMATION, SYSTEM_SET_TIME_-ADJUST_INFORMATION
0x001D	SystemSummaryMemoryInformation	
0x001E	SystemMirrorMemoryInformation	
0x001F	SystemPerformanceTraceInformation	
0x0020	SystemCrashDumpInformation	
0x0021	SystemExceptionInformation	SYSTEM_EXCEPTION_INFORMATION
0x0022	SystemCrashDumpStateInformation	
0x0023	SystemKernelDebuggerInformation	SYSTEM_KERNEL_DEBUGGER_-INFORMATION
0x0024	SystemContextSwitchInformation	SYSTEM_CONTEXT_SWITCH_-INFORMATION
0x0025	SystemRegistryQuotaInformation	SYSTEM_REGISTRY_QUOTA_INFORMATION
0x0026	SystemExtendServiceTableInformation	
0x0027	SystemPrioritySeperation	
0x0028	SystemVerifierAddDriverInformation	

ID	Classe	Type de données associé
0x0029	SystemVerifierRemoveDriverInformation	
0x002A	SystemProcessorIdleInformation	SYSTEM_PROCESSOR_IDLE_INFORMATION
0x002B	SystemLegacyDriverInformation	
0x002C	SystemCurrentTimeZoneInformation	RTL_TIME_ZONE_INFORMATION
0x002D	SystemLookasideInformation	
0x002E	SystemTimeSlipNotification	
0x002F	SystemSessionCreate	
0x0030	SystemSessionDetach	
0x0031	SystemSessionInformation	
0x0032	SystemRangeStartInformation	
0x0033	SystemVerifierInformation	SYSTEM_VERIFIER_INFORMATION
0x0034	SystemVerifierThunkExtend	
0x0035	SystemSessionProcessInformation	SYSTEM_SESSION_PROCESS_-INFORMATION
0x0036	SystemLoadGdiDriverInSystemSpace	
0x0037	SystemNumaProcessorMap	SYSTEM_NUMA_INFORMATION
0x0038	SystemPrefetcherInformation	
0x0039	SystemExtendedProcessInformation	
0x003A	SystemRecommendedSharedDataAlignment	
0x003B	SystemComPlusPackage	
0x003C	SystemNumaAvailableMemory	
0x003D	SystemProcessorPowerInformation	SYSTEM_PROCESSOR_POWER_-INFORMATION
0x003E	SystemEmulationBasicInformation	SYSTEM_BASIC_INFORMATION
0x003F	SystemEmulationProcessorInformation	SYSTEM_PROCESSOR_INFORMATION
0x0040	SystemExtendedHandleInformation	SYSTEM_HANDLE_INFORMATION_EX
0x0041	SystemLostDelayedWriteInformation	
0x0042	SystemBigPoolInformation	SYSTEM_BIGPOOL_INFORMATION
0x0043	SystemSessionPoolTagInformation	SYSTEM_SESSION_PROCESS_-INFORMATION
0x0044	SystemSessionMappedViewInformation	SYSTEM_SESSION_MAPPED_VIEW_-INFORMATION
0x0045	SystemHotpatchInformation	
0x0046	SystemObjectSecurityMode	
0x0047	SystemWatchdogTimerHandler	
0x0048	SystemWatchdogTimerInformation	
0x0049	SystemLogicalProcessorInformation	
0x004A	SystemWow64SharedInformationObsolete	
0x004B	SystemRegisterFirmwareTableInformationHandler	
0x004C	SystemFirmwareTableInformation	
0x004D	SystemModuleInformationEx	RTL_PROCESS_MODULES
0x004E	SystemVerifierTriageInformation	
0x004F	SystemSuperfetchInformation	
0x0050	SystemMemoryListInformation	
0x0051	SystemFileCacheInformationEx	
0x0052	SystemThreadPriorityClientIdInformation	
0x0053	SystemProcessorIdleCycleTimeInformation	
0x0054	SystemVerifierCancellationInformation	

ID	Classe	Type de données associé
0x0055	SystemProcessorPowerInformationEx	SYSTEM_FILECACHE_INFORMATION
0x0056	SystemRefTraceInformation	
0x0057	SystemSpecialPoolInformation	
0x0058	SystemProcessIdInformation	SYSTEM_PROCESS_ID_INFORMATION
0x0059	SystemErrorPortInformation	
0x005A	SystemBootEnvironmentInformation	SYSTEM_BOOT_ENVIRONMENT_-INFORMATION
0x005B	SystemHypervisorInformation	
0x005C	SystemVerifierInformationEx	
0x005D	SystemTimeZoneInformation	
0x005E	SystemImageFileExecutionOptionsInformation	
0x005F	SystemCoverageInformation	
0x0060	SystemPrefetchPatchInformation	
0x0061	SystemVerifierFaultsInformation	
0x0062	SystemSystemPartitionInformation	
0x0063	SystemSystemDiskInformation	
0x0064	SystemProcessorPerformanceDistribution	
0x0065	SystemNumaProximityNodeInformation	
0x0066	SystemDynamicTimeZoneInformation	
0x0067	SystemCodeIntegrityInformation	
0x0068	SystemProcessorMicrocodeUpdateInformation	
0x0069	SystemProcessorBrandString	
0x006A	SystemVirtualAddressInformation	
0x006B	SystemLogicalProcessorAndGroupInformation	
0x006C	SystemProcessorCycleTimeInformation	
0x006D	SystemStoreInformation	
0x006E	SystemRegistryAppendString	
0x006F	SystemAitSamplingValue	
0x0070	SystemVhdBootInformation	
0x0071	SystemCpuQuotaInformation	PS_CPU_QUOTA_QUERY_INFORMATION
0x0072	SystemNativeBasicInformation	
0x0073	SystemErrorPortTimeouts	
0x0074	SystemLowPriorityIoInformation	
0x0075	SystemBootEntropyInformation	
0x0076	SystemVerifierCountersInformation	
0x0077	SystemPagedPoolInformationEx	
0x0078	SystemSystemPtesInformationEx	
0x0079	SystemNodeDistanceInformation	
0x007A	SystemAcpiAuditInformation	
0x007B	SystemBasicPerformanceInformation	SYSTEM_BASIC_PERFORMANCE_-INFORMATION
0x007C	SystemQueryPerformanceCounterInformation	
0x007D	SystemSessionBigPoolInformation	
0x007E	SystemBootGraphicsInformation	
0x007F	SystemScrubPhysicalMemoryInformation	
0x0080	SystemBadPageInformation	
0x0081	SystemProcessorProfileControlArea	
0x0082	SystemCombinePhysicalMemoryInformation	

ID	Classe	Type de données associé
0x0083	SystemEntropyInterruptTimingInformation	
0x0084	SystemConsoleInformation	
0x0085	SystemPlatformBinaryInformation	
0x0086	SystemThrottleNotificationInformation	
0x0087	SystemHypervisorProcessorCountInformation	
0x0088	SystemDeviceDataInformation	
0x0089	SystemDeviceDataEnumerationInformation	
0x008A	SystemMemoryTopologyInformation	
0x008B	SystemMemoryChannelInformation	
0x008C	SystemBootLogoInformation	
0x008D	SystemProcessorPerformanceInformationEx	SYSTEM_PROCESSOR_PERFORMANCE_-INFORMATION_EX
0x008E	SystemSpare0	
0x008F	SystemSecureBootPolicyInformation	
0x0090	SystemPageFileInformationEx	
0x0091	SystemSecureBootInformation	
0x0092	SystemEntropyInterruptTimingRawInformation	
0x0093	SystemPortableWorkspaceEfiLauncherInformation	
0x0094	SystemFullProcessInformation	
0x0095	SystemKernelDebuggerInformationEx	
0x0096	SystemBootMetadataInformation	
0x0097	SystemSoftRebootInformation	
0x0098	SystemElamCertificateInformation	
0x0099	SystemOfflineDumpConfigInformation	
0x009a	SystemProcessorFeaturesInformation	SYSTEM_PROCESSOR_FEATURES_-INFORMATION
0x009b	SystemRegistryReconciliationInformation	

SYSTEM_BASIC_INFORMATION.

```
typedef struct _SYSTEM_BASIC_INFORMATION {
    ULONG Reserved;
    ULONG TimerResolution;
    ULONG PageSize;
    ULONG NumberOfPhysicalPages;
    ULONG LowestPhysicalPageNumber;
    ULONG HighestPhysicalPageNumber;
    ULONG AllocationGranularity;
    ULONG MinimumUserModeAddress;
    ULONG MaximumUserModeAddress;
    KAFFINITY ActiveProcessorsAffinityMask;
    CHAR NumberOfProcessors;
} SYSTEM_BASIC_INFORMATION, *PSYSTEM_BASIC_INFORMATION;
```

Tableau 3.3. Valeurs initiales des champs de la structure SYSTEM_BASIC_INFORMATION

Champ	Valeur initiale
ActiveProcessorsAffinityMask	KeActiveProcessors
HighestPhysicalPageNumber	MmHighestPhysicalPage
LowestPhysicalPageNumber	MmLowestPhysicalPage
MaximumUserModeAddress	MmHighestUserAddress
NumberOfProcessors	KeNumberProcessors
NumberOfPhysicalPages	MmNumberOfPhysicalPages
TimerResolution	KeMaximumIncrement

SYSTEM_BOOT_ENVIRONMENT_INFORMATION.

```
typedef struct _SYSTEM_BOOT_ENVIRONMENT_INFORMATION
{
    GUID BootIdentifier;
    FIRMWARE_TYPE FirmwareType;
    ULONGLONG BootFlags;
} SYSTEM_BOOT_ENVIRONMENT_INFORMATION, PSYSTEM_BOOT_ENVIRONMENT_INFORMATION;
```

*Tableau 3.4. Valeurs initiales des champs de la structure **SYSTEM_BOOT_ENVIRONMENT_INFORMATION***

Champ	Valeur initiale
BootFlags	LOADER_PARAMETER_EXTENSION BootFlags
BootIdentifier	LOADER_PARAMETER_EXTENSION BootIdentifier

SYSTEM_PROCESSOR_INFORMATION.

```
typedef struct _SYSTEM_PROCESSOR_INFORMATION {
    USHORT ProcessorArchitecture;
    USHORT ProcessorLevel;
    USHORT ProcessorRevision;
    USHORT MaximumProcessors;
    ULONG ProcessorFeatureBits;
} SYSTEM_PROCESSOR_INFORMATION, *PSYSTEM_PROCESSOR_INFORMATION;
```

- **Nombre de processeurs (MaximumProcessors)** Nombre total de processeurs logiques disponibles sur le système.
- **Architecture matérielle (ProcessorArchitecture)** Architecture des processeurs. Si l'on s'en tient aux versions de Windows conçues pour les architectures x86 (32 bits) et x64 (64 bits), les valeurs possibles pour cet attribut sont `PROCESSOR_ARCHITECTURE_INTEL` (0x0) et `PROCESSOR_ARCHITECTURE_AMD64` (0x9). Voir fonction GetSystemInfo ; attribut wProcessorArchitecture de structure SYSTEM_INFO ; variable noyau KeProcessorArchitecture.
- **Fonctionnalités (ProcessorFeatureBits)** Informations sur les fonctionnalités matérielles des processeurs. Chaque bit parmi ce champ peut indiquer la présence ou l'absence d'une fonctionnalité particulière, comme la virtualisation, la prise en charge de certaines instructions, etc. Voir attribut FeatureBits de structure KPRCB, variable noyau KeFeatureBits, structure `SYSTEM_PROCESSOR_FEATURES_INFORMATION`.
- **Niveau de processeur (ProcessorLevel)** Génération spécifique de processeurs partagant des caractéristiques similaires au sein d'une architecture donnée. Voir fonction GetSystemInfo ; attribut CpuType de structure KPRCB, attribut wProcessorLevel de structure SYSTEM_INFO ; variable noyau KeProcessorLevel.
- **Révision de processeur (ProcessorRevision)** Caractéristiques spécifiques du processeur. Voir fonction GetSystemInfo ; attribut wProcessorRevision de structure `SYSTEM_INFO` ; variable noyau KeProcessorRevision. Voir KPRCB CpuStep.

Tableau 3.5. Valeurs initiales des champs de la structure SYSTEM_PROCESSOR_INFORMATION

Champ	Valeur initiale
MaximumProcessors	-
ProcessorArchitecture	KeProcessorArchitecture
ProcessorFeatureBits	KeFeatureBits
ProcessorLevel	KeProcessorLevel
ProcessorRevision	KeProcessorRevision

SYSTEM_PERFORMANCE_INFORMATION.

```
typedef struct _SYSTEM_PERFORMANCE_INFORMATION {
    LARGE_INTEGER IdleProcessTime;
    LARGE_INTEGER IoReadTransferCount;
    LARGE_INTEGER IoWriteTransferCount;
    LARGE_INTEGER IoOtherTransferCount;
    ULONG IoReadOperationCount;
```

```
ULONG IoWriteOperationCount;
ULONG IoOtherOperationCount;
ULONG AvailablePages;
ULONG CommittedPages;
ULONG CommitLimit;
ULONG PeakCommitment;
ULONG PageFaultCount;
ULONG CopyOnWriteCount;
ULONG TransitionCount;
ULONG CacheTransitionCount;
ULONG DemandZeroCount;
ULONG PageReadCount;
ULONG PageReadIoCount;
ULONG CacheReadCount;
ULONG CacheIoCount;
ULONG DirtyPagesWriteCount;
ULONG DirtyWriteIoCount;
ULONG MappedPagesWriteCount;
ULONG MappedWriteIoCount;
ULONG PagedPoolPages;
ULONG NonPagedPoolPages;
ULONG PagedPoolAllocs;
ULONG PagedPoolFrees;
ULONG NonPagedPoolAllocs;
ULONG NonPagedPoolFrees;
ULONG FreeSystemPtes;
ULONG ResidentSystemCodePage;
ULONG TotalSystemDriverPages;
ULONG TotalSystemCodePages;
ULONG NonPagedPoolLookasideHits;
ULONG PagedPoolLookasideHits;
ULONG AvailablePagedPoolPages;
ULONG ResidentSystemCachePage;
ULONG ResidentPagedPoolPage;
ULONG ResidentSystemDriverPage;
ULONG CcFastReadNoWait;
ULONG CcFastReadWait;
ULONG CcFastReadResourceMiss;
ULONG CcFastReadNotPossible;
ULONG CcFastMdlReadNoWait;
ULONG CcFastMdlReadWait;
ULONG CcFastMdlReadResourceMiss;
ULONG CcFastMdlReadNotPossible;
ULONG CcMapDataNoWait;
ULONG CcMapDataWait;
ULONG CcMapDataNoWaitMiss;
ULONG CcMapDataWaitMiss;
ULONG CcPinMappedDataCount;
ULONG CcPinReadNoWait;
ULONG CcPinReadWait;
ULONG CcPinReadNoWaitMiss;
ULONG CcPinReadWaitMiss;
ULONG CcCopyReadNoWait;
ULONG CcCopyReadNoWaitMiss;
ULONG CcCopyReadWaitMiss;
ULONG CcMdlReadNoWait;
ULONG CcMdlReadWait;
ULONG CcMdlReadNoWaitMiss;
ULONG CcMdlReadWaitMiss;
ULONG CcReadAheadIos;
ULONG CcLazyWriteIos;
ULONG CcLazyWritePages;
ULONG CcDataFlushes;
ULONG CcDataPages;
ULONG ContextSwitches;
```

```
    ULONG FirstLevelTbFills;
    ULONG SecondLevelTbFills;
    ULONG SystemCalls;
} SYSTEM_PERFORMANCE_INFORMATION, *PSYSTEM_PERFORMANCE_INFORMATION;
```

Tableau 3.6. Valeurs initiales des champs de la structure SYSTEM_PERFORMANCE_INFORMATION

Champ	Valeur initiale
AvailablePages	MmAvailablePages
CacheIoCount	KPRCB MmCacheIoCount
CacheReadCount	KPRCB MmCacheReadCount
CacheTransitionCount	KPRCB MmCacheTransitionCount
CommitLimit	MmTotalCommitLimit
CommittedPages	MmTotalCommittedPages
ContextSwitches	KPRCB KeContextSwitches
CopyOnWriteCount	KPRCB MmCopyOnWriteCount
DemandZeroCount	KPRCB MmDemandZeroCount
DirtyPagesWriteCount	KPRCB MmDirtyPagesWriteCount
DirtyWriteIoCount	KPRCB MmDirtyWriteIoCount
MappedPagesWriteCount	KPRCB MmMappedPagesWriteCount
MappedWriteIoCount	KPRCB MmMappedWriteIoCount
PageFaultCount	KPRCB MmPageFaultCount
PageReadCount	KPRCB MmPageReadCount
PageReadIoCount	KPRCB MmPageReadIoCount
PeakCommitment	MmPeakCommitment
ResidentPagedPoolPage	MmPagedPoolPage
ResidentSystemCodePage	MmSystemCodePage
ResidentSystemCachePage	MmSystemCachePage
ResidentSystemDriverPage	MmSystemDriverPage
SystemCalls	KPRCB KeSystemCalls
TransitionCount	KPRCB MmTransitionCount

```
typedef struct _SYSTEM_TIMEOFDAY_INFORMATION {
    LARGE_INTEGER BootTime;
    LARGE_INTEGER CurrentTime;
    LARGE_INTEGER TimeZoneBias;
    ULONG TimeZoneId;
    ULONG Reserved;
    ULONGLONG BootTimeBias;
    ULONGLONG SleepTimeBias;
} SYSTEM_TIMEOFDAY_INFORMATION, *PSYSTEM_TIMEOFDAY_INFORMATION;
```

Tableau 3.7. Valeurs initiales des champs de la structure SYSTEM_TIMEOFDAY_INFORMATION

Champ	Valeur initiale
CurrentTime	KeQuerySystemTime
BootTime	KeBootTime
TimeZoneBias	ExpTimeZoneBias
TimeZoneId	ExpCurrentTimeZoneId
BootTimeBias	KeBootTimeBias

```
typedef struct _SYSTEM_PROCESSOR_IDLE_INFORMATION {
    ULONGLONG IdleTime;
    ULONGLONG C1Time;
    ULONGLONG C2Time;
    ULONGLONG C3Time;
    ULONG     C1Transitions;
```

```
    ULONG     C2Transitions;
    ULONG     C3Transitions;
    ULONG     Padding;
} SYSTEM_PROCESSOR_IDLE_INFORMATION, *PSYSTEM_PROCESSOR_IDLE_INFORMATION;
```

Tableau 3.8. Valeurs initiales des champs de la structure SYSTEM_PROCESSOR_IDLE_INFORMATION

Champ	Valeur initiale
IdleTime	KPRCB IdleThread.KernelTime
C1Time	KPCB PowerState.TotalIdleStateTime[0]
C2Time	KPCB PowerState.TotalIdleStateTime[1]
C3Time	KPCB PowerState.TotalIdleStateTime[2]
C1Transitions	KPCB PowerState.TotalIdleTransitions[0]
C2Transitions	KPCB PowerState.TotalIdleTransitions[1]
C3Transitions	KPCB PowerState.TotalIdleTransitions[2]

SYSTEM_DEVICE_INFORMATION.

```
typedef struct _SYSTEM_DEVICE_INFORMATION {
    ULONG NumberOfDisks;
    ULONG NumberOfFloppies;
    ULONG NumberOfCdRoms;
    ULONG NumberOfTapes;
    ULONG NumberOfSerialPorts;
    ULONG NumberOfParallelPorts;
} SYSTEM_DEVICE_INFORMATION, *PSYSTEM_DEVICE_INFORMATION;
```

SYSTEM_PROCESSOR_PERFORMANCE_INFORMATION.

```
typedef struct _SYSTEM_PROCESSOR_PERFORMANCE_INFORMATION {
{
    LARGE_INTEGER IdleTime;
    LARGE_INTEGER KernelTime;
    LARGE_INTEGER UserTime;
    LARGE_INTEGER DpcTime;
    LARGE_INTEGER InterruptTime;
    ULONG InterruptCount;
} SYSTEM_PROCESSOR_PERFORMANCE_INFORMATION, *PSYSTEM_PROCESSOR_PERFORMANCE_INFORMATION;
```

Tableau 3.9. Valeurs initiales des champs de la structure SYSTEM_PROCESSOR_IDLE_INFORMATION

Champ	Valeur initiale
DpcTime	KPRCB DpcTime
IdleTime	KPRCB IdleThread.KernelTime
InterruptCount	KPRCB InterruptCount
InterruptTime	KPRCB InterruptTime
KernelTime	KPRCB KernelTime
UserTime	KPRCB UserTime

SYSTEM_QUERY_TIME_ADJUST_INFORMATION.

```
typedef struct _SYSTEM_QUERY_TIME_ADJUST_INFORMATION {
    ULONG TimeAdjustment;
    ULONG TimeIncrement;
    BOOLEAN Enable;
} SYSTEM_QUERY_TIME_ADJUST_INFORMATION, *PSYSTEM_QUERY_TIME_ADJUST_INFORMATION;
```

SYSTEM_SET_TIME_ADJUST_INFORMATION.

```
typedef struct _SYSTEM_SET_TIME_ADJUST_INFORMATION {
    ULONG TimeAdjustment;
    BOOLEAN Enable;
```

```
} SYSTEM_SET_TIME_ADJUST_INFORMATION, *PSYSTEM_SET_TIME_ADJUST_INFORMATION;

typedef struct _SYSTEM_PROCESS_INFORMATION
{
    ULONG NextEntryOffset;
    ULONG NumberOfThreads;
    LARGE_INTEGER WorkingSetPrivateSize;
    ULONG HardFaultCount;
    ULONG NumberOfThreadsHighWatermark;
    ULONGLONG CycleTime;
    LARGE_INTEGER CreateTime;
    LARGE_INTEGER UserTime;
    LARGE_INTEGER KernelTime;
    UNICODE_STRING ImageName;
    KPRIORITY BasePriority;
    HANDLE UniqueProcessId;
    HANDLE InheritedFromUniqueProcessId;
    ULONG HandleCount;
    ULONG SessionId;
    ULONG_PTR UniqueProcessKey;
    SIZE_T PeakVirtualSize;
    SIZE_T VirtualSize;
    ULONG PageFaultCount;
    SIZE_T PeakWorkingSetSize;
    SIZE_T WorkingSetSize;
    SIZE_T QuotaPeakPagedPoolUsage;
    SIZE_T QuotaPagedPoolUsage;
    SIZE_T QuotaPeakNonPagedPoolUsage;
    SIZE_T QuotaNonPagedPoolUsage;
    SIZE_T PagefileUsage;
    SIZE_T PeakPagefileUsage;
    SIZE_T PrivatePageCount;
    LARGE_INTEGER ReadOperationCount;
    LARGE_INTEGER WriteOperationCount;
    LARGE_INTEGER OtherOperationCount;
    LARGE_INTEGER ReadTransferCount;
    LARGE_INTEGER WriteTransferCount;
    LARGE_INTEGER OtherTransferCount;
} SYSTEM_PROCESS_INFORMATION, *PSYSTEM_PROCESS_INFORMATION;
```

Tableau 3.10. Valeurs initiales des champs de la structure SYSTEM_PROCESS_INFORMATION

Champ	Valeur initiale
BasePriority	EPROCESS Pcb.BasePriority
CreateTime	EPROCESS CreateTime
KernelTime	EPROCESS Pcb.KernelTime
PeakVirtualSize	EPROCESS PeakVirtualSize
PageFaultCount	EPROCESS Vm.PageFaultCount
PagefileUsage	EPROCESS ProcessQuotaUsage[2] << PAGE_SHIFT
PeakPagefileUsage	EPROCESS ProcessQuotaPeak[2] << PAGE_SHIFT
PeakWorkingSetSize	EPROCESS Vm.PeakWorkingSetSize << PAGE_SHIFT
PrivatePageCount	EPROCESS CommitCharge << PAGE_SHIFT
QuotaPeakNonPagedPoolUsage	EPROCESS ProcessQuotaPeak[0]
QuotaPeakPagedPoolUsage	EPROCESS ProcessQuotaPeak[1]
QuotaPagedPoolUsage	EPROCESS ProcessQuotaUsage[1]
QuotaNonPagedPoolUsage	EPROCESS ProcessQuotaUsage[0]
SessionId	EPROCESS Session.SessionId
UserTime	EPROCESS Pcb.UserTime
VirtualSize	EPROCESS VirtualSize
WorkingSetSize	EPROCESS Vm.WorkingSetSize << PAGE_SHIFT

Champ	Valeur initiale
OtherOperationCount	EPROCESS OtherOperationCount
ReadOperationCount	EPROCESS ReadOperationCount
WriteOperationCount	EPROCESS WriteOperationCount
ReadTransferCount	EPROCESS ReadTransferCount
WriteTransferCount	EPROCESS WriteTransferCount
OtherTransferCount	EPROCESS OtherTransferCount

SYSTEM_FLAGS_INFORMATION.

```
typedef struct _SYSTEM_FLAGS_INFORMATION
{
    ULONG Flags;
} SYSTEM_FLAGS_INFORMATION, *PSYSTEM_FLAGS_INFORMATION;
```

SYSTEM_PROCESSOR_POWER_INFORMATION .

```
typedef struct _SYSTEM_PROCESSOR_POWER_INFORMATION
{
    UCHAR CurrentFrequency;
    UCHAR ThermalLimitFrequency;
    UCHAR ConstantThrottleFrequency;
    UCHAR DegradedThrottleFrequency;
    UCHAR LastBusyFrequency;
    UCHAR LastC3Frequency;
    UCHAR LastAdjustedBusyFrequency;
    UCHAR ProcessorMinThrottle;
    UCHAR ProcessorMaxThrottle;
    ULONG NumberOfFrequencies;
    ULONG PromotionCount;
    ULONG DemotionCount;
    ULONG ErrorCount;
    ULONG RetryCount;
    ULONGLONG CurrentFrequencyTime;
    ULONGLONG CurrentProcessorTime;
    ULONGLONG CurrentProcessorIdleTime;
    ULONGLONG LastProcessorTime;
    ULONGLONG LastProcessorIdleTime;
} SYSTEM_PROCESSOR_POWER_INFORMATION, *PSYSTEM_PROCESSOR_POWER_INFORMATION;
```

SYSTEM_PAGEFILE_INFORMATION.

```
typedef struct _SYSTEM_PAGEFILE_INFORMATION {
    ULONG NextEntryOffset;
    ULONG TotalSize;
    ULONG TotalInUse;
    ULONG PeakUsage;
    UNICODE_STRING PageFileName;
} SYSTEM_PAGEFILE_INFORMATION, *PSYSTEM_PAGEFILE_INFORMATION;
```

SYSTEM_FILECACHE_INFORMATION.

```
typedef struct _SYSTEM_FILECACHE_INFORMATION
{
    SIZE_T CurrentSize;
    SIZE_T PeakSize;
    ULONG PageFaultCount;
    SIZE_T MinimumWorkingSet;
    SIZE_T MaximumWorkingSet;
    SIZE_T CurrentSizeIncludingTransitionInPages;
    SIZE_T PeakSizeIncludingTransitionInPages;
    ULONG TransitionRePurposeCount;
    ULONG Flags;
```

```
} SYSTEM_FILECACHE_INFORMATION, *PSYSTEM_FILECACHE_INFORMATION;
```

SYSTEM_DPC_BEHAVIOR_INFORMATION.

```
typedef struct _SYSTEM_DPC_BEHAVIOR_INFORMATION
{
    ULONG Reserved;
    ULONG DpcQueueDepth;
    ULONG MinimumDpcRate;
    ULONG AdjustDpcThreshold;
    ULONG IdealDpcRate;
} SYSTEM_DPC_BEHAVIOR_INFORMATION, *PSYSTEM_DPC_BEHAVIOR_INFORMATION;
```

SYSTEM_INTERRUPT_INFORMATION.

```
typedef struct _SYSTEM_INTERRUPT_INFORMATION {
    ULONG ContextSwitches;
    ULONG DpcCount;
    ULONG DpcRate;
    ULONG TimeIncrement;
    ULONG DpcBypassCount;
    ULONG ApcBypassCount;
} SYSTEM_INTERRUPT_INFORMATION, *PSYSTEM_INTERRUPT_INFORMATION;
```

RTL_PROCESS_MODULES.

```
typedef struct _RTL_PROCESS_MODULES {
    ULONG NumberOfModules;
    RTL_PROCESS_MODULE_INFORMATION Modules[1];
} RTL_PROCESS_MODULES, *PRTL_PROCESS_MODULES;
```

RTL_PROCESS_MODULE_INFORMATION.

```
typedef struct _RTL_PROCESS_MODULE_INFORMATION {
    HANDLE Section;
    PVOID MappedBase;
    PVOID ImageBase;
    ULONG ImageSize;
    ULONG Flags;
    USHORT LoadOrderIndex;
    USHORT InitOrderIndex;
    USHORT LoadCount;
    USHORT OffsetToFileName;
    UCHAR  FullPathName[256];
} RTL_PROCESS_MODULE_INFORMATION, *PRTL_PROCESS_MODULE_INFORMATION;
```

SYSTEM_HANDLE_INFORMATION.

```
typedef struct _SYSTEM_HANDLE_INFORMATION {
    ULONG NumberOfHandles;
    SYSTEM_HANDLE_TABLE_ENTRY_INFO Handles[1];
} SYSTEM_HANDLE_INFORMATION, *PSYSTEM_HANDLE_INFORMATION;
```

SYSTEM_HANDLE_TABLE_ENTRY_INFO.

```
typedef struct _SYSTEM_HANDLE_TABLE_ENTRY_INFO {
    USHORT UniqueProcessId;
    USHORT CreatorBackTraceIndex;
    UCHAR ObjectTypeIndex;
    UCHAR HandleAttributes;
    USHORT HandleValue;
    PVOID Object;
    ULONG GrantedAccess;
} SYSTEM_HANDLE_TABLE_ENTRY_INFO, *PSYSTEM_HANDLE_TABLE_ENTRY_INFO;
```

```
typedef struct _SYSTEM_NUMA_INFORMATION
{
    ULONG HighestNodeNumber;
    ULONG Reserved;
    union
    {
        ULONGLONG ActiveProcessorsAffinityMask[MAXIMUM_NUMA_NODES];
        ULONGLONG AvailableMemory[MAXIMUM_NUMA_NODES];
    };
} SYSTEM_NUMA_INFORMATION, *PSYSTEM_NUMA_INFORMATION;

typedef struct _SYSTEM_OBJECTTYPE_INFORMATION
{
  ULONG NextEntryOffset;
  ULONG NumberOfObjects;
  ULONG NumberOfHandles;
  ULONG TypeIndex;
  ULONG InvalidAttributes;
  GENERIC_MAPPING GenericMapping;
  ULONG ValidAccessMask;
  ULONG PoolType;
  BOOLEAN SecurityRequired;
  BOOLEAN WaitableObject;
  UNICODE_STRING TypeName;
} SYSTEM_OBJECTTYPE_INFORMATION, *PSYSTEM_OBJECTTYPE_INFORMATION;

typedef struct _SYSTEM_OBJECT_INFORMATION
{
  ULONG NextEntryOffset;
  PVOID Object;
  HANDLE CreatorUniqueProcess;
  USHORT CreatorBackTraceIndex;
  USHORT Flags;
  LONG PointerCount;
  LONG HandleCount;
  ULONG PagedPoolCharge;
  ULONG NonPagedPoolCharge;
  HANDLE ExclusiveProcessId;
  PVOID SecurityDescriptor;
  OBJECT_NAME_INFORMATION NameInfo;
} SYSTEM_OBJECT_INFORMATION, *PSYSTEM_OBJECT_INFORMATION;

typedef struct _SYSTEM_MEMORY_INFO
{
  PUCHAR StringOffset;
  USHORT ValidCount;
  USHORT TransitionCount;
  USHORT ModifiedCount;
  USHORT PageTableCount;
} SYSTEM_MEMORY_INFO, *PSYSTEM_MEMORY_INFO;

typedef struct _SYSTEM_MEMORY_INFORMATION
{
  ULONG InfoSize;
  ULONG StringStart;
  SYSTEM_MEMORY_INFO Memory[1];
} SYSTEM_MEMORY_INFORMATION, *PSYSTEM_MEMORY_INFORMATION;

typedef struct _SYSTEM_GDI_DRIVER_INFORMATION
{
  UNICODE_STRING DriverName;
  PVOID ImageAddress;
  PVOID SectionPointer;
  PVOID EntryPoint;
  PIMAGE_EXPORT_DIRECTORY ExportSectionPointer;
  ULONG ImageLength;
```

```
} SYSTEM_GDI_DRIVER_INFORMATION, *PSYSTEM_GDI_DRIVER_INFORMATION;

typedef struct _SYSTEM_VERIFIER_INFORMATION
{
  ULONG NextEntryOffset;
  ULONG Level;
  UNICODE_STRING DriverName;
  ULONG RaiseIrqls;
  ULONG AcquireSpinLocks;
  ULONG SynchronizeExecutions;
  ULONG AllocationsAttempted;
  ULONG AllocationsSucceeded;
  ULONG AllocationsSucceededSpecialPool;
  ULONG AllocationsWithNoTag;
  ULONG TrimRequests;
  ULONG Trims;
  ULONG AllocationsFailed;
  ULONG AllocationsFailedDeliberately;
  ULONG Loads;
  ULONG Unloads;
  ULONG UnTrackedPool;
  ULONG CurrentPagedPoolAllocations;
  ULONG CurrentNonPagedPoolAllocations;
  ULONG PeakPagedPoolAllocations;
  ULONG PeakNonPagedPoolAllocations;
  SIZE_T PagedPoolUsageInBytes;
  SIZE_T NonPagedPoolUsageInBytes;
  SIZE_T PeakPagedPoolUsageInBytes;
  SIZE_T PeakNonPagedPoolUsageInBytes;
} SYSTEM_VERIFIER_INFORMATION, *PSYSTEM_VERIFIER_INFORMATION;

typedef struct _RTL_TIME_ZONE_INFORMATION
{
  LONG Bias;
  WCHAR StandardName[32];
  TIME_FIELDS StandardDate;
  LONG StandardBias;
  WCHAR DaylightName[32];
  TIME_FIELDS DaylightDate;
  LONG DaylightBias;
} RTL_TIME_ZONE_INFORMATION, *PRTL_TIME_ZONE_INFORMATION;

typedef struct _RTL_PROCESS_LOCK_INFORMATION
{
  PVOID Address;
  USHORT Type;
  USHORT CreatorBackTraceIndex;
  ULONG OwnerThreadId;
  ULONG ActiveCount;
  ULONG ContentionCount;
  ULONG EntryCount;
  ULONG RecursionCount;
  ULONG NumberOfSharedWaiters;
  ULONG NumberOfExclusiveWaiters;
} RTL_PROCESS_LOCK_INFORMATION, *PRTL_PROCESS_LOCK_INFORMATION;

typedef struct _RTL_PROCESS_LOCKS
{
  ULONG NumberOfLocks;
  RTL_PROCESS_LOCK_INFORMATION Locks[1];
} RTL_PROCESS_LOCKS, *PRTL_PROCESS_LOCKS;

typedef struct _SYSTEM_PROCESS_ID_INFORMATION {
    HANDLE ProcessId;
    UNICODE_STRING ImageName;
} SYSTEM_PROCESS_ID_INFORMATION, *PSYSTEM_PROCESS_ID_INFORMATION;
```

NUMA

Autre type de systèmes multi processeur reconnue par Windows : NUMA (*Non Uniform Memory Access*), laquelle topologie permet de regrouper les processeurs en unités de plus large échelle appelées noeuds. Chaque noeud a ses propres processeurs, canaux d'E/S et sa propre mémoire, et est relié au système global par l'intermédiaire d'un bus d'interconnexion.

Les systèmes NUMA sont dits non uniformes au sens où, vis à vis de chaque processeur, les temps d'accès diffèrent selon la mémoire accédée. Tous les processeurs de tous les noeuds peuvent accéder à toute la mémoire (avec un même espace d adressage), mais les accès à la mémoire du noeud sont plus rapides.

L'objectif fondamental suivi par NUMA est de pallier les limites des configurations SMP, dans lesquelles le couplage étroit entre les processeurs et le reste du système (en réalité la compétition que ce schéma implique, supportée du reste par un unique bus) augmente considérablement la charge sur l'interconnexion, et diminue les performances.

Tableau 3.11. Opérations concernant NUMA

Opération	Fonction	Service	Routine
Récupérer le noeud qui a le numéro le plus élevé	GetNumaHighestNodeNumber	NtQuerySystemInformation (SystemNumaProcessorMap)	KeQueryHighestNodeNumber
Récupérer le masque processeur d'un noeud	GetNumaNodeProcessorMask	NtQuerySystemInformation (SystemNumaProcessorMap)	
Récupérer le numéro de noeud d'un processeur	GetNumaProcessorNode	NtQuerySystemInformation (SystemNumaProcessorMap)	
Obtenir la quantité de mémoire disponible dans un noeud	GetNumaAvailableMemoryNode	NtQuerySystemInformation (SystemNumaAvailableMemory)	ExpQueryNumaAvailableMemory
Obtenir le numéro de noeud sur lequel s'exécute l'appelant			KeGetCurrentNodeNumber
Initialiser les structures noyau requises pour le support des systèmes NUMA.	s/o	s/o	KeNumaInitialize
Obtenir le masque d'affinité d'un noeud			KeQueryNodeActiveAffinity

Tableau 3.12. Variables noyau concernant NUMA

Variable	Type	Description
ExNode0		
KeNodeBlock	PKNODE	
KeNumberNodes	UCHAR	Voir routine KeQueryHighestNodeNumber

Hyperthread

Hyperthread est une technologie Intel qui créé plusieurs processeurs logiques (en l'occurrence deux) sur un même processeur physique. Chaque processeur a son propre état CPU (registres de données et de contrôle), mais le moteur d'exécution, le cache matériel et le bus système sont partagés. Cela permet à un coeur de processeur physique d'agir essentiellement tel un duo logique, et en définitive d'exécuter deux applications indépendantes en même temps.

Bien qu'hyperthread ne double pas les performances d'un système, il peut les optimiser en exploitant mieux les ressources oisives. Ainsi, un CPU logique peut mener de l'avant diverses opérations tandis que les autres CPU logiques sont occupés. Gardez cependant à l'esprit que les gains en performances apportées par Hyperthread dépendent étroitement du contexte applicatif.

Dans une configuration hyperthread, les processeurs logiques situés sur le même coeur possèdent des numéros de CPU consécutifs: les CPU 0 et 1 se trouvent tous deux sur le premier coeur, les CPU 2 et 3 sur le second coeur, et ainsi de suite.

Les algorithmes d'ordonnancement de Windows ont été conçus de telle sorte à tirer le meilleur parti des systèmes hyperthread, tout en prenant en compte leurs limites. Ainsi, considérant qu'une unité de calcul logique ne peut rivaliser avec une physique, Windows emploie hyperthread que si la charge de travail le justifie.

Indépendamment de la version de Windows qui anime l'ordinateur, les processeurs logiques, compte tenu de leur nature, n'entrent pas en ligne de compte au niveau de la licence d'utilisation. Ainsi, par exemple, sur un système hyperthread mono processeur, une version de Windows limitée à un processeur utilisera les deux processeurs logiques disponibles.

Multitraitement

Le multitraitement désigne l'aptitude du système d'exploitation à utiliser les processeurs présents dans la station de travail en vue de mener de l'avant l'exécution des threads. Dans le scénario idéal, Windows peut traiter simultanément autant de threads qu'il existe de processeurs sur la machine.

Windows est un système d'exploitation SMP (Symetric MultiProcessing), ce qui signifie que le système et les threads utilisateur peuvent prendre place sur n'importe quel processeur. Ce modèle s'oppose à un autre, à savoir ASMP, (ASymetric MultiProcessing), qui consiste à réserver un processeur à l'usage exclusif du système, les autres processeurs exécutant en ces circonstances le code utilisateur.

Les versions 32 bits de Windows peuvent gérer 32 processeurs, et les versions 64 bits 64 processeurs. (Deux nombres qui découlent directement de la taille native d'un mot machine au sein de ces architectures.) Dans les faits, le nombre réel de processeurs gérés dépend de l'édition de Windows qu'exécute l'ordinateur.

Les versions 64 bits de Windows apparues depuis Windows Vista et Windows Server 2008 s'appuient sur un noyau unifiée qui prend indistinctement en charge les systèmes mono processeur et les systèmes multi processeur. Comparativement aux versions antérieures de Windows, qui comportaient des versions distinctes du noyau et de la couche d'abstraction matérielle pour chaque type de machine, cette modification a pris place conformément au fait que la plus grande partie des configurations parmi le paysage informatique moderne ont au moins deux coeurs (logiques sinon physiques). Les versions 32 bits incorporent quand à elle toujours deux versions du noyau : une compatible PAE (avec en filigrane la prise en charge de la mémoire marquée non exécutable, qui dépend étroitement de ce support), et l'autre dont la fonctionnalité est absente.

Multitâche

Le multitâche est le procédé via lequel Windows (ainsi, du reste, que tout autre système d'exploitation) partage un processeur unique entre de multiples threads.

L'aptitude d'un système à fonctionner en traitement multitâche est indépendante du nombre de processeurs que renferme l'ordinateur. Une configuration multi processeur (ou multi coeur) n'est de ce fait pas nécessaire pour bénéficier d'une telle opportunité.

On distingue pour la mise en oeuvre du multitâche deux modèles : coopératif et préemptif, l'un comme l'autre relevant d'une approche distincte, dont ils sont en définitive la concrétisation.

Dans la perspective du multitâche coopératif, chaque application qui s'exécute doit explicitement libérer le processeur sur lequel elle est installée, permettant de la sorte à une autre application dans la file d'attente de disposer à son tour du processeur.

Les principaux atouts du multitâche coopératif consistent en une simplification des rapports entre système et applications d'un coté, et entre applications et ressources partagées d'un autre. Compte tenu dans cette configuration de la main-mise qu'a une application sur le processeur (application dont dépend, en toute fin, le prochain basculement basculement de contexte), il est dès lors plus facile de garantir l'indivisibilité des séquences de traitement (sections critiques). Si une interruption peut toujours survenir, la séquence d'instructions en cours reprendra automatiquement son exécution au point où elle a été interrompu. Le mode

coopératif présente cependant plusieurs inconvénients, le plus évident étant que rien ne garantit qu'une application ne mobilisera pas le processeur au détriment des autres. Pour cette raison, La forme coopérative est en pratique rarement utilisée (du moins pas sous son aspect le plus strict). Elle l'a été dans les produits Microsoft Windows jusque dans Windows 3.X.

Avec le multitâche préemptif, chaque application occupe le processeur pendant un laps de temps prédéfini ou jusqu'à ce que la priorité d'une autre application devienne supérieure à la sienne. Dans une tel scénario, l'attribution de temps processeur pour les applications est fait par le système d'exploitation, lui assurant ainsi un droit de préemption inaliénable.

Dans l'éventualité où tous les threads d'un processus ont la même charge de travail (ce qui n'est généralement pas le cas), le multitâche préemptif permet une répartition équitable du temps processeur (principe du temps partagé).

Chapitre 4. Mécanismes système

Particulièrement riche sur le plan des technologies et des services rendus afférant, Windows offre à cet égard kyrielle de dispositifs globaux sur lesquels s'appuient les composants mode noyau, tels que l'exécutif et les pilotes de périphérique, ainsi que ceux exécutés en mode utilisateur, y compris applications et processus de support. Dans ce chapitre, nous allons passer à la loupe les mécanismes système que voici :

- Flags Windows globaux
- Options d'exécution d'image
- Windows on Windows, incluant Wow32 et Wow64
- Windows Resource Protection

Windows on Windows

Windows on Windows, ou plus simplement Wow, désigne une infrastructure par laquelle le système d'exploitation présente ses services de manière analogue aux versions de Windows conçues par le passé, dont l'information élémentaire était traitée sous forme de donnes 16, puis 32 bits - cela conditionnant bon nombre de comportements internes et de surfaces logicielles exposées. C'est via l'approche Wow (et avec elle, ainsi que nous allons les voir dans cette section, diverses techniques) que Windows 32 bits est capable de reproduire les conditions attendues par des programmes écrits pour Windows 95 et Windows 98, voire Windows 3.1 (lui-même reposant sur MS-DOS), et, dans la même veine, que Windows 64 bits permet l'exécution d'applications x86 32 bits.

Historique de Windows on Windows. Windows évoluant au rythme des avancées technologiques et en fonction de la largeur des registres internes du processeur pour la manipulation des données, Wow a d'abord commencé en tant qu'appui aux programmes 16 bits exécutés dans Windows NT, premier système d'exploitation de la gamme à être entièrement 32 bits. Par la suite, avec l'introduction des processeurs 64 bits et la diminution subséquente de l'utilisation de l'informatique 32 bits (32 bits et 64 bits faisant ici référence non seulement à la façon dont le processeur traite les informations, mais aussi comment système et logiciels tirent profit de ces ressources), le terme Windows sur Windows a changé de connotation et se réfère aujourd'hui à la capacité du système d'exploitation 64 bits à exécuter côte à côte des processus Win32 et Win64. Notez, en ce qui concerne l'écosystème logiciel, que Wow vise moins à conserver l'existant qu'à préparer l'avenir. Que ce soit pour les utilisateurs, les concepteurs de logiciel ou les administrateurs système, il n'existe aucune raison de refuser les changements induits par le passage au 64 bits.

Wow32

Imitation des conditions attendues par les programmes écrits pour Windows 3.1, dernière version 16 bits d'un système d'exploitation Microsoft, et la dernière aussi à se reposer sur MS-DOS, l'environnement d'exécution Windows 16 bits est supporté par le biais d'une couche intermédiaire, dévolue à la compatibilité, appelée *Windows sur Windows* (WOW32, Windows on Windows 32-bits), laquelle intègre et étend les technologies permettant d'exécuter des programmes MS-DOS (NTVDM).

L'environnement Windows 16 bits est disponible dans les diverses éditions 32 bits de Microsoft Windows, dont Windows NT, 2000, XP, Serveur 2003, Vista, Serveur 2008, 7, 8 et 10. Les versions 64 bits de Windows, en revanche, n'incluent pas de couche d'émulation pour le code 16 bits, et ne peuvent donc pas exécuter des applications Win16.

Considérations sur la compatibilité 16 bits

Si les programmes 16 bits semblent a priori avoir disparus du paysage informatique contemporain (soit près de 20 ans après la sortie du premier Windows 32 bits), quelques cas persistent où ce type d'applications intervient encore. Certains jeux, par exemple, développés pour les anciens systèmes Microsoft peuvent ne pas fonctionner

correctement sur des systèmes d'exploitation plus récents. La sphère professionnelle, elle aussi, abrite encore potentiellement un certain nombre d'applications métiers héritées 16 bits.

Historiquement, les produits Windows de la famille 9x, basée sur Windows 95 et qui maintient un ancrage très fort sur MS-DOS, peuvent exécuter des applications 16 bits sans problèmes majeurs, ces systèmes s'appuyant en réalité sur un noyau hybride 16/32 bits. Avec Windows NT, Microsoft a implanté au sein de ses systèmes le dispositif Windows on Windows (WoW) qui permet de créer un environnement 16 bits pour les anciens programmes, cela alors que le système repose sur des fondations entièrement 32 bits.

Windows 64bits n'assure plus la compatibilité avec les programmes 16 bits. On aura donc recours pour l'occasion à des applications tierces. DOSBox, par exemple, est un émulateur simulant un environnement compatible DOS sous Windows dans le but d'exécuter des programmes conçus pour ce système. Autre exemple, VMware, une gamme de logiciels liés à la virtualisation d'architectures x86. Avec les systèmes 64 bits, le WoW original (Wow32) n'est plus présent, remplacé au même poste par WoW64, qui permet de lancer des applications 32 bits, très courantes, sur un système 64 bits.

Limitations et restrictions concernant Wow32

La machine virtuelle Windows 16 bits, et c'est son but, ressemble à Windows 3.1 par bien des aspects, mais présente du coup les mêmes limites. Une seule application Win16 peut fonctionner à la fois, les applications ne possèdent qu'un seul thread, elles résident dans le même espace d'adressage et partagent toutes la même file d'attente d'entrée. Cela implique qu'une application qui subit un dysfonctionnement peut affecter défavorablement toutes les autres applications s'exécutant dans le même environnement Win16. Par exemple, si une application ne libère pas le processeur, aucune autre ne peut y accéder, ou si une application bloque l'arrivée d'entrées, toutes les autres en sont privées. De plus, une application Win16 peut faire échouer toutes les autres en corrompant l'espace d'adressage commun.

La résolution des problèmes de comptabilité issus de certaines applications spécifiques 16 bits se trouvent à d'autres niveau que l'espace d'adressage ou le modèle mémoire utilisé, incluant les noms de fichiers longs (noms qui ne sont pas conforment à la convention 8.3), la multi utilisation ou le concept de moindre privilège. Eloignées du domaine de compétences de WOW32, ces questions ont pour quelques-unes une réponse, ou sont au moins (autant qu'il est possible de le faire) prises en compte. Par exemple, lorsque vous enregistrez un fichier avec un nom de fichier long vers un lecteur NTFS, NTFS crée, par défaut, une seconde entrée de répertoire de fichier avec un nom de fichier court conforme à la convention 8.3.

Si cette limite a moins le caractère critique des précédentes, il n'existe aucune mémoire partagée entre les applications qui s'exécutent dans le programme WOW et les autres applications sous Windows. Les applications Win16 ne peuvent pas appeler de fonctions incluses dans les bibliothèques 32 bits et les applications pour Windows ne peuvent pas appeler des DLL Win16.

Wow64

Les déclinaisons de Windows calibrées pour l'architecture matérielle x64 utilisent des adresses et le jeu d'instructions 64 bits de ces plateformes. Ainsi, et même en considérant l'inclusion dans la famille de processeurs x64 d'un support natif pour l'exécution d'applications x86 en utilisant les registres idoines (eax, ecx, et ainsi de suite), exécuter des programmes IA32 32 bits dans un environnement x64 64 bits nécessite une formule avec laquelle gommer les différences structurelles entre les deux architectures. Windows x64 implémente à cet effet une couche applicative distincte, nommée Windows on Windows, qui permet de faire fonctionner les programmes x86 32 bits dans le système d'exploitation 64 bits.

L'objectif premier de WOW64 est de fournir les interfaces requises pour permettre à des applications Windows 32 bits de s'exécuter sans être modifiées sur un système 64 bits. Cela inclut une couche de conversion des appels Win32 32 bits en appels 64 bits correspondants, plus divers aménagements en ce qui concerne la création des processus et des threads, la gestion des appels système ou le basculement du processeur entre les modes 32 bits et 64 bits.

Composants Wow64

Au plus haut niveau d'abstraction, l'environnement Wow64 se présente sous la forme d'une collection de modules (DLL) en mode utilisateur.

- Wow64.dll Fournit l'infrastructure de base sur laquelle repose la couche logicielle Wow64 ; gère la création des processus et des threads ; intercepte les appels système de base exportés par Ntoskrnl.exe ; implémente la redirection du système de fichiers, ainsi que la redirection et réflexion du registre.
- Wow64Cpu.dll Fait abstraction des caractéristiques du processeur hôte ; gère le contexte processeur de chaque thread exécuté dans Wow64 ; fournit le support, spécifique à l'architecture du processeur, du basculement du processeur entre les modes 32 bits et 64 bits.
- Wow64Win.dll Intercepte les appels système relatifs à la portion mode noyau du sous-système Windows, autrement dit les appels concernant l'interface graphique exportés par Win32k.sys.
- Ntvdm64.dll Accomplit diverses manœuvres et instructions dans le but de gérer les programmes d'installation 16 bits permettant l'intégration d'un logiciel 32 bits sur l'ordinateur.

Espace d'adressage des processus Wow64

Windows 64 bits installe le logiciel Wow64 dans l'espace d'adressage de tout processus né à partir d'une image exécutable (.exe) 32 bits. Les processus Wow64 peuvent fonctionner avec 2 ou 4 Go d'espace virtuel. Si l'application a dans son entête de l'image le flag d'adressage large a l'état actif, lui est offert un espace d'adressage de mémoire virtuelle de 4 Go. Dans le cas contraire, le gestionnaire de mémoire, par l'intermédiaire de Wow64, crée une fenêtre de mémoire virtuelle adressable en 32 bits, soit 2 Go, exactement comme si l'application fonctionnait sur une édition 32 bits de Windows. Evidement (mais il n'est peut-être pas inutile de le rappeler), malgré WoW64, les programmes 32 bits exécutées sous Windows 64 bits ne bénéficient ni de la taille de l'espace d'adressage 64 bits ni des registres élargis à 64 bits sur les processeurs.

Appels système

Le logiciel Wow64 instaure une présence dans les surfaces de contact entre la plateforme native et les applications héritées, y compris les appels système. Il intercepte pour ce faire tous les chemins d'exécution dans lesquels a lieu un basculement entre les modes 32 bits et 64 bits du processeur, où le code 32 bits sollicite des services implémentés au niveau du noyau 64 bits, ou où le système natif 64 bits appelle du code mode utilisateur 32 bits.

Lors de l'exécution d'un programme, Windows donne en premier lieu la main au code d'initialisation du chargeur (*loader*), lequel inspecte l'en-tête de l'image exécutable et, s'il la reconnaît en tant que 32 bits, met en place pour elle un environnement Wow64. Il charge pour ce faire Wow64.dll, faisant le lien entre le support 32 bits et le système d'exploitation 64 bits. Ensuite, Wow64 configure le contexte de démarrage à l'intérieur de Ntdll, bascule le processeur en mode 32 bits et commence l'exécution du chargeur 32 bits. A partir de là, étant donné que les processeurs x64 peuvent exécuter directement les instructions x86, l'exécution continue comme si le processus était exécuté sur un système 32 bits natif.

Wow64 embarque des versions 32 bits spéciales de quelques-unes des DLL fondamentales du sous-système Windows, incluant Kernel32.dll, User32.dll, Gdi32.dll. Ces modules, stockés dans le dossier \Windows \System32\Syswow64, sont pour la plupart fonctionnellement identiques à leurs homologues incorporés à Windows 32 bits. Une exception, toutefois, avec la bibliothèque de support Ntdll.dll, qui dans le contexte Wow64 s'occupe également de convertir les données 32 bits en vue de leur transmission vers les couches natives.

Paramètres d'entrée et de sortie des processus Wow64 opèrent a une échelle différente de celle des autres composants hébergés par le système natif. Ainsi, quand un tel processus réclame du traitement en mode noyau, Wow64 bascule le processeur en mode 64 bits, capture les paramètres au format 32 bits associés à l'appel système, puis émet l'appel 64 bits correspondant. A la fin du traitement, Wow64 convertit les paramètres de sortie avant de repasser dans le mode 32 bits.

Impression

Considérant l'impossibilité pour Wow64 de charger des pilotes mode noyau 32 bits (voir à ce sujet la section Restrictions), les pilotes d'imprimante 32 bits sont inutilisables sous Windows 64 bits. Un mécanisme d'impression spécifique aux processus 32 bits est cependant mis en oeuvre via le processus splwow64.exe, correspondant au serveur d'impression RPC de Wow64, que le système utilise chaque fois qu'une demande d'impression émane d'un processus 32 bits. Comme Splwow64 est un processus 64 bits, il peut charger des pilotes d'imprimante 64 bits. Dans Windows Vista et versions ultérieures de Windows, les pilotes d'imprimante ont obligation de suivre l'infrastructure de pilote en mode utilisateur (UMDF, User-Mode Driver Framework).

Performances de Wow64

Performances et consommation de mémoire des processus Wow64 sont déterminés par un ensemble de facteurs incluant la plate-forme matérielle sous-jacente, les frais imputables à la conversion des appels 32 bits en appels 64 bits natifs (*thunking*) et, finalement, au la dimension de l'ensemble de travail (*working set*).

- *Matériel processeur* Une large part des opérations effectuées par Wow64 viennent en réalité directement de la main de la plateforme matérielle x64, dont un mode d'exécution distinct, appelé mode de compatibilité (*compatibility mode*), permet l'exécution d'instructions x86 32 bits. De cette manière, la vitesse à laquelle lesdites instructions sont traitées sous Wow64 est similaire à celle observée sous les systèmes x86 natif.

- *Mémoire* Chaque fois qu'un processus Wow64 accède à une ressource gérée par le noyau, le chemin emprumté à cette occassion aboutit invariablement à un appel système. Les systèmes Windows 64 bits traitent des données perceptiblement différentes en taille de celles manipulées par les applications 32 bits, et qu'un protocole de conversion s'avère par conséquent nécessaire. Un tel dispositif a un surcout, pas tellement en ce qui concerne les performances - les frais au niveau du processeur sont négligeables dans la plupart des cas - mais relativement à la quantité de mémoire nécessaire pour la représentation des données. Amener des donnée 32 bits a un format 64 bits double essentiellement la quantité de mémoire nécessaire pour les stocker. Ce fait, particulièrement impactant, touche divers domaines, des stratégies d'allocation à la création de threads.

- *Working set* Considérant que les systèmes informatiques 64 bits disposent de de plus grandes quantités de mémoire physique que leurs homologues 32 bits, Wow64 dimensionne les working sets de processus de façon plus souple que ne le ferait Windows 32 bits. Avec plus de pages virtuelles résidant en mémoire physique, les threads de processus Wow64 ont plus de chance de référéncer des données sans provoquer de défaut de page. Notez que Wow64 étant majoritairement implémenté en mode utilisateur, le mécanisme n'a aucune incidence sur le code et les données paginables du système d'exploitation.

Wow64 permet aux applications 32 bits de bénéficier, ne serait-ce que partiellement, de l'architecture 64 bits sous-jacente. Elles peuvent par exemple utiliser un plus grand nombre de descripteurs faisant référence à des objets noyau ou à des fenêtres. Cependant, les procédures mises en œuvre pour assurer un bon fonctionnement aux applications 32 bits (Wow64 donc) sont elles-aussi consommatrices de ressources et, en tant que telles, tendent à peser sur les entités dont elles assurent la gestion. Ainsi, les applications 32 bits ont proportionnellement moins de possibilité de créer des threads dans Wow64 que dans leur écosystème natif, cela en raison de la création par le sous-système d'une pile supplémentaire 64 bits pour chaque thread. En outre, le système réserve automatiquement pour Wow64 et les structures de données qu'il utilise une certaine quantité de mémoire, ponctionnée pour l'occasion dans l'espace d'adressage mode utilisateur du processus.

Communication entre processus Wow64 et processus natifs

Comme tout programme informatique, les entités sous l'aile de Wow64 ont besoin quelquefois de partager des données ou des ressources avec d'autres composants du système d'exploitation. Les processus Wow64 disposent en la matière et à quelques exceptions près des mêmes mécanismes que ceux en œuvre pour la communication inter processus des processus exécutés sur le système 64 bits natifs.

- *Objets et handles d'objets* Par égard pour l'interopérabilité avec le parc applicatif existant, les éditions 64 bits de Windows emploient en divers endroits du système des mécanismes orientés non en fonction des propriétés du système natif, mais de celles de l'architecture héritée 32 bits. C'est, par exemple en ce qui concerne les méthodes d'accès aux objets, le cas des handles, qui conservent (en surface) une dimension unique entre les versions 32 bits et 64 bits de Windows.

- *RPC* Conçus pour atténuer les problèmes structurels auxquels sont confrontées les applications, les appels de procédure distante, agnostiques vis-à-vis de l'architecture sur laquelle ils fonctionnent, répondent de ce fait particulièrement bien aux problèmes des échanges et de la synchronisation entre processus Wow64 et processus natifs.

- *COM* Le système d'exploitation fournit de l'interopérabilité à travers la frontière 32/64 bits pour COM ainsi que pour diverses opérations élémentaires relatives au presse-papiers, telles que couper, copier et coller.

- *Mémoire partagée* La mémoire peut être partagée entre processus 32 bits et processus 64 bits à la condition que les types de données dépendants des pointeurs soient convertis.

- *Création de processus* Les interfaces Windows avec lesquelles créer des processus, dont font par exemple partie les API CreateProcess et ShellExecute, sont capables d'ouvrir des chemins d'exécution 32 bits ou 64 bits, selon les caractéristiques du fichier image (.exe) à exécuter. Un processus Wow64 peut ainsi donner naissance à un processus dans l'environnement natif du système d'exploitation, et un processus 64 bits demander la création d'un processus 32 bits.

Pilotes en mode noyau

Wow64, sans doute dans le cadre d'une politique visant à plus de modernité dans les systèmes Microsoft actuels, ne permet pas le chargement de pilotes en mode noyau 32 bits. Le noyau 64 bits ne les prenant pas en charge, de tels logiciels doivent par conséquent être portés vers le 64 bits natif.

Les pilotes peuvent appeler la routine IoIs32bitProcess pour déterminer si une requête d'E/S émane ou non d'un processus Wow64. Dans le mode utilisateur, deux fonctions intégrées à Kernel32.dll peuvent détecter les processus Wow64, IsWow64Process et GetNativeSystemInfo. IsWow64Process retourne un indicateur permettant de déterminer si un processus s'exécute sous WOW64. GetNativeSystemInfo extrait des informations sur le système pour une application fonctionnant sous WOW64. Si la fonction est appelée à partir d'une application 64 bits, elle est équivalente à GetSystemInfo.

Limitations et restrictions inhérentes à Wow64

Ce qui suit répertorie quelques-unes des limitations inhérentes à Wow64 et, partant, à la compatibilité des programmes 32 bits dans l'environnement Windows 64 bits.

- *Espace d'adressage* La taille par défaut de l'espace d'adressage pour un processus Wow64 est limitée à 2 Go, extensible à 4 si l'application gère les adresses de taille supérieure (option /LARGEADDRESSAWARE de l'Éditeur de liens).

- *16 bits* Les versions 64 bits de Windows, donc par association le sous-système Wow64, ne supportent aucune application 16 bits. De ce fait, si vous possédez une application héritée 16 bits (ou une application 32 bits avec un programme d'installation 16 bits), vous devrez soit conserver l'infrastructure existante en assumant la charge, soit vous tourner vers les diverses solutions de virtualisation présentes sur le marché.

- *Posix et OS/2* Wow64 n'intègre pas de support à destination des applications Posix ou OS/2, avec la conséquence que ces programmes ne fonctionnent pas nativement sur les systèmes Microsoft 64 bits. (Des alternatives sont néanmoins disponibles.)

- *32 bits en Mode noyau* Wow64 ne permet pas de charger des pilotes des périphériques mode noyau 32 bits. Cela a pour conséquence que les pilotes de niveau système doivent être compilés en mode 64 bits, portées, voire repensées, afin de tirer le meilleur avantage du 64 bits natif.

- *Mixité entre programmes 32 et 64 bits* Les processus Wow64 ne peuvent charger que des DLL 32 bits, mais pas de DLL 64 bits natives. De même, les processus 64 bits natifs ne peuvent pas charger de DLL 32 bits. Divers cas particuliers illustrant cette limite sont, par exemple, Microsoft Internet Explorer ne pouvant charger de contrôles ActiveX 32 bits, l'interpréteur de commandes standard ne pouvant interagir avec les extensions d'environnement 32 bits, ou les programmes d'installation 32 bits ne pouvant enregistrer des DLL en 64 bits.

Redirection du système de fichiers

Pour diverses raisons mal définies, un certain nombre de programmes 32 bits font référence aux répertoires système non via la méthode appropriée, à laquelle correspond la fonction Windows GetSystemDirectory, mais sur l'assertion que le chemin d'accès \Windows\System32 fasse référence inconditionnellement au dossier racine du système d'exploitation. Autrement dit, même une fois exécuté dans un contexte 64 bits, ces programmes essaieront toujours d'accéder au répertoire \Windows\System32. Pour donner une solution efficace à cette problématique, les noms des répertoires système communs entre Win32 et Windows 64 bits ont été conservés, et un mécanisme de redirection intégré pour assurer le bon fonctionnement de l'ensemble.

Remarque A l'époque de Windows 16 bits, le répertoire hôte de la plupart des fichiers nécessaires au système était \Windows\System. Avec l'introduction de Win32, un autre répertoire fut ajouté, soit \Windows\System32, contenant les fichiers et modules natifs 32 bits. Windows 95 jeta le trouble et commença à semer la confusion en faisant cohabiter dans System32 les deux types de fichiers système, 16 bits et 32 bits.

Dans les systèmes Windows 64 bits, le répertoire \Windows\System32 contient les images 64 bits natives, réservées aux applications 64 bits. (Notez que le nombre 32, partie du nom, perd pour l'occasion toute signification particulière.) Wow64, quand il intercepte les appels système, traduit les API liées aux chemins, de sorte que lorsqu'un processus 32 bits accède au répertoire \Windows\System32, ses opérations sont redirigées vers le répertoire \Windows\SysWOW64, lequel héberge les versions 32 bits des modules habituels. De plus, la plupart des programmes 32 bits sont détectés a l'installation et enregistrées dans le \Program Files (x86), alors que les programmes 64 bits vont dans le dossier \Program Files.

Les modalités dont s'effectuent l'accès à quelques sous-répertoires de \Windows\System32 sont, pour des raisons de compatibilité, exemptées de la redirection du système de fichiers, les opérations des applications 32 bits qui y accèdent ciblant du coup de vrais répertoires. Ceux-ci incluent :

- %windir%\system32\catroot
- %windir%\system32\catroot2
- %windir%\system32\driverstore
- %windir%\system32\drivers\etc
- %windir%\system32\logfiles
- %windir%\system32\spool

Par défaut, la redirection de système de fichiers intégrée à Wow64 est active pour tout processus exécutée conjointement à ce sous-système. Les processus peuvent pour interagir avec le mécanisme avoir recours aux API Wow64DisableWow64FsRedirection, Wow64EnableWow64FsRedirection et Wow64RevertWow64FsRedirection. Neutraliser la redirection du système de fichiers affecte l'ensemble des E/S de niveau fichier qui émanent du thread appelant. Une telle solution ne devrait par conséquent être envisagée que dans le cadre le cadre d'occasions spécifiques, et dont il résulte des temps d'exécution réduits. Bloquer les E/S orientées fichier pendant des périodes plus longues peut empêcher les application 32 bits de charger des DLL système, entraînant de ce fait leur échec.

Les applications 32 bits souhaitant accéder au répertoire système natif ont la possibilité de le faire à l'aide du chemin d'accès %windir%\Sysnative, que Wow64 reconnaît comme un alias spécial utilisé pour indiquer que le

système de fichiers ne doit pas rediriger l'accès. Notez que les applications 64 bits ne peuvent pas utiliser l'alias Sysnative, dans la mesure où il s'agit d'un répertoire virtuel, qui n'existe donc pas physiquement en tant que tel sur le système. Notez que si vous naviguez avec l'explorateur de fichiers à l'emplacement susmentionné, vous ne trouverez nulle part mention d'un répertoire Sysnative, y compris si l'option Afficher les dossiers cachés et système est exercée. Cela est simplement dû au fait que le dossier Sysnative est visible et accessible uniquement à partir de programmes 32 bits.

Les diverses fonctions Windows apparentées à la redirection de système de fichiers affectent toutes les E/S orientées fichier émises par le thread appelant. Il en résulte quelques situations conflictuelles du moment où la redirection est désactivée. Les procédures de chargement de DLL, par exemple, émettent des E/S dépendantes de chemins d'accès redirigées, et auront dans ce contexte des résultats inattendus. Ces procédures ne répondent en outre pas toujours à un chemin d'exécution évident : une DLL peut avoir des dépendances envers d'autres DLL, une application charger une DLL au nom d'une autre application, ou encore Windows différer le chargement d'une DLL jusque-là survenue d'un événement. Dans n'importe laquelle de ces situations, l'application pourrait dépendre d'un chemin d'accès habituellement redirigé par Wow64, qui ne l'est pas quand le système concrétise les traitements nécessaires à la requête d'E/S. Pour éviter ces problèmes, il est vivement conseillé de désactiver la redirection immédiatement avant une fonction d'E/S, et de la réactiver immédiatement après.

Les opérations concernant la redirection de système de fichiers s'effectuent à l'échelle du thread. Certaines fonctions, tel CreateProcessAsUser, qui exécutent diverses tâches pour le compte (et dans le contexte) d'autres threads ne sont donc pas concernées par l'état de la redirection dans le thread appelant.

Variables d'environnement

Corollaire aux quelques divergences que le sous-système instaure en ce qui concerne le registre et des parties du système de fichiers, WOW64 expose aux applications 32 bits des variables d'environnement différentes de celles vues par les applications 64 bits. Les processus 32 bits voient par exemple la variable d'environnement relative au chemin d'installation des programmes (%ProgramFiles%) en tant que \Program Files (x86), cela en référence au répertoire des programmes x86 enregistrés dans Windows 64 bits, alors que les programmes 64 bits voient la valeur \Program Files, soit le chemin normal des programmes installés.

Quand un processus 32 bits est créé par le système 64 bits natif, ou quand un processus 64 bits est créé par un processus 32 bits, Wow64 participe à la condition (l'état) de quelques-unes des variables dynamiques utilisées par ce processus. Le tableau montre ces variables, et présente les fluctuations se manifestant à ce niveau entre processus 32 bits natifs, processus 32 bits sur plateforme matérielle 64 bits - emmenées par Wow64 donc, et les processus 64 bits.

Tableau 4.1. Variables d'environnement pour processus 32 bits, processus 64 bits et processus Wow64

Variable	32 bits natif	64 bits natif	Wow64
PROCESSOR_ARCHITECTURE	x86	amd64	x86
PROCESSOR_ARCHITEW6432	Non défini	Non défini	amd64
ProgramFiles	\Program Files	\Program Files	\Program Files (x86)
ProgramW6432	-	\Program Files	\Program Files (x86)
CommonProgramFiles	\Program Files\Common Files	\Program Files\Common Files	\Program Files\Common Files(x86)
CommonProgramW6432	-	\Program Files\Common Files	-

Flags globaux de Windows

Pour faciliter le débogage et préparer le terrain à toutes sortes d'autres activités de suivi, Windows prend en charge un jeu de modificateurs (flags) faisant écho à diverses fonctionnalités de diagnostic, de trace et de validation internes. Appelés les *flags globaux de Windows*, ces paramètres servent à altérer la façon dont les logiciels interagissent avec les composants du système d'exploitation, à la fois en mode utilisateur et en mode

noyau, et font preuve de leur utilité en de multiples circonstances, allant de l'enquête sur les signes d'une corruption du tas à la préparation de tests de résistance.

Chacun des flags globaux de Windows se caractérise par un nom qui le désigne en surface et par une valeur numérique que le système d'exploitation emploie en interne. S'ajoute à cela une abréviation alphabétique de trois caractères qui, si le schéma dont elle fait partie n'a pas d'existence en dehors du plan fonctionnel (et absolument aucune sur le plan programmatique), n'en reste pas moins suivi par la plupart (sinon tous) les utilitaires permettant des interactions sur ces propriétés. Par exemple, le sous-ensemble des flags globaux correspondant à la fonctionnalité de balisage de pool (*Enable pool tagging*) renvoie au nom FLG_POOL_ENABLE_TAGGING, à la valeur 0x0400 et l'abréviation ptg.

Constante	Valeur	Abréviation	Signification
FLG_STOP_ON_EXCEPTION	0x00000001	soe	Stop on exception
FLG_SHOW_LDR_SNAPS	0x00000002	sls	Show loader snaps
FLG_DEBUG_INITIAL_-COMMAND	0x00000004	dic	Debug initial command
FLG_STOP_ON_HUNG_GUI	0x00000008	shg	Stop on hung GUI
FLG_HEAP_ENABLE_TAIL_-CHECK	0x00000010	htc	Enable heap tail checking
FLG_HEAP_ENABLE_FREE_-CHECK	0x00000020	hfs	Enable heap free checking
FLG_HEAP_VALIDATE_-PARAMETERS	0x00000040	hpc	Enable heap parameter checking
FLG_HEAP_VALIDATE_ALL	0x00000080	hvc	Enable heap validation on call
FLG_APPLICATION_VERIFIER	0x00000100	vrf	Enable application verifier
FLG_POOL_ENABLE_TAGGING	0x00000400	ptg	Enable pool tagging
FLG_HEAP_ENABLE_TAGGING	0x00000800	htg	Enable heap tagging
FLG_USER_STACK_TRACE_DB	0x00001000	ust	Create user mode stack trace database
FLG_KERNEL_STACK_TRACE_-DB	0x00002000	kst	kernel mode stack trace database
FLG_MAINTAIN_OBJECT_-TYPELIST	0x00004000	otl	Maintain a list of objects for each type
FLG_HEAP_ENABLE_TAG_BY_-DLL	0x00008000	htd	Enable heap tagging by DLL
FLG_DISABLE_STACK_-EXTENSION	0x00010000	dse	Disable stack extension
FLG_ENABLE_CSRDEBUG	0x00020000	d32	Enable debugging of Win32 subsystem
FLG_ENABLE_KDEBUG_-SYMBOL_LOAD	0x00040000	ksl	Enable loading of kernel debugger symbols
FLG_DISABLE_PAGE_KERNEL_-STACKS	0x00080000	dps	Disable paging of kernel stacks
FLG_ENABLE_CLOSE_-EXCEPTIONS	0x00400000	ece	Enable close exception
FLG_DEBUG_INITIAL_-COMMAND_EX	0x04000000	dwl	Debug WinLogon
FLG_DISABLE_PROTDLLS	0x80000000	dpd	Disable protected DLL verification

Les flags globaux du système sont stockés sous la forme d'un masque binaire dans une variable globale, nommée *NtGlobalFlag*, initialisée lors de l'amorçage de Windows depuis la clé de registre HKLM\SYSTEM \CurrentControlSet\Control\Session Manager dans la valeur GlobalFlag. Par défaut, cette valeur de registre est 0x400, à quoi correspond en interne l'activation du balisage de pool.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager
```

```
GlobalFlag    REG_DWORD    *0x400*
```

La représentation binaire (et à l'échelle d'une seule clé de registre) étant quelquefois insuffisante pour englober toutes les informations relatives à un statut, une condition ou à un chemin d'exécution alternatif, certains composants de Windows ont en partie pour tâche de gérer des protocoles additionnels en vue de leur propre configuration. C'est, par exemple, la situation du gestionnaire de tas qui, une fois ses options activées par l'intermédiaire de flags globaux, emploie une valeur de registre spécifique (*PageHeapFlags*) de sorte à pouvoir gérer ses fonctionnalités de façon plus granulaire.

Logiciel *Global Flags*

La manière dont sont stockés les flags Windows globaux (en l'occurrence un masque binaire) ne se prête que très malaisément à une perception intuitive. Heureusement, Windows SDK et les outils de débogage standard incluent un utilitaire graphique nommé Global Flags via lequel interagir sur ces aspects, et par conséquent voir et modifier les flags globaux du système (soit dans le registre, soit dans le système en cours d'exécution), ainsi que les flags globaux d'une image.

Le module exécutable sous-jacent à Global Flag, gflags.exe, peut être lancé à partir du chemin d'installation par défaut ou depuis un des raccourcis créés par le programme d'installation. L'utilitaire demande automatiquement l'élévation de ses privilèges au niveau administratif. Ainsi que vous pouvez le voir au niveau de la figure \<<gflags-system-registry>\>, l'interface principale de Global Flags est divisée en onglets regroupant plusieurs options d'une même catégorie.

Boîte de dialogue Global Flags . image::gflags-system-registry.png[effe, 200, scaledwidth="30%"]()

Vous pouvez basculer entre paramètres du registre (en cliquant sur System Registry) et valeur courante de la variable en mémoire système (en cliquant sur Kernel Flags). S'il est possible dans quelques cas de remanier la disposition générale des flags d'un système en cours d'exécution, la plupart des scénarios exigent un réamorçage système afin que les modifications prennent effet.

Les possibilités de configuration mises en avant dans l'onglet System Registry (le plus à gauche) s'appuient sur l'existence de paramètres dans le registre Windows. Les modifications ainsi faites sont répercutées sous la clé correspondante appartenant au gestionnaire de session, et alimentent la variable NtGlobalFlags lors du prochain démarrage du système d'exploitation.

Les options rendues visibles depuis l'onglet Kernel Flags peuvent être utilisées pour ajuster en temps réel la valeur courante de la variable globale système NtGlobalFlag. Tout changement apporté par ce biais est effectif sans avoir besoin de redémarrer Windows, au détriment de sa survie entre deux amorçages du système. Notez que quelques-uns des comportements appuyés par des flags globaux ne peuvent prendre effet à la volée, dans la mesure où les paramètres associés sont lus uniquement durant le démarrage du système d'exploitation - il n'existe pas de prise en compte dynamique de la conduite à tenir. Ceci est la raison pour laquelle quelques options de la liste des paramètres du registre sont absentes de la gamme des choix accessibles en ce qui concerne le noyau.

Les contrôles regroupés sous l'onglet Image File permettent de modifier les flags globaux relatifs à une image exécutable, plutôt que ceux du système dans son ensemble.

Configuration des flags globaux d'image avec Gflags. image::gflags-image-file.png[effe, 200, scaledwidth="30%"]()

Les informations dans le champ de saisie Image doivent faire référence à un nom de module, et non à un emplacement du système de fichiers. Cela signifie qu'il vous suffit d'entrer dans cette zone de texte le nom de fichier d'une image exécutable, par exemple notepad.exe. Remarquez que vous devez inclure l'extension de fichier dans le nom du fichier. Les flags globaux d'une image se propageant au processus associé seulement à l'occasion de sa mise en route, les modifications apportées à l'aide de l'outil Gflags (ou même les changements effectués directement dans le Registre) s'appliquent uniquement aux instances nouvellement créées.

Une fois que vous avez défini par l'intermédiaire de l'outil Gflags les flags globaux d'une image exécutable, vous pouvez vérifier que le Registre comporte bel et bien une nouvelle entrée concernant le module dont vous avez renseigné le nom dans la boite de dialogue Image (notepad.exe dans notre exemple). Pour ce faire, à l'aide de n'importe quel outil d'édition du registre, portez votre attention sur les clés et valeurs hébergées sous HKLM \SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options. Vous devriez voir quelque chose ressemblant à ce qui suit.

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\notepad.exe
    GlobalFlag    REG_DWORD    0x2000000
    PageHeapFlags    REG_DWORD    0x3
```

Idéalement, vous ne devriez interférer sur le cours des évènements induit par les flags globaux de Windows que par l'entremise d'outils tiers spécialisés à cette fin. Au cas où vous souhaiteriez passer outre cet avertissement et procéder à des changements directs dans le registre, veillez à garder à l'esprit que dans les versions 64 bits de Windows, programmes natifs et programmes 32 bits ne disposent pas de la même vue de certaines parties du système d'exploitation. Cela signifie que si un module renvoie à du code x86 32 bits, vous devrez mettre à jour la clé de registre située sous l'arborescence HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT \CurrentVersion\Image File Execution Options\. L'utilitaire Gflags est en la situation assez pratique pour modifier à la fois la vue native (portion 64 bits) et la vue Wow64 (portion 32 bits) du Registre, de sorte que vous n'ayez pas à vous préoccuper de ces détails.

Flags globaux d'image

De façon similaire à comment le comportement du système peut être dirigé par l'intermédiaire de modificateurs (flags), chaque image exécutable dispose également d'un ensemble de flags globaux chargés d'activer du code interne de trace, de vérification et de recouvrement d'erreurs. Au contraire des flags globaux du système, qui étendent leur portée sur tout l'environnement, les flans globaux relatifs à un fichier image n'ont d'influence que sur les instances d'un processus.

La configuration d'ensemble des flags globaux d'image trouve un abri dans le Registre sous HKLM\SOFTWARE \Microsoft\Windows NT\CurrentVersion\Image File Execution Options\. Lors de la création de processus, le système s'en remet à cet emplacement de sorte à voir s'il existe une sous-clé nommée à partir du nom de fichier et de l'extension de l'image exécutable. Si tel est le cas, il recherche une valeur nommée GlobalFlag pour cette clé. S'il y en a une, les données sont alors enregistrées en tant que flags globaux du processus. Autrement, à savoir dans l'éventualité où la valeur GlobalFlag est absente ou qu'il n'est nulle part fait mention de l'image sous Image File Execution Options (ce qui sur ce sujet revient au même), les flags globaux du processus sont dérivés de la valeur de la variable en mémoire système nommée NtGlobalFlag (bien que la disposition binaire de ces flags soit complètement différente de celles des flags système globaux).

Une fois que les flags globaux d'une image ont été déterminés, le chargeur répercute ces informations à l'attribut NtGlobalFlag du bloc PEB dans l'espace d'adressage mode utilisateur du processus exécutant l'image.

```
0:000> dt ntdll!_PEB NtGlobalFlag @$Peb
   +0x0bc NtGlobalFlag : 0x2000400
```

Lors de vos expérimentations sur les flags globaux d'une image, sachez que quelques-uns sont automatiquement mis en oeuvre dès lors que le processus associé est démarré depuis un débogeur en mode utilisateur. Quand une requête de création de processus survient, le code d'initialisation afférent dans le module ntdll, par nature instruit des conditions dans lesquelles tout fichier image a été exécuté, observe si ces dernières sont de nature à assurer le diagnostic des applications, et si elles le sont effectivement, anime de façon silencieuse un petit nombre de modificateurs. Le cas concerne uniquement les modules pour lesquelles aucun paramètre n'est explicitement défini à cet égard dans le Registre. A titre d'illustration, la sortie suivante montre la valeur courante des flags globaux du processus calc.exe lorsque l'image concernée n'a pas d'attributs déterminés en la matière.

```
0:000> !gflag
Current NtGlobalFlag contents: 0x00000070
    htc - Enable heap tail checking
    hfc - Enable heap free checking
```

```
    hpc - Enable heap parameter checking
```

Ainsi que vous pouvez le voir, un petit nombre paramètres qui ont trait au suivi du tas sont au niveau de l'image notepad.exe à l'état actif, sans par ailleurs que rien de concret nous ai préparé à cette situation. Un fait intéressant à relever ici et qui découle directement de cette caractéristique est qu'une application peut se comporter différemment selon son exécution au sein d'un environnement de débogage. Dans notre exemple, ceci explique pourquoi un programme fautif vis-à-vis du tas en manifeste tous les signes quand il est pris en charge par un débogueur, et jamais ou dans de rares occasions quand il ne l'est pas.

Vous pouvez si vous le souhaitez couper court à l'influence des procédés de diagnostic sur la gestion du tas, a fortiori sur les flags globaux de niveau image, en définissant la variable d'environnement _NO_DEBUG_HEAP avant de donner naissance à un processus, ou en indiquant l'option de ligne de commande dh lors du démarrage de WinDbg. Ces deux opérations ont pour effet de demander de disposer du tas standard plutôt que de celui axé sur le débogage. Une autre alternative consiste à attacher le débogueur à un processus en cours d'exécution, dont les conditions initiales ne le préparaient par conséquent pas de façon spécifique à l'analyse.

À titre informatif, notez que la manière dont influent les pratiques de débogage sur les flags globaux d'image constitue un moyen parmi d'autres de détecter la présence d'environnement virtuel ou d'outils utilisés dans le cadre de l'analyse dynamique. Une application qui souhaite freiner l'étude de son code exécutable ou de certaines de ses informations peut par exemple observer le positionnement des flags globaux qui lui ont été attribués, et adopter un comportement particulier en conséquence.

Commande d'extension *!gflag*

L'extension !gflag, accessible depuis les outils de débogage standard, permet d'interagir avec les flags globaux de Windows. Sollicitée dans un contexte mode utilisateur, la commande permet de voir l'état de la variable NtGlobalFlag conservée au niveau du bloc d'environnement du processus sous le contrôle du débogueur.

```
0:001> !gflag
*Current NtGlobalFlag contents: 0x00000002*
    sls - Show Loader Snaps
```

Vous pouvez si vous le souhaitez obtenir une assistance en ce qui concerne cette extension en spécifiant le commutateur -?. L'aide textuelle qui apparait alors contient une liste détaillée d'abréviations dont l'emploi ambitionne de simplifier la manipulation des flags globaux.

```
0:003> !gflag -?
usage: !gflag [-? | flags]
Flags may either be a single hex number that specifies all
32-bits of the GlobalFlags value, or it can be one or more
arguments, each beginning with a + or -, where the + means
to set the corresponding bit(s) in the GlobalFlags and a -
means to clear the corresponding bit(s).  After the + or -
may be either a hex number or a three letter abbreviation
for a GlobalFlag.  Valid abbreviations are:
    soe - Stop On Exception
    sls - Show Loader Snaps
    htc - Enable heap tail checking
    hfc - Enable heap free checking
    hpc - Enable heap parameter checking
    hvc - Enable heap validation on call
    vrf - Enable application verifier
    htg - Enable heap tagging
    ust - Create user mode stack trace database
    htd - Enable heap tagging by DLL
    dse - Disable stack extensions
    scb - Enable system critical breaks
    dhc - Disable Heap Coalesce on Free
    eel - Enable exception logging
    hpa - Place heap allocations at ends of pages
```

```
    cse - Early critical section event creation
    sue - Stop on Unhandled Exception
    dpd - Disable protected DLL verification
```

La commande !gflag du débogueur noyau permet de voir la valeur de la variable NtGlobalFlag en mémoire système.

```
lkd> dd nt!NtGlobalFlag
fffff800`2f5661c0  *00000400* 00000000 00000000 00000000

lkd> !gflag
Current NtGlobalFlag contents: *0x00000400*
    ptg - Enable pool tagging

lkd> !gflag -?
    soe - Stop On Exception
    sls - Show Loader Snaps
    dic - Debug Initial Command
    shg - Stop on Hung GUI
    htc - Enable heap tail checking
    hfc - Enable heap free checking
    hpc - Enable heap parameter checking
    hvc - Enable heap validation on call
    vrf - Enable application verifier
    *ptg - Enable pool tagging*
    htg - Enable heap tagging
    ust - Create user mode stack trace database
    kst - Create kernel mode stack trace database
    otl - Maintain a list of objects for each type
    htd - Enable heap tagging by DLL
    dse - Disable stack extensions
    d32 - Enable debugging of Win32 Subsystem
    ksl - Enable loading of kernel debugger symbols
    dps - Disable paging of kernel stacks
    scb - Enable system critical breaks
    dhc - Disable Heap Coalesce on Free
    ece - Enable close exception
    eel - Enable exception logging
    eot - Enable object handle type tagging
    hpa - Place heap allocations at ends of pages
    dwl - Debug WINLOGON
    ddp - Disable kernel mode DbgPrint output
    cse - Early critical section event creation
    sue - Stop on Unhandled Exception
    bhd - Enable bad handles detection
    dpd - Disable protected DLL verification
```

La commande !gflag peut également être utilisée afin de prendre la main sur l'état des flags globaux. Dans la pratique, cela peut s'effectuer de façon naturelle, en indiquant en paramètre une nouvelle valeur de la variable, soit sous la forme d'opérations basiques. Les signes plus (+) et moins (-) servent auquel cas à armer et désarmer tel ou tel drapeau. En guise d'exemple, ce qui suit montre comment activer et désactiver les informations de trace du chargeur d'image.

```
lkd> !gflag
Current NtGlobalFlag contents: 0x00000400
    ptg - Enable pool tagging

lkd> !gflag +soe
New NtGlobalFlag contents: *0x00000401*
    soe - Stop On Exception
    ptg - Enable pool tagging

lkd> !gflag -1
New NtGlobalFlag contents: *0x00000400*
```

```
    ptg - Enable pool tagging
```

Pour finir, notez que si l'extension !gflag permet une approche définitivement pratique des flags Windows globaux, vous restez bien évidement libre de correspondre avec eux sans son intermédiaire, moyennant un confort bien moindre. Utilisiez dans ce scénario les options classiques en matière d'interaction avec la mémoire, comme nous le faisons (en partie) dans l'exemple qui suit.

```
lkd> ed nt!NtGlobalFlag 0x401

lkd> !gflag
Current NtGlobalFlag contents: 0x00000401
    soe - Stop On Exception
    ptg - Enable pool tagging
```

Options d'exécution d'image

Les options d'exécution de fichier image (IFEO, *Image File Execution Options*) sont une collection de paramètres utilisés en amont de la création des processus de sorte à en assurer un contrôle spécifique. Étroitement liées au mode opératoire et aux agissements du chargeur (*loader*), un composant qui est par ailleurs doté lui-même de multiples ramifications dans de nombreux champs, ces données permettent de s'introduire à plusieurs étapes de la naissance d'un processus, mais aussi et surtout à divers niveaux : scénarios pour l'ouverture du fichier image à exécuter, niveaux d'analyse des fonctionnalités de débogage, détails des informations de trace, et bien d'autres.

Le registre Windows offre un toit aux options d'exécution par le biais de HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options, lequel contient pour chaque image exécutable à prendre en compte dans un scénario particulier (toutes ne le sont pas) une sous-clé nommée depuis le nom de fichier et l'extension de l'image (mais sans les données de répertoire et de chemin, par exemple Image.exe). Les valeurs sous ces clés peuvent constituer un référentiel d'informations potentiellement vaste. Pour voir quelles applications de votre système ont des options d'exécution, tournez-vous vers la clé Image File Execution Options au moyen de n'importe quel outil permettant de le faire, Regedit ou autre.

Les options d'exécution permettent une interaction très poussée tant avec les processus qu'avec les composants Windows chargés de les gérer. Elles sont de ce fait potentiellement impliquées dans un plus ou moins grand nombre de cas de figure, avec plus ou moins de relations entre eux. La liste qui suit énumère quelques-uns des leviers manœuvrés par des options d'exécution. (Pour des raisons de clarté, et afin de vous donner dès ici les bonnes pistes à suivre dans le registre, nous signalons entre parenthèses les noms des clés adéquates.)

- *Contrôle externe* (*Debugger*) Permet à un processus de démarrer automatiquement sous la coupe d'un autre logiciel, généralement un débogueur.
- *Gestion du tas* (*DisableHeapLookAside*) Joue un rôle dans la décision de délaisser l'emploi de listes look-acide parmi les mécanismes d'allocation du tas. Cela est notamment utile pour résoudre d'éventuels problèmes de compatibilité.
- *Débogage de tas* (*PageHeapFlag*s) Indique sur le plan des fonctionnalités de débogage de tas lesquelles lesquelles sont actives, et à quel degré.
- *Flags globaux* (*GlobalFlag*) Anime un ensemble de techniques en association avec le diagnostic et le suivi, y compris des données de trace et des fonctions de validation internes.
- *Informations de taille de pile* (*MinimumStackCommitInBytes*) Apporte des garanties en ce qui concerne le dimensionnement et l'accessibilité de l'espace de pile.
- *Application Verifier* (*VerifierFlags*) Base de données des contrôles et des vérifications auxquels se livre le logiciel Microsoft Application Verifier.

Il apparaît clairement à la teneur des mécanismes dans lesquels elles s'insèrent que les options d'exécution d'image ont une dimension pour l'essentiel orientée sur le débogage, et dans une moindre mesure sur le profilage,

des applications. Elles mettent en l'occurrence aux mains des concepteurs de logiciels diverses sources de données de trace, ainsi que les fonctions, les caractéristiques, voire les utilitaires, qui s'y réfèrent.

Performances

Compteurs de performance

Les compteurs de performance Windows permettent de mesurer l'évolution de certaines données de performance liées à une machine (taux d'occupation du processeur, mémoire utilisée, E/S) ou à un ou plusieurs processus (taille des piles, quantité de mémoire utilisée, etc). Windows offre nativement un large éventail de compteurs de performance, qu'administrateurs et utilisateurs peuvent étoffer en définissant les leurs propres.

Les compteurs de performance sont affichables sous forme graphique à l'aide de l'Analyseur de performance, ou sous forme textuelle à l'aide de l'utilitaire Logman.

L'architecture sous-jacente aux compteurs de performance se présente dans les grandes lignes comme suit :

- *Objets* Les objets de performance sont issus de la répartition logique des ressources système en plusieurs catégories distinctes. Il existe par exemple un objet de performance pour le processeur, un autre pour la mémoire, pour les services, et ainsi de suite.

- *Compteurs* Les compteurs forment la base de l'observation de l'usage des ressources, et constituent ainsi les paramètres réels mesurés.

- *Instances* Les instances d'objets de performance sont des ensembles d'un même type d'objet. Par extension, une instance identifie le paramètre de performance que le compteur mesure. Certains types d'objets comportent plusieurs instances. Par exemple, sur un système multi processeur, le type d'objet Processeur dispose de plusieurs instances. Si un système contient deux disques, le type d'objet Disque physique possède deux instances. Certains objets, tels que ceux relatifs à la mémoire, ne disposent que d'une seule instance.

Analyseur de performances (Perfmon)

Analyseur de performances permet de créer des ensembles interactifs de compteurs système ou des ensembles de collecteurs de données. (Lesdits ensembles sont étudiés plus en détail dans la section "Ensembles de collecteurs de données", plus loin dans ce chapitre.

Comme socle technique, l'utilitaire Performance s'appuie sur les fonctions de l'API Performance Data Helper afin d'interagir avec les diverses sources de données de performances, dont les compteurs de performance en temps réel ou les fichiers journaux spécifiques à cet égard.

L'Analyseur de performances se présente sous la forme d'un composant logiciel enfichable de la console MMC, mis en oeuvre dans perfmon.msc.

Utilisez l'une ou l'autre des méthodes suivantes afin d'accéder à l'utilitaire Performances.

- Ouvrez la console Gestion de l'ordinateur (compmgmt.msc), développez Outils système, puis cliquez sur Performance.

- Depuis Outils d'administration, sélectionnez le raccourci nommé Analyseur de performances.

- Saisissez perfmon.msc ou perfmon.msc et appuyez sur Entrée dans le menu Démarrer, la boite de dialogue Exécuter ou encore dans une fenêtre d'invite de commandes.

Ajouter des compteurs dans l'Analyseur de performances. L'Analyseur de performances affiche uniquement les informations relatives aux compteurs suivis. Plusieurs milliers de compteurs sont disponibles,

réparties en un petit nombre de catégories tels que Processus, Threads, Mémoire, Cache, et ainsi de suite. Les compteurs les plus utiles en matière de suivi du système du système et de ses composants incluent ceux que voici :

- *Processus* Rend visible des compteurs de performance apparentés aux différentes applications en cours d'exécution, par exemple l'activité CPU en mode utilisateur et noyau, le nombre de handles ouverts, etc.

- *Processeur* Rend visible des compteurs de performance concernant l'activité du processeur, par exemple le nombre d'interruptions par seconde, le nombre de DPC par seconde, etc.

- *Mémoire* Rend visible des compteurs de performance au sujet de la consommation mémoire, par exemple la quantité de mémoire disponible sur le moment, le nombre de pages et défauts de pages par seconde, etc.

Lorsque l'Analyseur de performances surveille un objet particulier, il peut suivre toutes les instances de tous les compteurs de cet objet. Les instances correspondent à toutes les instances d'un compteur particulier. Par exemple, si vous vous êtes engagés sur la piste de compteurs concernant les processus, vous avez le choix de suivre les instances de tous les processus ou celles de processus spécifiques.

Voici comment procéder en vue de sélectionner les compteurs à surveiller :

1. Démarrez l'Analyseur de performances (les méthodes pour ce faire ont été indiquées plus haut).

2. L'Analyseur de performances comporte plusieurs modes d'affichage. Assurez-vous d'afficher l'activité en cours en cliquant sur le bouton Affiche l'activité actuelle ou en appuyant sur CTRL+T. Pour basculer entre les types de vues (Ligne, Barre d'histogramme et Rapport), cliquez sur le bouton Modifier le type de graphique ou cliquez sur CTRL+G.

3. Pour ajouter des compteurs, cliquez sur le bouton Ajouter de la barre d'outils, ou appuyez sur CTRL+I. La boîte de dialogue Ajouter des compteurs apparait alors.

4. Dans le panneau Compteurs disponibles, les objets de performances sont classés par ordre alphabétique. Cliquez sur un objet pour le sélectionner : tous les compteurs correspondants le sont aussi. Si vous développez l'entrée d'un objet, vous affichez tous les compteurs associés et pouvez sélectionner des compteurs individuels en cliquant dessus.

5. Quand vous sélectionnez un objet ou l'un de ses compteurs, les instances en cours s'affichent dans le panneau Instances de l'objet sélectionné. Sélectionnez une instance particulière ou Toutes les instances. Pour rechercher une instance particulière, entre quelques lettres dans le champ déroulant sous Instances de l'objet sélectionné et cliquez sur Rechercher.

6. Lorsque vous avez sélectionné un objet ou un groupe de compteurs pour un objet, ainsi que les instances de l'objet, cliquez sur Ajouter pour intégrer ces compteurs à la liste Compteurs ajoutés.

7. Lorsque vous avez terminé, cliquez sur OK.

La manière la plus simple, et le meilleur moyen, d'appréhender les compteurs véhiculés par l'Analyseur de performance consiste à lire les explications disponibles dans la boîte de dialogue Ajouter des compteurs. Accédez dans cette optique audit élément d'interface et cochez ensuite la case Afficher la description dans l'angle inférieur gauche de la fenêtre.

Droits utilisateurs de l'analyseur de performances. Les droits utilisateurs de l'analyseur de performances sont spécifiés comme suit.

- *Administrateurs* Contrôle total local et distant.

- *Utilisateurs des journaux de performances* Peuvent accéder et mettre en journal des données de compteur de performance localement et à distance (créer, manipuler et voir les journaux).

- *Utilisateurs de l'Analyseur de performances* Peuvent accéder aux données de compteur de performance localement et à distance (voir les journaux).

Ensembles de collecteurs de données

Les ensembles de collecteurs de données sont des objets de performance qui permettent de gérer comme un tout plusieurs types de collecteurs de données : compteurs de performance, données de suivi d'événements et informations de configuration système (comme les valeurs de certaines clés spécifiques du Registre).

Introduits sous Windows Server 2008, les ensembles de collecteurs de données ressemblent dans une certaine mesure aux journaux de performance employés dans les versions précédentes de Windows. Ils sont cependant beaucoup plus élaborés, tant sur le plan de la réalisation que des schémas qu'ils rendent possibles.

Windows offre la possibilité d'interagir avec des ensembles de collecteurs de données au travers notamment de deux outils : Analyseur de performance et Logman.

Ensembles de collecteurs de données intégrés

Windows propose par l'intermédiaire de la console Performances plusieurs ensembles de collecteurs de données intégrés situés dans Ensembles de collecteurs de données\Système.

- *Performances du système* Regroupe un certain de compteurs de performance liés au disque, à la mémoire et au réseau, ainsi que différentes informations de suivi concernant le noyau. Utilisez cet ensemble de collecteurs pour auditer un ordinateur lent, et de la sorte diagnostiquer les facteurs qui diminuent ses performances générales.

- *Diagnostics du système* Enregistre toutes les informations comprises dans l'ensemble de collecteurs de données Performances du système, plus des métriques système détaillées. Utilisez cet ensemble de collecteur lorsque vous résolvez des problèmes de fiabilité, et faites par exemple face à du matériel défaillant, des pilotes défectueux ou des effondrements inopinés du système.

Pour employer un ensemble de collecteurs de données, effectuez dessus un clic avec le bouton droit de la souris, puis sélectionnez Démarrer. Dans la configuration par défaut, les ensembles Performances du système et Diagnostics du système s'arrêtent automatiquement au bout d'une minute.

Création d'un ensemble de collecteurs de données

Procédez comme suit afin d'enregistrer des informations de performances par le biais d'un ensemble de collecteurs de données.

1. Ouvrez Analyseur de performances.

2. Dans le volet gauche, effectuez un clic droit de la souris sur ensemble de collecteurs de données \Personnalisés, sélectionnez Nouveau, puis Ensemble de collecteurs de données.

3. Dans l'assistant Créer un nouvel ensemble de collecteurs de données, spécifiez un nom pour l'ensemble, idéalement qui reflète la nature des objets que vous voulez suivre, par exemple Performances du système ou Moniteur de l'état du processeur.

4. Sélectionnez l'option Créer manuellement et cliquez sur Suivant.

5. Sur la page Quels type de données inclure, l'option Créer des journaux de données est sélectionnée par défaut. Cochez la case Compteur de performance et cliquez sur Suivant.

6. Sur la page Quels compteurs de performance enregistrer dans un journal, cliquez sur Ajouter pour provoquer l'apparition de la boite de dialogue Ajouter des compteurs. Une fois les compteurs de performance selectionnés, cliquez sur le bouton Ok.

7. Toujours sur la page Quels compteurs de performance enregistrer dans un journal ?, personnalisez si besoin l'intervalle d'échantillonnage et sélectionnez l'unité de temps en secondes, minutes, heures, jours ou semaines. Cet intervalle spécifie quand collecter les nouvelles données. Cliquez sur Suivant pour poursuivre.

8. Sur la page Où enregistrer les données, spécifiez l'emplacement d'enregistrement des journaux (%systemdrive%\PerfLogs\Admin par défaut). Comme alternative, cliquez sur Parcourir et servez-vous de la boite de dialogue Rechercher un dossier afin de sélectionner un autre répertoire. (Voir note plus loin en ce qui concerne les proportions que peuvent atteindre les fichiers journaux.) Cliquez sur Suivant pour continuer.

9. Sur la page Créer l'ensemble de collecteurs de données, la zone Exécuter en tant que contient \<Par défaut \> de sorte à indiquer que le collecteur de données s'exécutera avec les privileges et les autorisations de l'utilisateur actuel. Cliquez éventuellement sur le bouton Modifier pour indiquer d'autres informations d'identification. Sélectionnez une des trois options avant de cliquer sur le bouton Terminer :

    a. *Ouvrir des propriétés pour cet ensemble de collecteurs de données* Personnalise immédiatement l'ensemble de collecteurs de données.

    b. *Démarrer maintenant cet ensemble de collecteurs de données* Commence sur-le-champ à enregistrer des données sans passer par la personnalisation de l'ensemble de collecteurs de données.

    c. *Enregistrer et fermer* Ferme l'ensemble de collecteur de données sans le démarrer. Vous pouvez modifier les propriétés et le démarrer à tout moment après l'avoir enregistré.

Notez que les paramètres que vous aurez choisis au niveau de l'intervalle d'échantillonnage ont un impact direct plus ou moins significatif sur la charge de travail demandée à l'ordinateur d'une part, et le volume des données résultant d'autre part (autrement dit la taille du fichier journal). C'est précisément la raison pour laquelle, au titre de ce second point, il est généralement conseillé d'utiliser comme lieu de stockage des journaux une partition différente de celle où le système est installée.

Création de collecteurs depuis l'invite de commandes

Vous pouvez créer des collecteurs de données depuis une invite de commandes à l'aide de l'utilitaire logman. Les commandes admissibles en la circonstance comptent entre autres celles voici.

- *Logman create counter* Cette commande crée un collecteur de données de compteur de performance. Par exemple, la commande logman create counter perf_log -c "\Processor{_ Total/)\% Processor Time" crée un compteur nommé perf_log qui enregistre les valeurs du compteur Pourcentage de temps processeur de l'instance de compteur Processeur(_Total).

- *Logman create trace* Cette commande crée un collecteur de données de suivi d'événement. Par exemple, la commande logman create trace trace_log -o c:\ trace_log_file crée un collecteur de données de suivi d'événement nommé trace_log et enregistre les résultats dans le fichier trace_log_file à la racine du disque.

- *Logman create alert* Cette commande crée une alerte de compteur de performances. Par exemple, la commande logman create alert alert -th "\ Processor(_Total)\% Processor Time\>90" crée une alerte nommée alert qui se déclenche lorsque le compteur de performances Pourcentage temps processeur de l'instance de compteur Processeur(_Total) excède 90.

Moniteur de fiabilité

Moniteur de fiabilité procure des informations en lesquelles jouxtent les aspects robustesse et stabilité de la station de travail. Les ordinateurs les moins frappés par des événements causés par des défaillances sont considérés comme stables et peuvent éventuellement obtenir l'index de stabilité maximum (10). Plus le système enregistre des dysfonctionnements de toute nature, plus la stabilité du système décroit. La valeur minimale de l'index est zéro. Les événements de fiabilité suivants peuvent être enregistrées dans le rapport de stabilité du système.

- Modification de l'horloge système
- Installations et désinstallations de logiciels
- Défaillances d'applications
- Défaillances matérielles
- Echec Windows
- Echecs divers

Moniteur de fiabilité conserve un historique d'un an des événements de stabilité système et de fiabilité.

Pour prendre les bonnes décisions, le moniteur de fiabilité produit des rapports détaillés permettant une juste compréhension des évènements qui ont contribué à abaisser l'index de stabilité système. Vous pouvez cliquer sur Afficher tous les rapports de problèmes pour afficher l'intégralité des problèmes rencontrés triés par logiciels.

Accéder au moniteur de fiabilité

Pour accéder au moniteur de fiabilité de Windows, rendez-vous dans le Panneau de configuration puis cliquez ensuite sur Sécurité et maintenance. Déroulez le panneau Maintenance puis cliquez sur Afficher l'historique de fiabilité. Plus simplement, saisissez *perfmon /rel* à l'intérieur d'une fenêtre d'invite de commandes. Une fenêtre semblable à la figure suivante s'affiche.

Pour obtenir des informations détaillées sur les problèmes rencontrés un certain jour ou une certaine semaine, il suffit de cliquer sur la bande temporelle correspondante et de lire les données affichées dans la partie inférieure de la fenêtre.

En bas de la fenêtre de Moniteur de fiabilité figurent plusieurs options :

- *Enregistrer l'historique de fiabilité* Permet d'enregistrer des détails complets sur l'ordinateur pour référence ultérieure. L'information est enregistrée comme rapport Moniteur de fiabilité au format XML.
- *Afficher tous les rapports de problèmes* Ouvre la fenêtre d'historique des problèmes, qui affiche l'historique de tous les problèmes rencontrés et leur statut.
- *Rechercher des solutions à tous les problèmes* Lance le processus d'examen automatique des problèmes. Une fois celui-ci achevé, Centre de maintenance est actualisé de sorte à montrer les solutions à tous les problèmes nouvellement découverts.

Index de stabilité système

Dans la perspective Windows, la stabilité d'un ordinateur est exprimée par un indice variant entre 1 (stabilité déplorable) et 10 (stabilité parfaite), qui est une mesure pondérée dérivant du nombre de problèmes matériels et logiciels observées pendant une durée historique continue (période roulante). Dès qu'un événement a une influence sur la stabilité globale de l'ordinateur, Windows procède à une réévaluation de ladite valeur. Au premier lieu, cet effet de rééquilibrage permet de refléter au plus près l'amélioration ou la dégradation qui s'est produite au fil du temps dans un index de stabilité du système ascendant ou descendant lorsqu'un problème de fiabilité a été rencontré, ou au contraire résolu.

Fondée sur des données collectées au cours de la durée de vie d'un système, la valeur d'index est calculée sur les 28 jours précédents. Les jours où le système est éteint, en état de veille ou d'hibernation ne sont pas pris en compte. S'il n'y a pas assez de données pour établir un index de stabilité système fiable et sérieux, Moniteur de fiabilité affiche une ligne pointillée (continue autrement).

A titre informatif, notez pour finir que l'index de stabilité ne constitue pas une mesure exacte de fiabilité. Il existe à cet égard un petit nombre de situations spéciales qui tendent à diminuer la valeur d'index mais rendent finalement le système plus fiable qu'il ne l'était par le passé. Entre dans cette catégorie notamment toute forme de mise à jour nécessitant à terme un redémarrage de l'ordinateur, y compris service packs et mises à niveau d'édition.

Windows System Assessment Tool (WinSAT)

L'outil d'évaluation système Windows (WinSAT, *Windows System Assessment Tool*) mesure les performances de divers équipements intégrés à la station de travail, et donne ce faisant un pressenti de l'utilisation du système d'exploitation s'exécutant dessus.

Les résultats des mesures effectués par WinSAT sont présentés sous la forme d'un indice de performance, sujet qui sera approfondi plus loin.

Emplacements du registre

WinSAT enregistre les résultats d'évaluation sous la clé HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\WinSAT.

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\WinSAT
    LastExitCode    REG_DWORD    0x0
    LastExitCodeCantMsg    REG_SZ    Évaluation terminée.
    LastExitCodeWhyMsg    REG_SZ    Les évaluations ou les autres opérations se sont terminées avec succès.
    CKCLStart    REG_QWORD    0x0
    PrimaryAdapterString    REG_SZ    NVIDIA GeForce GTX 770
    VideoMemoryBandwidth    REG_DWORD    0x2fe741c
    VideoMemorySize    REG_DWORD    0x7ae61000
    DedicatedVideoMemory    REG_DWORD    0x7c5d0000
    SharedVideoMemory    REG_DWORD    0xfe891000
    NumMonitors    REG_DWORD    0x1
    TotalMonitorPixels    REG_DWORD    0xe1000
    DedicatedSystemMemory    REG_DWORD    0x0
    MOOBE    REG_DWORD    0x2
    WindeployTimeDate    REG_QWORD    0x5446fa42
    GraphicsFPS    REG_DWORD    0x2d2404
    WsSwapResult    REG_DWORD    0x74c00
    WsSwapInterference    REG_DWORD    0xd0004
    WsSwapThroughput    REG_DWORD    0x74c00
    MFMediaMaxRunTime    REG_DWORD    0x1d4c0
    MediaMaxRunTime    REG_DWORD    0x1d4c0
    Disk0RandomReadThroughput    REG_DWORD    0x6588f
    Disk1RandomReadThroughput    REG_DWORD    0x5e1
    CpuUPExpressCompressThroughput    REG_DWORD    0x164
    MemoryThroughputMBS    REG_DWORD    0x4a9d
    RandomReadDiskScore    REG_DWORD    0x6588f
    SequentialReadDiskScore    REG_DWORD    0x79e43
    WinCRSScore    REG_DWORD    0x51
    MemoryScore    REG_DWORD    0x54
    CPUScore    REG_DWORD    0x54
    DWMScore    REG_DWORD    0x55
    D3DScore    REG_DWORD    0x63
    DiskScore    REG_DWORD    0x51
    HistoryVersionRead    REG_DWORD    0x0
    TimeLastFormalAssessment    REG_QWORD    0x1d1d7908df3b8e5
```

WEI (*Windows Experience Index*)

1. Ouvrez une fenêtre d'invite de commandes et tapez la commandes *winsat formal*. Votre ordinateur devrait alors travailler pendant quelques temps.

2. Depuis Explorateur de fichiers, rendez-vous sur C:\Windows\Performance\WinSAT\DataStore. En toute logique, ce dossier devrait héberger deux sous-ensembles de fichiers, l'un concernant les fichiers générés par le programme d'installation de Windows, l'autre les fichiers générés à l'instant.

3. Ouvrez le fichier Format.Assessment, par exemple en utilisant Internet Explorer, Edge ou tout autre outil via lequel visualiser facilement du contenu XML, et intéressez-vous plus particulièrement à la section nommée WinSPR (pour Windows System Performance Rating). Le résultat devrait ressembler à quelque chose comme ce qui suit.

```
<WinSPR>
 <SystemScore>8.15</SystemScore>
 <MemoryScore>8.4</MemoryScore>
 <CpuScore>8.4</CpuScore>
 <CPUSubAggScore>8.3</CPUSubAggScore>
 <VideoEncodeScore>8.4</VideoEncodeScore>
 <GraphicsScore>8.5</GraphicsScore>
 <Dx9SubScore>9.9</Dx9SubScore>
 <Dx10SubScore>9.9</Dx10SubScore>
 <GamingScore>9.9</GamingScore>
 .
 .
 .
</WinSPR>
```

Emplacements de registre pour WinSAT

WinSAT enregistre les résultats des évaluations auxquelles il procède par l'intermédiaire des diverses clés et valeurs situés sous HKLM\Software\Microsoft\Windows NT\CurrentVersion\WinSAT.

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\WinSat
    LastExitCode    REG_DWORD    0x0
    LastExitCodeCantMsg    REG_SZ    ...valuation terminÈe.
    LastExitCodeWhyMsg    REG_SZ    Les Èvaluations ou les autres opÈrations se sont terminÈes avec succËs.
    CKCLStart    REG_QWORD    0x0
    PrimaryAdapterString    REG_SZ    NVIDIA GeForce GTX 770
    VideoMemoryBandwidth    REG_DWORD    0x2fe741c
    VideoMemorySize    REG_DWORD    0x7ae61000
    DedicatedVideoMemory    REG_DWORD    0x7c5d0000
    SharedVideoMemory    REG_DWORD    0xfe891000
    NumMonitors    REG_DWORD    0x1
    TotalMonitorPixels    REG_DWORD    0xe1000
    DedicatedSystemMemory    REG_DWORD    0x0
    MOOBE    REG_DWORD    0x2
    WindeployTimeDate    REG_QWORD    0x5446fa42
    GraphicsFPS    REG_DWORD    0x2d2404
    WsSwapResult    REG_DWORD    0x65000
    WsSwapInterference    REG_DWORD    0xa0001
    WsSwapThroughput    REG_DWORD    0x65000
    MFMediaMaxRunTime    REG_DWORD    0x1d4c0
    MediaMaxRunTime    REG_DWORD    0x1d4c0
    Disk0RandomReadThroughput    REG_DWORD    0x6588f
    Disk1RandomReadThroughput    REG_DWORD    0x5e1
    CpuUPExpressCompressThroughput    REG_DWORD    0x164
    MemoryThroughputMBS    REG_DWORD    0x4a9d
    RandomReadDiskScore    REG_DWORD    0x6588f
    SequentialReadDiskScore    REG_DWORD    0x79e43
    WinCRSScore    REG_DWORD    0x51
    MemoryScore    REG_DWORD    0x54
    CPUScore    REG_DWORD    0x54
    DWMScore    REG_DWORD    0x55
    D3DScore    REG_DWORD    0x63
    DiskScore    REG_DWORD    0x51
    HistoryVersionRead    REG_DWORD    0x0
```

```
    TimeLastFormalAssessment    REG_QWORD    0x1d1d7908df3b8e5

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\WinSat\MediaEx
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\WinSat\WindowsExperienceIndexOemInfo
```

Composants de l'architecture Performances

La liste qui suit décrit quelques-uns des composants binaires spécifiques à l'architecture Performances.

- *pdh.dll* Sert à interagir avec les différences sources de compteurs de performance. Pour plus d'informations, consultez la section PDH (*Performance Data Helper*).
- *pdhui.dll* Procure les boîtes de dialogue réutilisables qui sont communes à toutes les applications liés aux performances.
- *pla.dll* DLL sous-jacente au service Journaux et alertes de performance (PLA, *Performance Logs and Alerts*).
- *perftrack.dll* Suivi des performances.
- *perfdisk.dll* Recueille des données liées à la performance des disques physiques et logiques.
- *relmon.dll* Moniteur de fiabilité.

PLA (*Performance Logs and Alerts*)

L'outil Journaux et alertes de performance prend en charge la définition d'objets de performance, de compteurs de performance et d'instances d'objets de performance. Il permet également de définir des intervalles d'échantillonnage pour l'analyse de données relatives aux ressources matérielles et aux services système.

Le service PLA intervient à la demande et démarre lorsqu'il est nécessaire de mettre en journal des données de performance. Il s'exécute dans un processus hôte partagé (svchost) avec comme nom abrégé PLA. Les informations de configuration du service PLA sont dans la clé nommé pla sous HKLM\System\CurrentcontrolSet \Services.

Paramètres de registre pour la configuration PLA.

```
HKEY_LOCAL_MACHINE\System\CurrentcontrolSet\Services\pla
    DisplayName    REG_SZ    @%systemroot%\system32\pla.dll,-500
    ErrorControl    REG_DWORD    0x1
    ImagePath    REG_EXPAND_SZ    %SystemRoot%\System32\svchost.exe -k LocalServiceNoNetwork
    Start    REG_DWORD    0x3
    Type    REG_DWORD    0x20
    Description    REG_SZ    @%systemroot%\system32\pla.dll,-501
    DependOnService    REG_MULTI_SZ    RPCSS
    ObjectName    REG_SZ    NT AUTHORITY\LocalService
    ServiceSidType    REG_DWORD    0x3
    RequiredPrivileges    REG_MULTI_SZ    SeImpersonatePrivilege
    FailureActions    REG_BINARY
 840300000000000000000000030000001400000001000000C0D4010001000000E093040000000000000000000

HKEY_LOCAL_MACHINE\System\CurrentcontrolSet\Services\pla\Configuration
HKEY_LOCAL_MACHINE\System\CurrentcontrolSet\Services\pla\Parameters
HKEY_LOCAL_MACHINE\System\CurrentcontrolSet\Services\pla\Security
```

PDH (*Performance Data Helper*)

L'API Performance Data Helper (Pdh.dll) offre des moyens d'accès structurés aux informations de performance et de diagnostic issues des différentes sources établies en la matière, y compris les compteurs de performance en temps réel, les fichiers binaires stockés et d'autres plus anciens formats de journaux de compteurs.

Les interfaces PDH présentent un niveau d'abstraction plus fort et une meilleure flexibilité que les fonctions du registre Windows. Les concepteurs de logiciels peuvent, entre autres, employer les interfaces PDH pour lire les valeurs des compteurs de performances, configurer, analyser et agréger des journaux de compteurs, et accéder à bien d'autres fonctionnalités indisponibles autrement.

Fonctions PHD

- *PdhOpenQuery* Crée une nouvelle requête utilisable pour gérer la collecte des données de performance.

Codes d'erreur

La liste qui suit énumère les codes d'erreur spécifiques à PDH. Ces valeurs sont définies dans le fichier d'entête pdhmsg.h.

- 0x00000000 - PDH_CSTATUS_VALID_DATA
- 0x00000001 - PDH_CSTATUS_NEW_DATA
- 0x800007D0 - PDH_CSTATUS_NO_MACHINE
- 0x800007D1 - PDH_CSTATUS_NO_INSTANCE
- 0x800007D2 - PDH_MORE_DATA
- 0x800007D3 - PDH_CSTATUS_ITEM_NOT_VALIDATED
- 0x800007D4 - PDH_RETRY
- 0x800007D5 - PDH_NO_DATA
- 0x800007D6 - PDH_CALC_NEGATIVE_DENOMINATOR
- 0x800007D7 - PDH_CALC_NEGATIVE_TIMEBASE
- 0x800007D8 - PDH_CALC_NEGATIVE_VALUE
- 0x800007D9 - PDH_DIALOG_CANCELLED
- 0x800007DA - PDH_END_OF_LOG_FILE
- 0x800007DB - PDH_ASYNC_QUERY_TIMEOUT
- 0x800007DC - PDH_CANNOT_SET_DEFAULT_REALTIME_DATASOURCE
- 0xC0000BB8 - PDH_CSTATUS_NO_OBJECT
- 0xC0000BB9 - PDH_CSTATUS_NO_COUNTER
- 0xC0000BBA - PDH_CSTATUS_INVALID_DATA
- 0xC0000BBB - PDH_MEMORY_ALLOCATION_FAILURE
- 0xC0000BBC - PDH_INVALID_HANDLE
- 0xC0000BBD - PDH_INVALID_ARGUMENT
- 0xC0000BBE - PDH_FUNCTION_NOT_FOUND

- 0xC0000BBF - PDH_CSTATUS_NO_COUNTERNAME
- 0xC0000BC0 - PDH_CSTATUS_BAD_COUNTERNAME
- 0xC0000BC1 - PDH_INVALID_BUFFER
- 0xC0000BC2 - PDH_INSUFFICIENT_BUFFER
- 0xC0000BC3 - PDH_CANNOT_CONNECT_MACHINE
- 0xC0000BC4 - PDH_INVALID_PATH
- 0xC0000BC5 - PDH_INVALID_INSTANCE
- 0xC0000BC6 - PDH_INVALID_DATA
- 0xC0000BC7 - PDH_NO_DIALOG_DATA
- 0xC0000BC8 - PDH_CANNOT_READ_NAME_STRINGS
- 0xC0000BC9 - PDH_LOG_FILE_CREATE_ERROR
- 0xC0000BCA - PDH_LOG_FILE_OPEN_ERROR
- 0xC0000BCB - PDH_LOG_TYPE_NOT_FOUND
- 0xC0000BCC - PDH_NO_MORE_DATA
- 0xC0000BCD - PDH_ENTRY_NOT_IN_LOG_FILE
- 0xC0000BCE - PDH_DATA_SOURCE_IS_LOG_FILE
- 0xC0000BCF - PDH_DATA_SOURCE_IS_REAL_TIME
- 0xC0000BD0 - PDH_UNABLE_READ_LOG_HEADER
- 0xC0000BD1 - PDH_FILE_NOT_FOUND
- 0xC0000BD2 - PDH_FILE_ALREADY_EXISTS
- 0xC0000BD3 - PDH_NOT_IMPLEMENTED
- 0xC0000BD4 - PDH_STRING_NOT_FOUND
- 0x80000BD5 - PDH_UNABLE_MAP_NAME_FILES
- 0xC0000BD6 - PDH_UNKNOWN_LOG_FORMAT
- 0xC0000BD7 - PDH_UNKNOWN_LOGSVC_COMMAND
- 0xC0000BD8 - PDH_LOGSVC_QUERY_NOT_FOUND
- 0xC0000BD9 - PDH_LOGSVC_NOT_OPENED
- 0xC0000BDA - PDH_WBEM_ERROR
- 0xC0000BDB - PDH_ACCESS_DENIED
- 0xC0000BDC - PDH_LOG_FILE_TOO_SMALL
- 0xC0000BDD - PDH_INVALID_DATASOURCE

- 0xC0000BDE - PDH_INVALID_SQLDB
- 0xC0000BDF - PDH_NO_COUNTERS
- 0xC0000BE0 - PDH_SQL_ALLOC_FAILED
- 0xC0000BE1 - PDH_SQL_ALLOCCON_FAILED
- 0xC0000BE2 - PDH_SQL_EXEC_DIRECT_FAILED
- 0xC0000BE3 - PDH_SQL_FETCH_FAILED
- 0xC0000BE4 - PDH_SQL_ROWCOUNT_FAILED
- 0xC0000BE5 - PDH_SQL_MORE_RESULTS_FAILED
- 0xC0000BE6 - PDH_SQL_CONNECT_FAILED
- 0xC0000BE7 - PDH_SQL_BIND_FAILED
- 0xC0000BE8 - PDH_CANNOT_CONNECT_WMI_SERVER
- 0xC0000BE9 - PDH_PLA_COLLECTION_ALREADY_RUNNING
- 0xC0000BEA - PDH_PLA_ERROR_SCHEDULE_OVERLAP
- 0xC0000BEB - PDH_PLA_COLLECTION_NOT_FOUND
- 0xC0000BEC - PDH_PLA_ERROR_SCHEDULE_ELAPSED
- 0xC0000BED - PDH_PLA_ERROR_NOSTART
- 0xC0000BEE - PDH_PLA_ERROR_ALREADY_EXISTS
- 0xC0000BEF - PDH_PLA_ERROR_TYPE_MISMATCH
- 0xC0000BF0 - PDH_PLA_ERROR_FILEPATH
- 0xC0000BF1 - PDH_PLA_SERVICE_ERROR
- 0xC0000BF2 - PDH_PLA_VALIDATION_ERROR
- 0x80000BF3 - PDH_PLA_VALIDATION_WARNING
- 0xC0000BF4 - PDH_PLA_ERROR_NAME_TOO_LONG
- 0xC0000BF5 - PDH_INVALID_SQL_LOG_FORMAT
- 0xC0000BF6 - PDH_COUNTER_ALREADY_IN_QUERY
- 0xC0000BF7 - PDH_BINARY_LOG_CORRUPT
- 0xC0000BF8 - PDH_LOG_SAMPLE_TOO_SMALL
- 0xC0000BF9 - PDH_OS_LATER_VERSION
- 0xC0000BFA - PDH_OS_EARLIER_VERSION
- 0xC0000BFB - PDH_INCORRECT_APPEND_TIME
- 0xC0000BFC - PDH_UNMATCHED_APPEND_COUNTER

- 0xC0000BFD - PDH_SQL_ALTER_DETAIL_FAILED
- 0xC0000BFE - PDH_QUERY_PERF_DATA_TIMEOUT

Event Tracing for Windows

Le suivi d'événements Windows (ETW, Event Tracing for Windows) est une infrastructure de journalisation qui fournit des données de trace et permet l'analyse dynamique des événements déclenchés par les différents composants du système, dont le noyau et divers pilotes de périphérique.

Dans les grandes lignes, ETW fonctionne sur la base d'un modèle de prestataires et d'abonnés. Des applications et des composants du système peuvent conformément à cette perspective enregistrer et rendre visible des données de trace, et d'autres applications et composants peuvent récupérer et exploiter ces données, le plus souvent à des fins de dépannage ou d'audit. Il est possible d'accéder aux informations émises depuis ETW de nombreuses façons : langages natifs et gérés, scripts, PowerShell, utilitaire Performances, entres autres.

Les applications utilisant ETW entrent dans l'une au moins des catégories suivantes :

- *Contrôleur* Un contrôleur démarre et arrête les sessions de journalisation.
- *Fournisseur* Un fournisseur définit des classes d'événements pour lesquelles il peut produire des traces.
- *Consommateur* Un consommateur sélectionne une ou plusieurs sessions de trace dont il veut lire les données.

Les fonctions mises en œuvre par ETW le sont par l'intermédiaire de la bibliothèque Advapi32 jusque Windows 8, et de la bibliothèque Sechost pour ce qui concerne les versions supérieures du système. Les interfaces idoines, prototypes d'appels et structures de données notamment, sont quant à elles déclarés dans le fichier en-tête EvnTrace.h du kit de développement Windows.

Contrôleurs

Les contrôleurs d'événements ETW orchestrent l'ensemble du processus de suivi des événements, notamment :

- Démarrage et arrêt des sessions de journalisation.
- Association de fournisseurs d'événements à une trace.
- Transmission d'informations de contrôle aux fournisseurs sur les événements à consigner.
- Contrôle des ressources nécessaires à une session de journalisation, par exemple le nombre de tampons mémoire alloués aux événements, les caractéristiques du fichier journal qui en résulte, etc.

Consommateurs

Le troisième et dernier type de composant qui joue un rôle dans le suivi des événements Windows est le consommateur, terme générique qui renvoie en réalité à toute application faisant appel à ETW dans l'optique d'être alimenté en informations.

Windows est de base livré avec plusieurs utilitaires susceptibles de traiter et afficher les données issues de sessions de trace, par exemple l'observateur d'événements ou le gestionnaire de performances.

Trace des événements noyau

Pour faciliter la compréhension des phénomènes qui se manifestent au sein du système d'exploitation, Windows intègre une trace spéciale préconfigurée, appelée Trace des événements noyau (Windows Kernel Trace), qui

véhicule des informations détaillées concernant les opérations émanant du noyau et des composants en périphérie.

Pour voir les différentes classes pour lesquelles la trace des événements noyau permet la journalisation, employez la commande logman query providers dans une invite de commandes avec privilèges.

```
C:\>logman query providers "Windows Kernel Trace"

Fournisseur                              GUID
------------------------------------------------------------------------
Windows Kernel Trace                     {9E814AAD-3204-11D2-9A82-006008A86939}

Valeur              Mot clé              Description
-------------------------------------------------------------------------
0x0000000000000001  process              Process creations/deletions
0x0000000000000002  thread               Thread creations/deletions
0x0000000000000004  img                  Image load
0x0000000000000008  proccntr             Process counters
0x0000000000000010  cswitch              Context switches
0x0000000000000020  dpc                  Deferred procedure calls
0x0000000000000040  isr                  Interrupts
0x0000000000000080  syscall              System calls
0x0000000000000100  disk                 Disk IO
0x0000000000000200  file                 File details
0x0000000000000400  diskinit             Disk IO entry
0x0000000000000800  dispatcher           Dispatcher operations
0x0000000000001000  pf                   Page faults
0x0000000000002000  hf                   Hard page faults
0x0000000000004000  virtalloc            Virtual memory allocations
0x0000000000010000  net                  Network TCP/IP
0x0000000000020000  registry             Registry details
0x0000000000100000  alpc                 ALPC
0x0000000000200000  splitio              Split IO
0x0000000000800000  driver               Driver delays
0x0000000001000000  profile              Sample based profiling
0x0000000002000000  fileiocompletion     File IO completion
0x0000000004000000  fileio               File IO


L'opération s'est bien déroulée.

C:\>
```

Structure d’un enregistrement

Une trace d’événement se compose d’un nombre variable de séquence d’enregistrements d’événement, chacun d’entre eux étant composé d’un entête standard (EVENT_TRACE_HEADER) qui réunit des informations communes à tous les événements, telles que l’heure à laquelle l’événement a eu lieu, ou les conditions dans lesquelles il a fait surface.

```
typedef struct _EVENT_TRACE_HEADER {
  USHORT        Size;
  union {
    USHORT FieldTypeFlags;
    struct {
      UCHAR HeaderType;
      UCHAR MarkerFlags;
    };
  };
  union {
    ULONG  Version;
    struct {
      UCHAR  Type;
```

```
      UCHAR  Level;
      USHORT Version;
    } Class;
  };
  ULONG         ThreadId;
  ULONG         ProcessId;
  LARGE_INTEGER TimeStamp;
  union {
    GUID      Guid;
    ULONGLONG GuidPtr;
  };
  union {
    struct {
      ULONG ClientContext;
      ULONG Flags;
    };
    struct {
      ULONG KernelTime;
      ULONG UserTime;
    };
    ULONG64 ProcessorTime;
  };
} EVENT_TRACE_HEADER, *PEVENT_TRACE_HEADER;
```

Windows Resource Protection

Pour préserver l'intégrité du système d'exploitation et maintenir au mieux ses capacités opérationnelles, Windows possède une technologie appelée Windows Resource Protection (WRP) qui permet de détecter tout changement suspect de la configuration ou au sein de fichiers, réduisant par là même la possibilité d'altérations nuisibles induites par des logiciels ou des utilisateurs.

Suite logique des fonctionnalité WFP (*Windows File Protection*) et SFP (*System File Protection*) des versions de Windows précédant Vista, WRP couvre les éléments que voici :

- Composants impliqués dans le processus de démarrage
- Fichiers exécutables et bibliothèques (DLL)
- Répertoires et fichiers
- Clés de registre essentielles au bon fonctionnement du système

En interne, la protection offerte par WRP l'est par l'intermédiaire de listes de contrôle d'accès minimales attribués aux paramètres, fichiers et emplacements critiques de la plateforme, listes qui sont configurés pour n'accorder l'accès qu'à un seul compte, en l'occurence le service Trusted Installer. Autrement dit, dans la perspective WRP, même les administrateurs et utilisateurs disposant d'informations d'identification administratives ne peuvent opérer de changements parmi les ressources protégées.

WRP possède une copie des fichiers indispensables au démarrage de Windows. Ces fichiers sont sauvegardés dans le répertoire servant de cache à l'emplacement \Windows\winsxs\Backup. Les fichiers du système protégés par WRP mais non utiles au démarrage ne sont pas copiés dans le répertoire de cache. La taille de ce répertoire ainsi que la liste des fichiers qu'il contient ne sont pas modifiables.

Vérificateur de ressources (Sfc)

Expression parmi les plus manifestes de la protection des ressources Windows, le Vérificateur de ressources est un outil en ligne de commandes qui vérifie que la plateforme possède les versions correctes des fichiers système protégés. Pour utiliser le Vérificateur de ressources, ouvrez une fenêtre d'invite de commandes en

tant qu'administrateur et tapez la commande Sfc suivie du paramètre approprié. Le tableau suivant liste les paramètres admissibles en matière d'interaction avec le programme Vérificateur de ressources.

Tableau 4.2. Paramètres du Vérificateur de ressources.

/SCANNOW	Analyse les fichiers protégés et répare ceux éventuellement endommagés.
/VERIFYONLY	Analyse les fichiers protégées mais n'effectue aucune pas de réparation.
/VERIFYFILE	Analyse le fichier dont le chemin d'accès est spécifié.
/OFFBOOTDIR	Pour les réparations hors ligne. Spécifie l'emplacement du répertoire d'amorçage hors ligne.
/OFFWINDIR	Pour les réparations hors ligne. Spécifie l'emplacement du répertoire Windows hors ligne.

Dans l'éventualité où Sfc découvre qu'un fichier sous sa tutelle a été remplacé, modifié ou est corrompu, l'utilitaire se met alors en quête d'une mouture correcte dudit fichier depuis le répertoire cache prévu à cet égard : \Windows\System32\Dllcache pour les versions de Windows antérieures à Vista et \Windows\Winsxs\Backup pour les versions de Windows depuis Vista. S'il en trouve une, l'élément incriminé est alors supprimé afin de céder la place à une version attendue, dûment approuvée - et par ailleurs signée.

Les investigations menées sous la houlette du Vérificateur de ressources, de même que les actions qui potentiellement en découlent (autrement dit les réparations), peuvent s'étendre à l'ensemble des fichiers protégées du système, ou à plus petite échelle ne concerner qu'un seul d'entre eux. Dans ce second cas de figure, indiquez auprès des commutateurs verifyfile et scanfile le le chemin d'accès complet à un fichier. Par exemple, sfc /verifyfile=c:\windows\system32\kernel32.dll vérifie si le fichier kernel32.dll est valide.

Sfc consigne les détails relatifs aux opérations effectuées par ses soins via le fichier CBS.log, situé sous \Windows \Logs\CBS. Chaque entrée dans ce journal en lien avec Sfc comporte la balise prédéfinie [SR](), de sorte que vous puissiez repérer facilement quelles procédures de vérification et/ou de réparation ont eu lieu. Pour effectuer des recherches parmi ce fichier, utilisez la commande findstr.

Threads système auxiliaires

Tôt dans la phase d'amorçage du système, Windows orchestre la création parmi le processus System de plusieurs threads, appelés *threads système auxiliaires*, chargés d'effectuer des taches pour le compte d'autres threads.

L'un des rôles premiers des threads système auxiliaires est d'agir en tant qu'intermédiaire entre des threads exécutés au niveau IRQL haut et des opérations dont la nature et/ou les caractéristiques les prédisposent à un IRQL inférieur. Par exemple, une routine DPC exécutée dans un contexte de thread à l'IRQL DPC/dispatch (car l'exécution DPC peut usurper n'importe quel thread du système) pourrait avoir besoin d'accéder au pool paginé. Etant donné, d'une part, qu'il est impossible de satisfaire aux défauts de page au niveau DPC/dispatch, et d'autre part, qu'une routine DPC ne peut pas diminuer l'IRQL, elle doit en l'occurrence passer le relais pour ce genre de traitement à un thread exécuté à un IRQL inférieur, ce à quoi répond précisément le mécanisme des threads auxiliaires.

Un autre avantage procuré par les threads système auxiliaires, du reste non le moindre, réside dans la diminution de la surcharge d'ordonnancement et de mémoire induite par la multiplication des threads dans le système.

Pilotes de périphériques et composants de l'exécutif interagissent avec les threads auxiliaires en premier lieu par l'intermédiaire des fonctions ExQueueWorkItem et IoQueueWorkItem, toutes deux chargées d'insérer un élément de travail (*work item*) dans une file d'attente où le système recherche des tâches à accomplir. Chaque élément de travail contient un pointeur vers une routine de rappel et un paramètre spécifiant les informations de contexte qui seront utiles quand ladite routine prendra vie. A un moment ultérieur, un thread système auxiliaire supprimera l'élément de travail de sa file et exécutera la routine idoine. Lorsque cette dernière se termine, le thread système auxiliaire vérifie si d'autres éléments de travail sont en attente et, si oui, les traite. Cette approche et ses effets se poursuivent indéfiniment.

Il existe trois types de thread système auxiliaires :

- Les *threads auxiliaires différés*, exécutés au niveau de priorité 12, prennent en charge des fonctions qui ne sont pas considérés comme temporellement critiques et peuvent voir leur pile remisée dans un fichier de pagination.
- Les *threads auxiliaires critiques*, exécutés au niveau de priorité 13, prennent en charge des fonctions temporellement critiques.
- L'unique *thread auxiliaire hypercritique* est exécuté au niveau de priorité 15 et a sa pile toujours présente en mémoire physique.

Le nombre de threads auxiliaires critiques est revue en permanence en fonction de la charge de travail de l'ordinateur. Chaque seconde, un sous-ensemble du balance set manager examine le besoin de nouveaux threads auxiliaires critiques et, si ce dernier est jugé réel, en crée un. Les threads créés par ce biais sont dits *threads auxiliaires dynamiques* ; la création d'un tel thread exige que soient remplies un certain nombre de conditions, à savoir :

- La file de travail critique contient des éléments à traiter.
- Le nombre de threads auxiliaires critiques inactifs est inférieur au nombre de processeurs dont dispose le système. (Les threads critiques inactifs le sont du fait d'attendre soit des éléments de travail, soit des objets dispatcher.)
- Le nombre de threads auxiliaires dynamiques est inférieur à 16.

Dans la configuration par défaut, le nombre initial de threads auxiliaires différés et critiques dépend de la capacité mémoire du système (valeur retournée par MmQuerySystemSize) et de la version de Windows. Les administrateurs peuvent apporter des modifications sur ces aspects en modifiant les valeurs AdditionalCriticalWorkerThreads et AdditionalDelayedWorkerThreads sous la clé de registre HKLM\SYSTEM \CurrentControlSet\Control\SessionManager\Executive.

La commande !exqueue du débogueur noyau affiche des informations détaillées en ce qui concerne les threads système auxiliaires.

Une chose importante à garder à l'esprit lors de l'utilisation des threads système auxiliaires se situe dans le fait que tous les composants noyau se partagent à cet égard les mêmes ressources (une file de travail ne peut satisfaire qu'un nombre limité de threads). Les opérations exécutées de la sorte, quand elles sont couteuses en temps, retardent l'exécution d'autres services, et peuvent de ce fait entrainer une diminution plus ou moins accentuée des performances du système. Par conséquent, l'exécution par l'intermédiaire d'éléments de travail devrait dans l'idéal ne concerner que des séquences de traitement relativement courtes.

LPC (*Local Procedure Call*)

L'appel de procédure locale (*Local Procedure Call*, mais aussi *Lightweight Procedure Call* ou *Local Inter-Process Communication*) est une technologie de communication inter processus pour transmission rapide de messages. S'imposant comme l'une des clés de voûte du modèle de communication de Windows (architecture client-serveur), LPC se rencontre très fréquemment partout dans le système d'exploitation, y compris les scénarios que voici :

- Les programmes applicatifs requièrent des services de la part des différents sous-systèmes de Windows via LPC. Par exemple, quelques API Windows aboutissent à l'envoi de messages au processus du sous-système Windows, Csrss.
- Les applications Windows qui emploient RPC (*Remote Procedure Call*) utilisent indirectement LPC quand elles spécifient RPC local (protocole ncalrpc), variante de RPC employée pour la communication entre processus d'un même système.

- Le processus serveur d'authentification de sécurité locale (Lsass), le gestionnaire de session (Smss) et le gestionnaire de contrôle des services (Scm) se servent de LPC pour communiquer avec leur processus client respectifs.
- Certains processus système s'appuient sur LPC de sorte à interagir entre eux, par exemple Winlogon et Lsass, ou encore le moniteur de références de sécurité et Lsass.

Essentiellement fondé sur le paradigme du passage de messages, LPC s'utilise en principe entre un processus serveur et un ou plusieurs processus client. Il est possible de créer une connexion LPC entre deux processus mode utilisateur, ou entre un composant mode noyau et un processus utilisateur.

Similaire par bien des aspects à RPC (*Remote Procédure Call*), lequel protocole est utilisé dans une vaste gamme de systèmes d'exploitation (y compris, du reste, Windows) pour faire du traitement distribué sur un réseau, LPC est par contraste optimisé pour l'usage local et est spécifique à Windows.

Les fonctions mettent en oeuvre LPC ont comme point d'ancrage ntdll et ntoskrnl.exe. Elles ne sont en l'occurence pas directement accessibles depuis l'API Windows.

Le composant LPC distingue deux types de ports : les ports de connexion et les ports de communication. Les ports de connexion sont des objets dotés d'un nom, visibles de tous les processus, qui donnent aux applications un moyen commode de mettre un canal de communication, lequel canal est dans la perspective LPC constitué d'une paire de ports privés de communication : un pour les message du client vers le serveur et l'autre pour les messages du serveur vers le client. Les canaux de communication supportent un mécanisme de rappel. Le client et le serveur peuvent donc accepter des requêtes tandis qu'ils attendent une réponse. Voici, en définitive, quelles sont les différentes extrémités d'une communication LPC. Voici, en définitive, quelles sont les différentes sorts de port potentiellement visibles parmi une communication LPC.

- *Port de connexion serveur* Port nommé créé par le serveur de façon à être à l'écoute des éventuelles connexions entrantes, et sur lequel les processus client peuvent se connecter pour dialoguer avec le serveur.
- *Port de communication serveur* Port non nommé permettant au processus serveur de dialoguer avec un processus client donné. Le serveur a un seul port de ce genre par client actif.
- *Port de communication client* Port non nommé utilisé par un processus client pour communiquer avec un processus serveur donné.
- *Port de communication anonyme* Port non nommé utilisé par deux threads d'un même processus pour communiquer.

LPC s'utilise généralement comme ceci :

1. Un serveur crée un objet de type port de connexion nommé (*NtCreatePort*) puis donne lieu à une écoute du port en question (*NtListenPort*).
2. Un client fait une demande de connexion à ce port (*NtConnectPort*).
3. Le serveur détermine si la demande est valide et, si elle l'est, l'accepte (*NtAcceptConnectPort*, suivie d'un appel à *NtCompleteConnectPort*). Il y a à ce moment création de deux ports sans nom, un port de communication serveur et un port de communication client. Le client et le serveur se servent alors de ces nouveaux ports pour leur communication.
4. Le serveur enregistre une boucle par l'intermédiaire de laquelle il continue à traiter de nouveaux messages et à y répondre en conséquence (*NtReplyWaitReceivePort*).
5. Le client envoie une nouvelle demande au serveur et attend que ce dernier lui donne suite (*NtRequestWaitReplyPort*). Cette étape et la précédente se répètent tout le temps du lien entre serveur et client.

LPC offre trois méthodes d'échange de messages :

- La première méthode, également la plus simple d'entre elles, est adaptée aux petits messages (d'au plus 256 octets). Dans ce cas, la file d'attente de messages du port est employée comme espace de stockage intermédiaire et les messages sont copiées d'un processus à l'autre. Le noyau effectue une copie du tampon contenant le message depuis l'espace d'adressage du processeur émetteur vers l'espace d'adressage système, puis réitère l'opération dans le sens nécessaire, autrement dit depuis l'espace d'adressage système vers l'espace d'adressage du processus récepteur.

- La seconde méthode, approprié aux messages plus grands, s'appuie sur un object section de mémoire partagée donnant lieu auprès des processus émetteur et récepteur à une représentation du canal de communication. Dans cette configuration, l'émetteur écrit tout d'abord dans la section partagée les informations qu'il souhaite publier, puis envoie dans la file d'attente du port un court message (en utilisant la méthode décrite plus haut) qui contient des pointeurs vers l'emplacement des données dans la section partagée, permettant de cette façon au receveur de les voir.

- La troisième technique de messages LPC, mise en place du moment qu'un serveur veut interagir avec des données trop volumineuses pour être hébergées dans une section partagée, s'appuie sur des API qui lisent/écrivent directement dans l'espace d'adressage d'un processus. Le composant LPC fournit un petit nombre de fonctions permettant au serveur de procéder de la sorte, et ainsi pouvoir accéder à une donnée dans un client. (Pour d'évidentes raisons de sécurité, l'inverse n'est bien entendu pas possible.)

Tableau 4.3. Services concernant LPC

NTSTATUS	*NtAcceptConnectPort* (PHANDLE PortHandle, PVOID PortContext, PPORT_MESSAGE ConnectionRequest, BOOLEAN AcceptConnection, PPORT_VIEW ServerView, OUT PREMOTE_PORT_VIEW ClientView)
NTSTATUS	*NtCreateWaitablePort* (PHANDLE PortHandle, POBJECT_ATTRIBUTES ObjectAttributes, ULONG MaxDataSize, ULONG MaxMessageSize, ULONG Reserved)
	NtImpersonateClientOfPort
NTSTATUS	*NtListenPort* (HANDLE PortHandle, PPORT_MESSAGE Message)
NTSTATUS	*NtQueryInformationPort* (HANDLE PortHandle, PORT_INFORMATION_CLASS PortInformationClass, PVOID PortInformation, ULONG PortInformationLength, PULONG ReturnLength)
	NtQueryPortInformationProcess
NTSTATUS	*NtReplyPort* (HANDLE PortHandle, PPORT_MESSAGE ReplyMessage)
NTSTATUS	*NtRequestPort* (HANDLE PortHandle, PPORT_MESSAGE LpcMessage)
NTSTATUS	*NtRequestWaitReplyPort* (HANDLE PortHandle, PPORT_MESSAGE LpcReply, PPORT_MESSAGE LpcRequest)

Tableau 4.4. Variables noyau concernant LPC

Variable	Type	Description
AlpcpPortMapping	GENERIC_MAPPING	Mappage générique des droits d'accès aux ports (A)LPC
LpcPortObjectType	POBJECT_TYPE	
LpcWaitablePortObjectType	POBJECT_TYPE	

APC (*Asynchronous Procedure Calls*)

Les APC (*Asynchronous Procedure Calls*) offrent un moyen aux programmes utilisateur et aux composants système d'exécuter du code dans le contexte (et partant, dans l'espace d'adressage) d'un thread spécifique. Étant donné que les APC sont mis en file pour exécution dans le contexte d'un thread choisi (par opposition à un thread arbitraire) et qu'ils sont exécutés à un IRQL inférieur au niveau DPC/dispatch, ils ne sont pas soumis aux mêmes restrictions qu'un DPC. Une routine APC peut ainsi acquérir des ressources, attendre des handles d'objet, subir des défauts de page, et ainsi de suite.

Il existe deux sortes d'APC : mode noyau et mode utilisateur, l'un et l'autre se différenciant essentiellement par les conditions attendues et par la temporalité dans laquelle le traitement qui résulte d'une interruption APC peut entrer en lice (du point de vue de l'ordonnancement). Ainsi, les APC mode noyau interrompent un thread et exécutent une procédure à l'insu et sans le consentement des parties concernées (autrement dit le thread

cible) ; les APC mode utilisateur sont quant à eux livrés à un thread seulement quand ce dernier se trouve dans une situation spécifique, à savoir l'état d'attente alertable. La catégorie des APC mode noyau se subdivise encore en deux sous-types : normal et spécial. Les APC mode noyau normales sont exécutées au niveau passif, les APC mode noyau spéciales au niveau APC. Un thread peut désactiver les APC normales par le biais de la routine KeEnterCriticalRegion, la livraison des APC spéciales étant dans cette configuration toujours maintenue. Un thread peut désactiver à la fois les APC normales et les APC spéciales soit par augmentation de l'IRQL au niveau APC (KeRaiseIrql), soit en s'en remettant à une région gardée (KeEnterGuardedRegion).

Plusieurs des mécanismes centraux pris en charge par Windows le sont via des APC. Le système emploie par exemple des APC pour initier l'exécution des threads, compléter les entrées/sorties, mettre fin à des processus et des threads, attacher un débogueur à une applications, gérer les expirations de minuterie, etc.

Du code mode utilisateur peut mettre en file d'attente un APC pour un thread par le biais de la fonction QueueUserAPC, laquelle s'appuie en interne sur le service système NtQueueUserApcThread. Suite à cet appel, le noyau alloue alors depuis le pool paginé un objet de contrôle KAPC, associé à l'APC en vue de le représenter. Dans l'éventualité où la quantité de mémoire disponible est trop faible pour répondre positivement à la demande, l'opération échoue, empêchant de ce fait la distribution de l'APC - ce qui dans le cas contraire serait susceptible d'entraver le fonctionnement efficace du système d'exploitation (relativement quand même à ce à quoi l'APC est employé). Pour anticiper un tel scénario, une application peut lors de son démarrage utiliser NtAllocateReserveObject afin de demander au noyau l'allocation préalable de la structure susmentionnée.

Comme nous l'avons déjà dit, les APC mode utilisateur sont livrés à un thread seulement quand il est dans un état d'attente alertable, ce qui peut potentiellement impliquer un délai plus ou moins long entre le moment où un APC est mis en file et le moment où la routine APC apparentée est réellement exécutée. Quand un thread en état alertable a un APC mode utilisateur en suspens, le noyau alerte le thread, passe le relais à la routine APC puis, du moment que cette dernière a fini, reprend l'exécution du thread. Dans l'éventualité où un thread en état alertable voit son attente satisfaite avant mise en file de l'APC (ce qui a pour effet en définitive de court-circuiter l'état alertable), la routine n'est pas exécutée, et le sera quand les circonstances se montreront propices (autrement dit la prochaine fois que le thread sera en état d'attente d'altertable.

Chaque APC est au niveau du noyau représenté par le biais d'une structure KAPC.

```
lkd> dt nt!_KAPC
   +0x000 Type             : UChar
   +0x001 SpareByte0       : UChar
   +0x002 Size             : UChar
   +0x003 SpareByte1       : UChar
   +0x004 SpareLong0       : Uint4B
   +0x008 Thread           : Ptr64 _KTHREAD
   +0x010 ApcListEntry     : _LIST_ENTRY
   +0x020 KernelRoutine    : Ptr64     void
   +0x028 RundownRoutine   : Ptr64     void
   +0x030 NormalRoutine    : Ptr64     void
   +0x020 Reserved         : [3] Ptr64 Void
   +0x038 NormalContext    : Ptr64 Void
   +0x040 SystemArgument1  : Ptr64 Void
   +0x048 SystemArgument2  : Ptr64 Void
   +0x050 ApcStateIndex    : Char
   +0x051 ApcMode          : Char
   +0x052 Inserted         : UChar
```

- *Mode processeur (ApcMode)* Mode d'exécution du processeur (noyau ou utilisateur) durant lequel la fonction spécifiée par le paramètre NormalRoutine doit être exécutée. Voir routine KeInitializeApc (ApcMode) ; structure de données KPROCESSOR_MODE.

- *État d'insertion (Inserted)* Voir routine KeInsertQueueApc.

- *Routine noyau (KernelRoutine)* Pointeur vers une fonction appelée à être exécuté dans le contexte du thread ciblé par un APC. Voir routine KeInitializeApc (KernelRoutine).

- *NormalRoutine* Voir routine KeInitializeApc (NormalRoutine).
- *RundownRoutine* Voir routine KeInitializeApc (RundownRoutine).
- *Taille (Size)* Taille qu'occupe une structure KAPC en mémoire. Voir routine KeInitializeApc.
- *Thread apparenté (Thread)* Pointeur vers le bloc KTHREAD du thread pour lequel l'APC est destiné.
- *Type d'objet (Type)* Voir énumération KOBJECTS.

Niveaux de requête d'interruption (IRQL)

Pour gérer la concurrence entre différents traitements, en particulier ceux liés aux interruptions, Windows impose en la matière son propre mécanisme de priorisation, connu sous le nom de *niveaux de requête d'interruption* (IRQL, Interrupt Request Level), qui servent à accorder une importance préférentielle aux requêtes issues des divers composants du système, autant physiques que logiques. Le noyau représente les IRQL en interne sous forme de numéros compris entre 0 et 31 sous x86, et entre 0 et 15 sur les systèmes 64 bits ; plus le numéro est élevé, plus la propriété est forte. (Autrement dit, plus l'IRQL est élevé, plus la requête à laquelle il se rattache revêt d'importance, et impose un traitement d'autant plus immédiat. Nous reviendrons ultérieurement sur ces aspects.)

La configuration IRQL de chaque processeur détermine quelles interruptions sont susceptibles d'être reçues, agissant concrètement à la manière d'un filtre. Ainsi, en fonction de l'IRQL en vigueur, un processeur ignore toutes les interruptions dont le niveau est inférieur ou égal à ce seuil, et ne sera interrompu que par des événements plus urgents, dont l'IRQL est supérieur. Par exemple, un IRQL de 2 bloque les interruptions de niveau 2, 1 et 0, tout en permettant celles de niveau 3 et supérieur.

Quand survient une interruption, le gestionnaire d'interception augmente l'IRQL du processeur pour lui donner la même valeur que celle assignée à la source de l'interruption (clavier, souris, thread, etc.). Cette augmentation masque toutes les interruptions d'un rang inférieur ou égal à cet IRQL (sur ce processeur uniquement). Les interruptions masquées sont soit traitées par un autre processeur, soient suspendus jusqu'à révision à la baisse de l'IRQL. Par conséquent, tous les composants du système, y compris le noyau et les pilotes de périphérique, s'efforcent de maintenir l'IRQL à son seuil le plus faible.

Pendant l'exécution d'un thread mode noyau, le thread ajuste (augmente ou abaisse) l'IRQL du processeur soit explicitement en faisant appel aux fonctions KeRaiseIrql et KeLowerIrql soit, plus communément, de manière indirecte en sollicitant des fonctions qui acquièrent des objets noyau de synchronisation, lesquelles au cours de leur cheminement normal actualisent d'elle-même la configuration IRQL.

Distinction niveaux d'interruption / priorités d'ordonnancement Niveaux de priorité d'interruption et priorités d'ordonnancement de thread, s'ils peuvent sembler apparentés (et sont dans la pratique souvent confondues à tort), n'ont en réalité pas du tout la même signification, et n'existent pas non plus à la même échelle. Une priorité d'ordonnancement détermine l'ordre d'exécution des threads dans la file d'attente d'un processeur, là où un IRQL entérine la capacité d'une interruption à bousculer, voire supplanter, l'exécution en cours.

Association entre interruptions matérielles et IRQL Sur les architectures modernes sous-jacentes à Windows, l'attribution des niveaux IRQL aux périphériques (et, partant, aux interruptions matérielles) repose sur la configuration des contrôleurs d'interruption (notamment le PIC hérité, ou l'APIC programmable). Chaque ligne d'interruption matérielle (IRQ) se voit assigné une priorité logique, que le noyau transpose ensuite vers un IRQL interne.

IRQL et mode utilisateur En mode utilisateur, le niveau IRQL est fixé à son seuil le plus bas (autrement dit au niveau 0). Seul le code exécuté en mode noyau est habilité à en ajuster la valeur, garantissant ainsi qu'aucune application ne puisse interférer avec le fonctionnement du système. Sur le plan factuel, cela signifie que l'IRQL d'un processeur est toujours au niveau le plus faible (autrement dit le plus sujet aux interruptions de toutes sortes) quand il exécute du code mode utilisateur, et ce n'est que quand le processeur exécute du code mode noyau que l'IRQL peut être supérieur.

IRQL prédéfinis

La hiérarchie IRQL se compose comme suit :

- L'IRQL le plus faible, à savoir le niveau *passif* (0), correspond à l'exécution normale du code mode noyau. À ce niveau, ainsi du reste que celui d'après, la planification des threads s'effectue conformément aux priorités qui les régissent du point de vue de l'ordonnanceur.
- Le niveau *appel de procédure asynchrone* (1) est utilisé conjointement à l'exécution d'APC noyau spéciaux, pour garantir leur bon déroulent sans interférence des niveaux inférieurs. (Nous reviendrons plus loin sur les APC) Comme au niveau passif, il est encore possible de passer d'un thread à un autre.
- Le niveau *aiguillage* (2) est celui dans lequel fonctionne la planification des threads, avec ce que cela suppose comme effets sur les stratégies système établies à cet égard. Ainsi, dans les faits, tant qu'un thread maintient l'IRQL à ce niveau, le planificateur ne peut plus l'interrompre pour passer à un autre thread, même si un autre de priorité supérieure est prêt à s'exécuter. Aucun changement ne peut avoir lieu jusqu'à ce que l'IRQL soit réajustée, ramenée en l'occurrence au niveau APC (1) ou, encore mieux, passif (0).
- Les IRQL *périphérique* (3-26 sur x86, 3-12 sur x64), servent à déterminer l'ordre de traitement des interruptions émanant des périphériques. Lorsqu'une telle interruption survient, le noyau augmente l'IRQL et appelle la routine de service (ISR) de l'interruption. (Pour en apprendre plus sur la façon dont le système concilie niveaux d'interruption et requêtes matérielles, consultez la section Association entre interruptions et IRQL.)
- Le niveau *horloge* est employé pour l'horloge du système, que le noyau utilise pour gérer l'heure ainsi que pour mesurer et allouer le temps processeur aux threads.
- Le niveau *alimentation* est associé à des interventions spécifiques du noyau en lien avec l'alimentation, comme la mise en veille ou la gestion des états ACPI. Héritage des spécifications originelles de Windows, cet IRQL ne semble concrètement jamais avoir été employé.
- Le niveau *haut* (31 sur x86, 15 sur x86) est employé de manière exceptionnelle par le noyau pour protéger certains chemins d'exécution hautement critiques, par exemple ceux liées à un effondrement du système ou aux interruptions non masquables (NMI). À ce niveau, plus rien ne peut interrompre l'exécution en cours ; toutes les interruptions sont inhibées, en conséquence de quoi le panel des actions envisageables est sérieusement limité.

Tableau 4.5. IRQL prédéfinis

IRQL	Valeur x86	Valeur x64	Usage typique
PASSIVE_LEVEL	0	0	Exécution normale (mode utilisateur ou noyau non critique)
APC_LEVEL	1	1	Livraison des APC
DISPATCH_LEVEL	2	2	Livraison des DPC, spinlocks
DIRQL	3–26	3–12	Interruptions matérielles, traitement ISR
PROFILE_LEVEL	27	13	Interruptions de profilage (statistiques CPU)
CLOCK_LEVEL	28	14	Interruptions d'horloge système (scheduler, timer)
IPI_LEVEL	29	15	Interruptions inter-processeurs (synchronisation CPU)
POWER_LEVEL	29	12 env.	Transitions d'alimentation, états ACPI
HIGH_LEVEL	31	15	Niveau maximal : blocage total des interruptions

Restrictions au niveau IRQL haut

La planification des threads sous Windows s'effectue au niveau IRQL *aiguillage* (2), excluant par conséquent toute possibilité de remplacer le thread en cours d'exécution par un autre. Plusieurs contraintes strictes s'imposent de ce fait au code exécuté à partir de ce niveau (IRQL 2 et supérieurs), à savoir :

- **Pas d'accès à la mémoire paginée** Le système n'étant en la circonstance plus en mesure de satisfaire de telles demandes, aucun défaut de page ne peut avoir survenir. Tout accès à de la mémoire potentiellement

paginée (pool noyau paginable ou tampons utilisateur) se verrait sanctionné par un effondrement immédiat du système. Seul le pool non paginé est garanti sûr.

- **Les objets de synchronisation ne peuvent être attendus** Attendre sur un objet de synchronisation, tel qu'un sémaphore ou un événement, suppose de suspendre le thread en cours et d'en planifier un autre, chose qu'il est dans une telle configuration impossible d'accomplir. Seule exception à cette règle : une attente non bloquante (la durée d'attente spécifiée est auquel cas nulle), qui se limite à interroger l'état de l'objet (signalé ou non).

En pratique, ces restrictions exigent que tout thread exécuté à un IRQL élevé (égal au supérieur à 2) ne reste actif que pour une fenêtre d'exécution aussi brève que possible.

Motivations du modèle IRQL

Comparativement à d'autres systèmes d'exploitation, où divers mécanismes jouent un rôle fonctionnel similaire à l'IRQL (masquage d'interruptions, suspension de la stratégie globale de planification, exécution différée, etc.), le choix d'un tel modèle dans Windows repose sur plusieurs motivations fondamentales :

- **Prédictibilité** En forçant une hiérarchie stricte, qui plus est profondément ancrée au coeur du système, l'IRQL garantit que certaines sections de code critiques ne seront jamais interrompues par des traitements de moindre priorité.

- **Fiabilité** Certaines routines, notamment celles qui manipulent des structures internes, exigent d'arriver à leurs termes sans interférence. Augmenter l'IRQL constitue alors un moyen sûr, commode qui plus est, d'écarter tout risque d'interruption nuisible.

- **Simplicité** La synchronisation explicite (via des verrous) induit sur le plan des performances un coût non négligeable. Aussi, Windows exploite l'IRQL comme une sorte de verrou implicite : l'élever au niveau *aiguillage*, par exemple, désactive (stoppe) la planification de threads, neutralisant de la sorte certains conflits sans recourir aux primitives classiques de synchronisation.

Kernel Transaction Manager (KTM)

Windows intègre en vue d'assurer la pérennité de certaines opérations critiques communes, telles l'actualisation du registre ou le remplacement de modules en cours d'exécution, un composant spécial dit gestionnaire de transaction noyau (KTM, Kernel Transaction Manager), qui confère à différents types d'objets, comme les fichiers, une sémantique transactionnelle.

Emprunté aux bases de données, le concept de transaction fait référence à une suite d'opérations qui est accomplie de telle manière à respecter les caractéristiques suivantes : (1) atomique, la totalité ou aucune des opérations composant la transaction doit être validée ; (2) consistante, les données doivent rester cohérentes à la fin de la transaction, que celle-ci ai été validée ou annulée ; (3) isolée, une transaction n'est pas visible par les autres tant qu'elle n'est pas validée, (4) durable, les effets d'une transition doivent persister même en cas d'incident.

KTM fait tout particulièrement preuve de sa pertinence lors de sous-tâches entraînant des modifications du système de fichiers ou du registre, à l'issue desquelles il est important de savoir si les traitement impliquées - et d'autant plus les requêtes d'E/S qui en découlent - ont correctement eu lieu, et s'ils ne l'ont pas été, comment les annuler. Les exemples abondent : mises à jour, installation de correctifs, chargement et déchargement de modules, sauvegarde de documents. etc.

De par les perspectives qu'il ouvre, KTM permet à différentes composants de s'élever au rang de gestionnaire de ressources transactionnelles. C'est le cas, par exemple, du système de fichiers NTFS ou du Registre, lesquels coordonnent par ce biais les opérations dont ils ont la charge. Les applications tierce partie peuvent également utiliser KTM dans l'optique de mettre en place leurs propres gestionnaires de ressources.

NTFS et le registre s'interfaçent avec KTM par le biais d'extensions, nommés respectivement TxF (Transactional NTFS) et TxR (Transactional Registry), dont les interfaces de programmation associés rendent visibles la prise en charge des transactions. Nous reviendrons par la suite plus en détail sur ces aspects.

KTM s'appuie pour ce qui est de son fonctionnement interne sur CLFS, lequel sous-système permet la création et l'exploitation de journaux de transactions.

KTM fournit plusieurs objets permettant de prendre en charge diverses opérations :

- L'object Transaction représente une séquence d'opérations à effectuer.
- L'objet Inscription (Enlistment) associe un gestionnaire de ressources et une transaction. Un gestionnaire de ressources crée un objet d'inscription lorsqu'il s'inscrit dans une transaction. Cet objet signale à KTM que le gestionnaire de ressources demande des notifications concernant la transaction spécifiée.
- L'objet Gestionnaire de ressources (Resource Manager) regroupe les transactions et transactions et les données sur lesquelles elles opèrent. Un système de traitement de transaction simple peut avoir un seul gestionnaire de ressources.
- L'object Gestionnaire de transactions (Transaction Manager) suit l'état de chaque transaction qui fait partie des gestionnaires de ressources associés.

Pour voir les gestionnaires de transactions enregistrés auprès du système, ouvrez une invite de commande avec privilèges élevés et exécutez ktmutil tm list.

```
C:\>ktmutil tm list
TmGuid                                 TmLogPath
````````````````````````````-- ````````````````````````````-
{ed3c026a-5af9-11e9-9d35-005056f11c37} \Device\HarddiskVolume4\Users\arnaud\NTUSER.DAT{1c3790b4-b8ad-11e8-aa21-
e41d2d101530}.TM
{ed3c026f-5af9-11e9-9d35-005056f11c37} \Device\HarddiskVolume4\Users\arnaud\AppData\Local\Microsoft\Windows
\UsrClass.dat{ed3c026d-5af9-11e9-9d35-005056f11c37}.TM
{e285a798-5b01-11e9-9d33-806e6f6e6963} \SystemRoot\System32\Config\TxR\{093539f2-b8b8-11e8-abe9-e41d2d0dac50}.TM
{e285a79c-5b01-11e9-9d33-806e6f6e6963} \Device\HarddiskVolume4\Windows\System32\config\DRIVERS{1c37907b-b8ad-11e8-aa21-
e41d2d101530}.TM
{e285a7ad-5b01-11e9-9d33-806e6f6e6963} \Device\HarddiskVolume4\Windows\ServiceProfiles\NetworkService\NTUSER.DAT{1c3790b4-
b8ad-11e8-aa21-e41d2d101530}.TM
{e285a7b3-5b01-11e9-9d33-b6ad6129f830} \Device\HarddiskVolume4\Windows\ServiceProfiles\LocalService\NTUSER.DAT{1c3790b4-
b8ad-11e8-aa21-e41d2d101530}.TM
{e1553cc7-5b03-11e9-9d38-806e6f6e6963}
{e895f0f3-5aff-11e9-ae5d-ad0802081af3} \Device\HarddiskVolume1\$Extend\$RmMetadata\$TxfLog\$TxfLog::KtmLog
{e895f0fe-5aff-11e9-ae5d-ad0802081af3} \Device\HarddiskVolume4\$Extend\$RmMetadata\$TxfLog\$TxfLog::KtmLog
```

La structure KENLISTMENT se présente comme telle :

```
lkd> dt nt!_KENLISTMENT
   +0x000 cookie   : Uint4B
   +0x008 NamespaceLink   : _KTMOBJECT_NAMESPACE_LINK
   +0x030 EnlistmentId   : _GUID
   +0x040 Mutex   : _KMUTANT
   +0x078 NextSameTx   : _LIST_ENTRY
   +0x088 NextSameRm   : _LIST_ENTRY
   +0x098 ResourceManager : Ptr64 _KRESOURCEMANAGER
   +0x0a0 Transaction   : Ptr64 _KTRANSACTION
   +0x0a8 State   : _KENLISTMENT_STATE
   +0x0ac Flags   : Uint4B
   +0x0b0 NotificationMask : Uint4B
   +0x0b8 Key   : Ptr64 Void
   +0x0c0 KeyRefCount   : Uint4B
   +0x0c8 RecoveryInformation : Ptr64 Void
   +0x0d0 RecoveryInformationLength : Uint4B
   +0x0d8 DynamicNameInformation : Ptr64 Void
```
```

```
+0x0e0 DynamicNameInformationLength : Uint4B
+0x0e8 FinalNotification : Ptr64 _KTMNOTIFICATION_PACKET
+0x0f0 SupSubEnlistment : Ptr64 _KENLISTMENT
+0x0f8 SupSubEnlHandle  : Ptr64 Void
+0x100 SubordinateTxHandle : Ptr64 Void
+0x108 CrmEnlistmentEnId : _GUID
+0x118 CrmEnlistmentTmId : _GUID
+0x128 CrmEnlistmentRmId : _GUID
+0x138 NextHistory      : Uint4B
+0x13c History          : [20] _KENLISTMENT_HISTORY
```

Hotpatch

Windows comprend, de façon à assurer la disponibilité du système lors de mises à niveau, une méthode de correction à l'exécution, dite hotpatch, qui permet l'application sans discontinuité et l'activation immédiate de correctifs.

Par le passé (Le Hot Patching est présent nativement depuis Windows Server 2008), une large majorité des opérations de maintenance système exigeait un réamorcage afin que les changements soient pris en compte. Loin d'être idéale, cette façon de procéder était d'autant plus pénalisante pour les milieux professionnels : interruptions de service, diminution de l'accessibilité serveur, et ainsi de suite. Les versions actuelles de Windows apportent par l'intermédiaire de la correction lors de l'exécution une réponse à ces lacunes.

La correction lors de l'exécution prend en charge un spectre étendu d'opérations :

- *Renommage* Remplacement d'une image (apparentée par exemple à une DLL ou à un pilote) chargée par le noyau ou les applications.
- *Échange d'objet* Renommage d'un objet parmi un espace de noms, ce qui est utile quand un fichier en cours d'utilisation doit auprès des applications qui en font usage changer de nom de manière transparente. Le nouveau fichier est alors disponible dans l'immédiat, par l'intermédiaire de l'ancien handle.
- *Correction de code* Remplacement du code machine hébergé au sein d'un fichier image par une version actualisée.
- *Actualisation de DLL système* Reconstruit les objets section nécessaires au partage de fichiers en mémoire. Quelques-unes des bibliothèques chargées dynamiquement qu'incorpore Windows le sont une fois seulement lors du démarrage, puis dupliquées dans l'espace d'adressage de chaque nouveau processus. (C'est le cas, par exemple, de Ntdll.dll.) Dans l'éventualité où un tel fichier a été mis à jour, le système en actualise les représentations sous-jacentes, de sorte à charger la version la plus récente.

Au travers des fonctionnalités qu'elle offre, la correction lors de l'exécution se prête à de nombreux usages, parmi lesquelles, en premier lieu le déploiement de correctifs, mais aussi l'instrumentation binaire, l'ajout de code, la pose de déviations au sein d'instructions (hooking). Les possibilités sont pour ainsi dire illimitées.

Débogage mode utilisateur

La prise en charge du débogage en mode utilisateur est répartie entre entre trois composants :

- *Infrastructure de débogage mode utilisateur* Fonctions internes nécessaires à l'enregistrement et à l'écoute des événements de débogage. Ces interfaces font partie intégrante de l'exécutif Windows, où sont elles identifiées par le biais du préfixe Dbgk.
- *Fonctions de support* Fonctions natives (bibliothèque Ntdll, préfixe associé DbgUi) responsables de l'encapsulation des mécanismes se rapportant aux objets de débogage, et de la communication avec de tels objets. Cela permet aux sous-système d'établir leurs propres sémantiques - et de ce fait, leurs propres services - en ce qui concerne le dépannage d'applications.

- *Fonctions de débogage* Sous-ensemble de l'API Windows dévolu au débogage de logiciels.

Le noyau Windows prend en charge le débogage en mode utilisateur via un objet spécialement dévolu à cet effet, dit objet débogage, qui centralise l'ensemble des informations essentielles dans le domaine.

Tableau 4.6. Événements de débogage

Événement	Signification	Déclenché par
DbgKmExceptionApi	Une exception s'est produite	Gestionnaire d'exception (KiDispatchException) lors d'une exception survenue en mode utilisateur
DbgKmCreateThreadApi	Un nouveau thread a été créé	Démarrage d'un thread en mode utilisateur
DbgKmCreateProcessApi	Un nouveau processus a été créé	Démarrage du premier thread mode utilisateur d'un processus
DbgKmExitThreadApi	Un thread a pris fin	Terminaison d'un thread
DbgKmExitProcessApi	Un processus a pris fin	Le dernier thread d'un processus s'est terminé
DbgKmLoadDllApi	Une DLL a été chargée	NtMapViewOfSection lorsque la section est un fichier image (DLL ou EXE)
DbgKmUnloadDllApi	Une DLL a été déchargée	NtUnmapViewOfSection lorsque la section est un fichier image (DLL ou EXE)
DbgKmErrorReportApi	Une exception doit être relayée à Windows Error Reporting	Gestionnaire d'exception (KiDispatchException) lors d'une exception survenue en mode utilisateur et que le débogueur n'a su traiter.

structure DEBUG_EVENT.

```
typedef struct _DEBUG_EVENT {
  DWORD dwDebugEventCode;
  DWORD dwProcessId;
  DWORD dwThreadId;
  union {
    EXCEPTION_DEBUG_INFO      Exception;
    CREATE_THREAD_DEBUG_INFO  CreateThread;
    CREATE_PROCESS_DEBUG_INFO CreateProcessInfo;
    EXIT_THREAD_DEBUG_INFO    ExitThread;
    EXIT_PROCESS_DEBUG_INFO   ExitProcess;
    LOAD_DLL_DEBUG_INFO       LoadDll;
    UNLOAD_DLL_DEBUG_INFO     UnloadDll;
    OUTPUT_DEBUG_STRING_INFO  DebugString;
    RIP_INFO                  RipInfo;
  } u;
} DEBUG_EVENT, *LPDEBUG_EVENT;
```

Chapitre 5. Mécanismes de gestion

Ce chapitre procure une vue détaillée de quelques-uns des dispositifs fondamentaux de Microsoft Windows en rapport avec l'administration et la configuration du système :

- Registre
- Services
- WMI (*Windows Management Instrumentation*)
- Événements

Registre

Jouant un rôle clé dans la supervision et le contrôle des systèmes, composants et logiciels Windows, le registre sert de lieu d'hébergement à moult paramètres de configuration, qu'ils soient de niveau global ou spécifiques à l'utilisateur. Cela inclut, par exemple, les options d'amorçage du noyau, les paramètres d'applications et de pilotes, les préférences et affinités des utilisateurs, leur profil de sécurité, et bien d'autres.

Une large majorité des utilisateurs voient le registre tel un ensemble de données statiques rangées sur le disque dur, à la façon, par exemple, d'une boite de dépôt. Si elle a de solides raisons d'exister, cette perceptive est aussi très réductrice, dans la mesure où, ainsi que vous le verrez plus loin, le registre constitue en réalité une passerelle vers toutes sortes de données mouvantes dans le temps : état du système, liste des services actifs, reflets des structures gérées en mémoire par l'exécutif et le noyau, compteurs de performance, etc.

Tableau 5.1. Fonctions Windows concernant le registre

Fonction	Description
RegCloseKey	Ferme la clé spécifiée.
RegCreateKey	Crée, ou éventuellement ouvre, une clé de registre.
RegDeleteKey	Supprime une clé.
RegEnumKeyEx	Liste les sous-clés présentes sous une clé.
RegEnumValue	les valeurs contenues dans une clé.
RegOpenKeyEx	Ouvre une clé de registre.
RegQueryValueEx	Interroge la clé spécifié.
RegSetValueEx	Crée ou modifie les valeurs contenues dans une clé.

Origines du registre

S'il dispose *aujourd'hui* bel et bien d'une place prépondérante au sein des systèmes Microsoft, le registre n'a pas toujours toujours endossé des responsabilités aussi lourdes. Les versions initiales de Windows se passaient du reste totalement de ce genre de procédé, et les quelques déclinaisons qui suivirent (jusqu'à Windows 95 en réalité) ne l'employaient que de façon très superficielle.

Avant Windows 3.1, les plateformes Windows utilisaient pour la mémorisation des paramètres relatifs aux composants systèmes et aux programmes des fichiers INI. Hébergeant des données de nature uniquement textuelle (donc facilement manipulable au travers une large gamme de logiciels), ces fichiers étaient divisés en sections comportant chacune un nombre plus ou moins important de déclarations. Chaque donnée était déclarée par un nom de variable, suivi par un signe égal, lui-même suivi par la valeur de la variable. Les noms des sections étaient indiqués entre crochets et tout ce qui suivait ce nom appartenait à la section. Partant de cette nomenclature, un fichier INI ressemble à ce qui suit.

```
[section 1]()
Parameter1=Value1
```

```
Parameter2=Value2
Parameter3=Value3

Avec l'apparition d'OLE dans Windows 3.1, et plus généralement des composants basés sur COM, les besoins en stockage et
 consultation de paramètres divers augmentèrent considérablement, cela avec comme effet secondaire d'attirer l'attention
 sur les limites, tant pratiques que conceptuelles, des fichiers INI. Depuis la sortie du système d'exploitation Windows
 NT, en 1993, l'essentiel des éléments de la configuration est enregistré à l'échelle d'une base de données dévolue
 spécifiquement à cet usage, en l'occurence le registre.
```

Composants du registre

Les éléments suivants sont à considérer pour leurs relations (plus ou moins directes) avec le registre :

- **Gestionnaire de configuration** Le gestionnaire de configuration est le composant de l'exécutif Windows chargé d'implémenter la base de données du registre.
- **Ruches** La sauvegarde physique (sur disque) de la base de données du registre se présente comme un ensemble de fichiers appelés ruches. Chaque ruche correspond à un fichier, et contient une arborescence de registre.
- **Regedit** Windows inclut nativement plusieurs outils afin de superviser le registre, dont Regedit, qui permet d'agir de manière très fine dans la configuration du système, et met entre les mains d'utilisateurs avancés la possibilité de modifier la base, et d'attribuer des droits spécifiques sur les clés du registre. Un portait plus large de l'Editeur de Registre est donné plus loin dans ce chapitre.
- **Clés, sous-clé, valeurs** La base de données du registre est faite de clés, de sous-clés et de valeurs. Une clé peut contenir d'autres clés (sous-clés) ou des valeurs. Une valeur contient des données.
- **Synchronisation de ruche** Un thread système fait de la synchronisation de ruche, opération rendue nécessaire par le fait que les données mémoire correspondant au registre doivent un temps après leur modification être conservées (écrites) sur le disque (ce que l'on appelle l'écriture différée).

Scénarios d'utilisation du Registre

Windows sollicite les données de configuration stockées dans le registre en de multiples occasions :

- **Amorçage du système** Lors de l'amorçage, le système lit des paramètres spécifiant quels sont les pilotes de périphérique et les services à charger et comment les divers composants (gestionnaire de mémoire, gestionnaire de processus, et d'autres.) interagissent entre eux et optimisent le comportement du système. La structure initiale du registre et nombre de paramètres par défaut sont définis par un prototype du registre qui est copié sur le disque lors d'une nouvelle installation de Windows.
- **Ouverture de session** Lors d'une ouverture de session, le registre sert à la mise en conformité de l'environnement par rapport aux préférences de l'utilisateur connecté. Cela comprend, par exemple, les désignations (les lettres) de lecteur réseau, l'économiseur d'écran, l'environnement de bureau (comprendre l'explorateur), le style des menus, le papier peint du bureau, l'emplacement des icônes, etc.
- **Démarrage d'application** Lors de son démarrage, une application peut éventuellement orienter son comportement en fonction de paramètres de niveau système, en rapport par exemple à des informations de licence ou à d'autres données qualitatives, ou de niveau utilisateur, par exemple l'emplacement des menus et barres d'outils ou la liste des derniers documents manipulés. Certaines applications surveillent leurs données de configuration dans le registre et lisent les paramètres modifiés quand elles détectent un changement.
- **Installation d'application** Le registre permet aux applications de disposer d'un endroit unique où conserver leurs options de configuration. Les programmes d'installation des applications créent des paramètres et des profils qui reflètent les choix de configuration de l'installation.

- **Ajout de nouveau pilote** Lors de l'installation d'un pilote de périphériques, le système de prise en charge Plug and Play créé dans le registre des paramètres indiquant au gestionnaire d'E/S comment démarrer et configurer le pilote.
- **Création de profils utilisateur** La création de nouveaux utilisateurs alimente implicitement le registre, dans la mesure où c'est à cet endroit qu'est rangé l'ensemble des données spécifiques à un utilisateur et à ses préférences.
- **Modification de la configuration** Que vous modifiiez des paramètres de niveau système ou de niveau application, les modifications idoines sont souvent reportées dans le registre. Elles sont de cette façon pérennes et existent au delà une réinitialisation du système.

Organisation du registre

Le Registre est agencée sous une forme analogue à celles de volumes disque, faite en l'occurence d'éléments empruntés aux systèmes de fichiers. Les informations dans le Registre sont rangées dans un système en arbre contenant des clés et des valeurs, équivalents des répertoires et des fichiers. Une clé peut contenir d'autres clés (sous-clés) ou des valeurs. Une valeur, en revanche, contient des données. Les clés de premier niveau (dans l'arbre) sont les clés racine.

Les conventions de nommage à l'intérieur du Registre s'inspirent de celles à l'oeuvre dans une partition Windows. Ainsi, une valeur de nom y stockée dans une clé de nom x est identifié par le nom x\y. On retrouve comme élément utilisé pour la classification la barre contre-oblique. S'il vient seul (\) ce caractère représente la racine du registre. Autrement, il permet de repérer sans ambiguïté clés et valeurs de registre.

Types de données du registre

A l'instar des langages de programmation, le registre Windows offre plusieurs types primitifs grâce auxquels représenter (et donc manipuler) des informations de toutes sortes, tels que chaînes, compteurs, intervalles, booléens (valeurs oui/non), liens symboliques (clé pointant vers une autre), etc.

Le tableau ci-dessous apporte un bref éclairage sur les quelques types de données (au nombre, du reste, de 15) admis dans le Registre, et les sections qui suivent des informations complémentaires en ce qui concerne chacun d'entre eux.

Tableau 5.2. Types des valeurs du registre

Type	Description
REG_BINARY	Données binaires de longueur variable
REG_DWORD	Nombre de 32 bits
REG_DWORD_BIG_ENDIAN	Nombre de 32 bits avec octet de poids fort en premier
REG_DWORD_LITTLE_ENDIAN	Nombre de 32 bits avec octet de poids faible en premier (équivaut à REG_DWORD)
REG_EXPAND_SZ	Chaîne Unicode de longueur variable
REG_FULL_RESOURCE_DESCRIPTOR	Description de ressource matérielle
REG_LINK	Lien symbolique
REG_MULTI_SZ	Tableau de chaînes
REG_NONE	Aucun type
REG_QWORD	Nombre de 64 bits
REG_QWORD_BIG_ENDIAN	Nombre de 64 bits avec octet de poids fort en premier
REG_QWORD_LITTLE_ENDIAN	Nombre de 64 bits avec octet de poids faible en premier (équivaut à REG_QWORD)
REG_RESOURCE_LIST	Description de ressource matérielle
REG_RESOURCE_REQUIREMENTS_LIST	Exigences de la ressource
REG_SZ	Chaîne Unicode de longueur fixe

REG_BINARY

Les valeurs REG_BINARY renferment des données binaires brutes, telles que des clés cryptographiques ou des mots de passe chiffrés. Par nature, de telles informations n'incorporent pas de commodité ni au niveau de l'analyse ni de la présentation (pas de caractère de terminaison, d'expansion de chaines, etc.) Une large majorité des utilitaires permettant de visualiser le registre montrent ces informations au format hexadécimal. Pour voir concrètement à quoi ressemble des données du type REG_BINARY, consultez par exemple la valeur nommée FailureActions sous la clé HKLM\SYSTEM\CurrentControlSet\Services\lmhosts.

REG_DWORD

Le type REG_DWORD stocke un nombre entier sur 4 octets (32 bits). Moults paramètres sont dépeints à l'échelle de ce format, par exemple des compteurs, intervalles, booléens (valeurs oui/non), masques binaires, etc. Les valeurs connexes sont DWORD_LITTLE_ENDIAN, où l'octet le moins important se trouve à l'adresse la plus basse, et REG_DWORD_BIG_ENDIAN, où l'octet le moins important se trouve à l'adresse la plus haute.

REG_EXPAND_SZ

Les valeurs REG_EXPAND_SZ renferment des chaînes extensibles qui elles-mêmes intègrent des variables résolues lors de leur utilisation. Les applications peuvent intégrer un jeton spécial dans la chaîne de caractères, signifiant que la chaîne contient un élément dont le système devra se charger de remplacer le contenu. Le jeton est une variable dont le nom est délimité par le caractère %. Par exemple, la variable d'environnement %SystemRoot% est un type REG_EXPAND_SZ représentant le dossier Windows effectif. Quand le système lit la chaine de caractères, il remplace %SystemRoot% par le chemin d'accès complet au répertoire d'installation du système d'exploitation.

REG_FULL_RESOURCE_DESCRIPTOR

Le type REG_FULL_RESOURCE_DESCRIPTOR enregistre des informations sur les ressources système requises par un périphérique particulier. Etant donné sa nature, ce type n'est visible que parmi les sous-clés de HKLM \HARDWARE.

REG_LINK

Le type REG_LINK instaure dans le registre différents niveaux d'indirection et, à l'instar des liens symboliques des systèmes de fichier, permet à une clé de pointer de façon transparente vers une autre clé ou valeur. Quand vous traversez le registre via un lien, la recherche de chemin continue à partir de la cible du lien. Par exemple, si \Root1\Link à une valeur REG_LINK \Root2\RegKey, et que RegKey contient la valeur RegValue, alors deux chemins identifient RegValue : \Root1\Link\RegValue et \Root2\RegKey\RegValue. Windows fait un usage intensif des liens : trois des six clés de premier niveau définies dans le registre sont en réalité des liens vers des sous-clés des trois autres clé racine. Les liens sont recréés dynamiquement après chaque réamorçage du système.

REG_MULTI_SZ

Le type REG_MULTI_SZ est un ensemble de valeurs REG_SZ et correspond à ce titre à une collection de chaine. Les valeurs qui contiennent des listes ou plusieurs valeurs dans un format que l'utilisateur peut lire sont généralement enregistrées sous cette forme. Les entrées sont séparées par des espaces, des virgules ou autre forme de ponctuation - c'est donc aux applications de savoir comment extraire une portion spécifique d'une valeur REG_MULTI_SZ.

REG_NONE

Très certainement le plus basique des types de données exprimables au sein du registre, REG_NONE a trait au seul fait qu'un emplacement soit laissé vacant ou est au contraire occupé. Les composants utilisant ce

schéma le font généralement dans l'optique de vérifier la présence d'une clé ou d'une valeur pour adapter leur comportement au moment de l'exécution.

REG_QWORD

Le type REG_QWORD représente des données par un nombre entier 64 bits. Les valeurs connexes sont REG_QWORD_LITTLE_ENDIAN (nombre de 64 bits avec octet de poids faible en premier) et REG_QWORD_BIG_ENDIAN (nombre de 64 bits avec octet de poids fort en premier). Notez que, Windows étant conçu pour fonctionner sur des architectures informatiques little-endian, les valeurs REG_QWORD et REG_QWORD_LITTLE_ENDIAN sont équivalentes.

REG_RESOURCE_LIST

Le type REG_RESOURCE_LIST stocke des informations sur les ressources utilisées par un pilote de périphérique matériel ou l'un des périphériques physiques qu'il contrôle. Ces données sont détectées et écrites par le système dans l'arborescence HKLM\HARDWARE\ResourceMap.

REG_SZ

Le type REG_SZ correspond à une chaine de caractères de longueur variable. Les valeurs REG_SZ renferment généralement des messages, des noms de fichier, des données de chemin, etc.

Structure du registre

Si les informations publiées via le Registre sont de forme et de nature diverses (un système comme Windows est entouré d'un nombre important de logiciels et matériels, donc a fortiori de très nombreux paramètres de configuration), il est possible de dresser la cartographie dudit lieu en fonction des données qui y sont stockées.

Clés volatiles

Certaines clés et valeurs du Registre sont volatiles, c'est-à-dire qu'elles peuvent disparaitre à tout moment, et ne perdurent surtout pas au-delà de l'extinction de la machine. Par exemple, aucune des données que renferme la clé HKLM\HARDWARE n'ont d'existence sur le disque, mais uniquement en mémoire. Les clés volatiles sont pratiques pour les données qu'il n'est pas nécessaire de conserver entre deux démarrages.

Clés racines

Le registre est organisé à la façon d'une structure arborescente à six niveaux, dites clés racines (il est impossible d'en ajouter ou supprimer), montrées au tableau suivant.

Tableau 5.3. Clés racine

Clé racine	Description
HKEY_CLASSES_ROOT	Contient les données d'associations de fichier et de déclarations d'objet COM
HKEY_CURRENT_CONFIG	Contient des données relatives au le profil matériel courant
HKEY_CURRENT_USER	Contient les données associées à l'utilisateur courant
HKEY_LOCAL_MACHINE	Contient des informations de configuration spécifiques à l'ordinateur
HKEY_PERFORMANCE_DATA	Contient des données concernant les performances.
HKEY_USERS	Contient les données concernant les comptes/profils utilisateur

Les noms des clés racine commencent par la lettre H, car ils représentent des handles (H) Windows vers les clés (KEY). Pour des raisons de concision, les noms des clés racine sont souvent montrés sous une forme plus brève, par exemple HKCU pour HKEY_CURRENT_USER, HKLM pour HKEY_LOCAL_MACHINE, et ainsi de suite.

Les sections qui suivent apportent plus de précisions sur chacune des clés racine.

HKEY_CURRENT_USER

La ruche HKEY_USERS contient une clé, enregistrée sous forme de SID, pour chaque compte d'utilisateur sur l'ordinateur. Quand un utilisateur ouvre une session, le système repère le SID du compte auquel le profil correspond et charge depuis HKEY_USERS vers la branche HKEY_CURRENT_USER la portion du profil d'utilisateur.

HKEY_CURRENT_USER

La clé racine HKEY_CURRENT_USER contient des données relatives aux préférences et à la configuration logicielle de l'utilisateur actif. Les informations dans cette clé sont créées pendant le processus d'ouverture de session, ce à partir du profil de l'utilisateur ayant ouvert la session courante, profil situé sur le disque dans \Users \<username\>\Ntuser.dat.

Chaque fois qu'un profil utilisateur est chargé (par exemple, lors d'une ouverture de session ou lors du démarrage d'un service exécuté sous le contexte d'un certain utilisateur), HKEY_CURRENT_USER est créé sous forme de lien vers la clé associée à l'utilisateur sous HKEY_USERS. (Pas étonnant, donc, que toutes les données trouvées dans la clé HKEY_CURRENT_USER soient par défaut une copie de la sous-clé HKEY_USERS\nom d'utilisateur, où nom d'utilisateur est le nom de l'utilisateur actif.) Le tableau suivant énumère quelques-unes des sous-clés de HKEY_CURRENT_USER.

Tableau 5.4. Sous-clés de HKEY_CURRENT_USER

Sous-clé	Description
AppEvents	Associations son/événement (par exemple, quand un message d'erreur s'affiche).
Console	Paramètres de fenêtre d'invite de commandes (hauteur, largeur, couleurs, etc.-)
Control Panel	Paramètres concernant l'économiseur d'écran, l'aspect du bureau, le clavier et la souris, plus quelques paramètres concernant l'accessibilité et l'internationalisation/régionalisation
Environnement	Variables d'environnement
Keyboard Layout	Paramètres concernant la disposition du clavier (par exemple, US ou FR)
Network	Mappages et paramètres des lecteurs réseau
Printers	Paramètres concernant les connexions aux imprimantes
RemoteAccess	Adresses et profils pour l'accès réseau à distance
Software	Préférences logicielles spécifiques à l'utilisateur

HKEY_LOCAL_MACHINE

HKEY_LOCAL_MACHINE (HKLM) est une sous arborescence du registre où sont définit l'ensemble des paramètres associés à la configuration de l'environnement local. La clé racine englobe toutes les sous clés de configuration de niveau système : HARDWARE (processeur, carte mère…), SOFTWARE (options des logiciels installés), SYSTEM (dernière bonne configuration connue), SECURITY (Sam et stratégies de sécurité locales), et BCD (données d'amorçage).

```
C:\>reg QUERY HKEY_LOCAL_MACHINE
```

HKEY_LOCAL_MACHINE\BCD00000000 HKEY_LOCAL_MACHINE\HARDWARE HKEY_LOCAL_MACHINE\SAM HKEY_LOCAL_MACHINE\SECURITY HKEY_LOCAL_MACHINE\SOFTWARE HKEY_LOCAL_MACHINE\SYSTEM

HKLM\BCD contient les données de configuration définies pour l'amorçage du système, lesquelles suppléent au fichier boot.ini des anciennes versions de Windows. Plus proche de l'époque actuelle, les fichiers de données de configuration de démarrage (BCD, Boot Configuration Data) fournissent un répertoire d'informations sur les applications de démarrage et les paramètres de ces dernières. Les magasins BCD remplissent essentiellement les mêmes fonctions que leurs ainés (les fichiers ini donc), mais offrent un éventail d'options de démarrage plus large et améliorent la prise en charge de l'écriture de script.

HKLM\SAM contient les informations sur les comptes et groupes locaux, par exemple les mots de passe utilisateur, les associations de groupe et de domaine. HKLM\SECURITY contient les stratégies de sécurité.

HKLM\SECURITY

HKLM\SECURITY restitue de façon tangible bon nombre des aspects de Windows liés à la sécurité. Les informations stockées à cet égard incluent les paramètres de stratégie, les valeurs de sécurité par défaut, les secrets partagés par les serveurs, les informations de compte, telles que les informations d'identification mises en cache d'ouverture de session, et bien d'autres informations apparentées à la protection de l'ordinateur. Une copie de la base de données SAM est également enregistrée à cet endroit, même s'il est protégé.

Par défaut, les mécanismes de protection en oeuvre concernant HKLM\SECURITY se montrent particulièrement restrictifs, cela en considération de la nature critique des informations que ladite clé héberge, et sont en l'occurence configurés pour accorder l'accès uniquement au compte système local. Vous pouvez néanmoins modifier le descripteur de sécurité pour permettre l'accès en lecture aux administrateurs, ou éventuellement utiliser PsExec pour exécuter Regedit (ou n'importe quel autre utilitaire dévolu à la lecture du Registre) dans le contexte du compte système local.

HKLM\HARDWARE

HKLM\HARDWARE contient les descriptions des matériels de l'ordinateur et toutes les associations périphérique-à-pilote. Elle permet ce faisant à Windows et à ses pilotes de périphériques d'enregistrer et de partager les informations sur les équipements présents au démarrage du système, ou reconnus par la suite lors d'une procédure Plug and Play.

De nombreuses commandes d'administration permettant l'examen de données relatives au matériel, dont l'utilitaire Gestionnaire de périphériques et la boite de dialogue Système, prennent leur source à partir de valeurs appropriées dans la clé HKLM\HARDWARE.

HKLM\SOFTWARE

HKLM\SOFTWARE fait office de racine de stockage des paramètres globaux des composants et applications Windows, renfermant de la sorte autant des données de configuration de niveau système (mais qui ne sont pas requises pour l'amorçage) que des paramètres de niveau application.

Une large majorité des logiciels, tierce partie ou livré avec Windows, enregistre sous HKLM\SOFTWARE leurs paramètres, par exemple les chemins des fichiers et répertoires, les informations de licence et date d'expiration, etc. Les paramètres présents à cet endroit s'appliquent à tous les utilisateurs. D'autres paramètres spécifiques à l'utilisateur se trouveront dans les clés HKEY_USERS ou HKEY_CURRENT_USER.

HKLM\SYSTEM

HKLM\SYSTEM contient les options de configuration requises lors de l'amorçage du système, par exemple quels sont les pilotes à charger et les services à démarrer.

HKLM\SYSTEM\CurrentControlSet

Une fois terminée la procédure d'amorçage de l'ordinateur, Windows met à jour le Registre afin qu'il reflète les derniers composants et contrôles utilisés pour démarrer correctement. La clé CurrentControlSet fait pour l'occasion référence à l'ensemble de contrôles actuellement en vigueur sur la plateforme (sous entendu l'ensemble le plus récent qui ai permis de démarrer). Dans un contexte parallèle, la sous-clé HKLM\SYSTEM renferme un nombre variable de clés ControlSetXXX, qui chacune représente un ensemble de contrôles ayant existé, indépendamment du succès ou de l'échec de l'entreprise auquel il donnait lieu.

Visualisation des jeux de contrôle dans HKLM\SYSTEM.

```
C:\>reg query HKLM\SYSTEM /f ControlSet
```

HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001 HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet Fin de la recherche : 2 correspondance(s) trouvée(s).

C:\>reg query HKLM\SYSTEM\ControlSet001

HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control HKEY_LOCAL_MACHINE\SYSTEM \ControlSet001\Enum HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Hardware Profiles HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Policies HKEY_LOCAL_MACHINE\SYSTEM \ControlSet001\Services HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Software

CurrentControlSet renferme plusieurs sous-clés :

- **Control** Contient des informations de contrôle pour les services et les outils du système.
- **Enum** Contient une entrée pour chaque périphérique physique ou logique détectable par le système. Certaines de ces informations sont accessibles depuis le gestionnaire de périphériques de Windows.
- **Harware Profiles** Contient une entrée par profil matériel défini sur la machine. De la même façon que le système de numérotation de l'arborescence HKLM\SYSTEM, chaque profil possède un numéro, le premier étant 0001. HKLM\SYSTEM\Harware Profiles\Current pointe toujours vers le profil sélectionné lors de l'amorçage du système. Les informations relatives aux profils matériels peuvent être configurés depuis Panneau de configuration / onglet Système / Matériel / Profils Matériels.
- **Services** Contient une sous-clé par service installé. Ces sous-clés incluent les informations de configuration nécessaires aux services Windows.

HKEY_PERFORMANCE_DATA

HKEY_PERFORMANCE_DATA est utilisée de sorte à accéder aux valeurs des compteurs de performance sous Windows, ce qui donne par ailleurs une dimension toute nouvelle au relevé de ces informations. Déjà spéciale par nature, HKEY_PERFORMANCE_DATA l'est encore plus du fait de n'être accessible que du point de vue de la programmation, c'est à dire par l'intermédiaire des fonctions de l'API Windows (Kernel32.dll) ou de l'API Performance Data Helper (Pdh.dll).

Les informations que véhicule HKEY_PERFORMANCE_DATA ne sont pas en réalité stockés dans le registre - et n'ont de toutes les façons pas vocation à l'être. Il faut en l'occurence imaginer une telle clé non comme un intermédiaire direct vers les valeurs des compteurs de performances, mais plutôt comme une rampe d'accès vers divers fournisseurs de données de performance.

HKEY_CURRENT_CONFIG

HKEY_CURRENT_CONFIG est un lien vers le profil matériel courant, stocké sous HKLM\SYSTEM \CurrentControlSet\Hardware Profiles\Current. Les profils matériels permettent de décrire quelles pièces de matériel sont activés ou désactivés quand le système est démarré sous un profil donné. Ils donnent ainsi la possibilité à l'administrateur d'adapter les paramètres des pilotes du système aux variations de l'environnement matériel de l'ordinateur. Par exemple, sur un ordinateur portable avec un plug and play compatible station d'accueil, deux profils pourraient être créés et chargés automatiquement selon les besoins : un pour quand l'ordinateur est amarré à la station, un autre pour quand il ne l'est pas. Si plusieurs profils matériels sont connus du système, ce dernier affiche au démarrage un menu spécifique, attendant le choix d'une configuration appropriée à charger.

HKEY_CLASSES_ROOT

HKEY_CLASSES_ROOT renferme principalement trois sortes d'information : des associations d'extension de fichier, la vue virtualisée du Registre, et des détails de configuration concernant les objets COM.

Il existe sous HKEY_CLASSES_ROOT une clé pour toute extension de nom de fichier connue. La plupart des clés contiennent une valeur REG_SZ qui pointe vers une autre clé de HKCR incluant les données d'association concernant la classe de fichiers que cette extension représente. Ainsi, par exemple, HKCR\.doc pointe vers des informations sur les fichiers Microsoft Word stockées dans Word.Document.8. S'ajoute à cela, les propriétés, la commande associée à un double clic et les commandes disponibles dans le menu contextuel du dit document.

Les données sous HKEY_CLASSES_ROOT proviennent de deux sources (la branche en elle-même est en quelque sorte le produit de ce couple) :

- Les données de déclaration de classe de niveau utilisateur sous HKCU\SOFTWARE\Classes (correspondant au fichier \Users\<username\>\AppData\Local\Microsoft\Windows\Usrclass.dat)
- Les données de déclaration de classe de niveau système sous HKLM\SOFTWARE\Classes.

La raison pour laquelle il existe une séparation entre données de déclaration de classe de niveau utilisateur et données de déclaration de classe de niveau système vient de ce qu'on veut que les profils itinérants puissent contenir ces personnalisations. En outre, une telle manière de procéder referme d'emblée une potentielle brèche de sécurité : un utilisateur non privilégié ne peut ni modifier ni supprimer des clés dans la version HKEY_CLASSES_ROOT de niveau système, et ne peut donc pas affecter le fonctionnement global des applications.

Visualisation et modification du Registre

S'il reste communément admis que la manipulation du registre ne devrait pas s'effectuer de manière directe (les paramètres des applications et du système susceptibles de modifications devraient légitimement avoir une interface utilisateur associée rendant visible les possibilités les concernant), certains paramètres du système n'ont aucun intermédiaire pour les restituer, raison pour laquelle Windows offre plusieurs outils permettant de visualiser et modifier le registre.

- Éditeur du Registre
- Reg.exe
- Windows PowerShell

Editeur du Registre

L'Éditeur du Registre est composé de deux parties : un volet de navigation à gauche et un volet d'informations à droite. Le volet de navigation contient la liste hiérarchique des ruches de Registre correspondant au Registre d'un ordinateur cible (une riche est un groupe de clés, de sous-clés et de valeurs dans le Registre). Quand une ruche de Registre est sélectionnée dans le volet de navigation, toutes les entrées du Registre ajoutées pour cette ruche sont affichées dans le volet de droite.

Manipuler le registre via Windows PowerShell

Conçu spécialement pour l'administration du système, dont une des attributions courantes consiste à surveiller voire modifier le Registre, Windows PowerShell intègre en la matière diverses commandes et fonctionnalités. En utilisant le fournisseur Registry de Windows PowerShell, vous pouvez faire des recherches entre les informations du Registre, créer de nouvelles clés, supprimer des clés existantes et modifier des listes de valeurs et de contrôles d'accès (ACL) depuis Windows PowerShell.

Afin de permettre l'accès aux informations du Registre, externes à l'environnement du Shell, le fournisseur Registry maintient un espace de noms hiérarchique organisé à partir des clés et sous-clés de la base de données du Registre. Ces données, des éléments sur les lecteurs Windows PowerShell, sont rendues visibles comme s'il s'agissait d'une structure de fichiers de disque ; leur utilisation est en conséquence très semblable à celle des

fichiers et dossiers. Les valeurs et données, considérées dans Powershell comme des propriétés des éléments (et non des éléments distincts), ne sont pas des composants de cette hiérarchie.

Le fournisseur Registry prend en charge pour l'interaction avec les clés de Registre tous les applets de commande Item (**- Item), telles que Get-Item, Copy-Item et Rename-Item. Utilisez les applets de commande Item lors de l'utilisation de clés et sous-clés de Registre. Par exemple, Vous pouvez afficher tous les éléments situés directement dans une clé de Registre en utilisant Get-ChildItem. Le fournisseur Registry prend également en charge les applets de commande ItemProperty (**- ItemProperty). Utilisez les applets de commande ItemProperty lors de l'utilisation de valeurs et données de Registre. Par exemple, L'applet de commande Remove-ItemProperty supprime une propriété et sa valeur d'un élément. Vous pouvez l'utiliser pour supprimer des valeurs de Registre et les données qu'elles contiennent.

Fichiers d'inscription

Établis comme un moyen commode de gérer le registre autrement que par l'intermédiaire d'utilitaires dédiés, les fichiers d'inscription (extension .reg) sont des fichiers à base de texte (donc facilement lisibles par l'homme) qui permettent l'importation et l'exportation de constituants (clés, sous-clés et valeurs) depuis et vers la base de registre. Lorsque vous exécutez un tel fichier, son contenu est fusionné dans le Registre local. Les administrateurs peuvent du même coup utiliser ces fichiers pour distribuer à distance des modifications du Registre sur plusieurs ordinateurs Windows.

Vous pouvez envoyer un fichier d'inscription à des utilisateurs dans un message électronique, placer un fichier .reg sur un partage réseau et demander aux utilisateurs du partage réseau de l'exécuter, ou encore ajouter une commande aux scripts d'ouverture de session des utilisateurs pour importer automatiquement le fichier .reg lorsqu'ils se connectent.

Lorsque les utilisateurs exécutent le fichier .reg, ils reçoivent les messages suivants : "Éditeur du Registre / Voulez-vous vraiment ajouter les informations contenues dans chemin_fichier_.reg au Registre ?". Si l'utilisateur clique sur Oui, il reçoit le message suivant : "Éditeur du Registre / Les informations de chemin_fichier_.reg ont été inscrites dans le Registre.". Regedit.exe prend en charge un commutateur de ligne de commande /s pour ne pas afficher ces messages. Par exemple, pour exécuter silencieusement le fichier .reg (avec le commutateur /s) à partir d'un fichier de commandes de script de connexion, utilisez la syntaxe chemin_fichier_.reg.

Utilitaire Reg (Reg.exe)

L'utilitaire Reg (Reg.exe) permet de visualiser, de modifier ou d'ajouter des entrées dans le Registre à partir de l'invite de commandes. Aussi performant que n'importe lequel de ses homologues en mode graphique, Reg est essentiellement destinée aux administrateurs système, lesquels l'utilisent surtout dans des fichiers de commandes appliquant de manière successive et répétitive des modifications au Registre. L'emploi de la commande Reg reste néanmoins un choix stratégique dans la plupart des scénarios et pour de nombreux utilisateurs.

Comme tant autres utilitaires sollicités depuis l'invite de commandes, l'utilitaire Reg possède sa propre série de sous-commandes. Les options disponibles sont nombreuses et se passent en tant que paramètres à l'exécutable. Voici la syntaxe générale de la commande Reg est la forme `reg action [options]`, où *reg* correspond au nom de l'exécutable Reg, *action* à la commande de registre à effectuer et *options* à la liste des paramètres spécifiques à la commande. Les principales commandes reconnues par Reg sont listées dans le tableau suivant.

Tableau 5.5. Commandes Reg

Commande	Action
add	Ajoute une nouvelle sous-clé ou entrée au Registre
compare	Compare les valeurs de deux sous-clés ou de deux entrées
copy	Copie les informations du Registre à un nouvel emplacement d'un Registre local ou réseau
delete	Supprime des informations du Registre

Commande	Action
export	Crée une copie des sous-clés ou valeurs spécifiées dans un fichier. Ce fichier permet de transférer les données du Registre vers d'autres ordinateurs à l'aide de la commande Import.
import	Importe un fichier registre et crée et modifie les clés du fichier dans le Registre local.
load	Permet de déplacer les données du registre à un nouvel emplacement.
query	Affiche toutes les informations du Registre sous la clé spécifiée.
restore	Restaure les informations du Registre à partir d'un fichier crée avec la commande save.
save	Enregistre les informations du Registre dans un fichier. Cela est utile pour effectuer une sauvegarde avant de modifier une clé.
Unload	Supprime les données du Registre ajoutées avec la commande Load.

Plusieurs remarques additionnelles concernant la syntaxe de Reg.exe :

- Reg.exe exige que les noms des clés des racines figurent sous forme abrégée. Saisissez HKLM, par exemple, et non HKEY_LOCAL_MACHINE
- Dans l'éventualité où un nom de clé ou de valeur comporte des espaces, il est impératif de le faire apparaitre entre guillemets.
- Si vous introduisez une commande Reg.exe dans un programme batch et qu'elle renferme une variable d'environnement, il est impératif de placer le nom de la variable entre deux paires de symboles de pourcentage (et non un seul). Autrement, l'interprète de commandes développe le nom de la variable avant de la passer à Reg.exe.

Registre et sécurité

Compte tenu les fonctions essentielles qu'assume le Registre vis-à-vis de la configuration - par extension, l'état et le fonctionnement - du système, il est possible d'attribuer des autorisations (permissions) affectant tout ou partie de ladite base. Ainsi, par exemple, la ruche HKLM\SAM, est-elle protégée originellement contre toute tentative de lecture, même depuis un compte disposant de niveaux de privilèges élevés, et pour cause : cet ensemble centralise rien moins que les paramètres d'identification des utilisateurs, y compris leurs mots de passe.

D'autres segments ou portions spécifiques du Registre se voient également protégés. Il est important de souligner que cette sécurisation, implémentée à l'aide de listes de contrôle d'accès, a comme pré-requis essentiel l'emploi sur le support hébergeant le système d'exploitation du système de fichiers NTFS.

Tableau 5.6. Autorisations du Registre

Autorisation	Implication de cette autorisation
Contrôle total	L'utilisateur peut ouvrir, modifier et prendre possession d'une clé.
Requête sur une valeur	L'utilisateur peut interroger la valeur que renferme une clé.
Définir la valeur	L'utilisateur peut créer de nouvelles valeurs sous une clé ou remplacer une valeur existante.
Créer une sous-clé	L'utilisateur peur créer une nouvelle sous-clé sous une clé
Énumérer les sous-clés	L'utilisateur a accès à la liste de toutes les sous-clés d'une clé particulière (dans la perspective NTFS, cette possibilité revient à traverser un répertoire).
Avertir	L'utilisateur peut enregistrer une fonction qui se déclenche lorsque la valeur sélectionnée a fait l'objet de modifications.
Créer une liaison	L'utilisateur peut créer une liaison vers une clé.
Supprimer	L'utilisateur peut créer une clé ou une valeur.

Accès en écriture à la liste de contrôle d'accès	L'utilisateur dispose d'un accès en écriture aux contrôles d'accès de la clé sélectionnée.
Accès du propriétaire	L'utilisateur peut prendre possession de la clé sélectionnée.
Contrôle en lecture	L'utilisateur peut lire la liste de contrôle d'accès discrétionnaire (DACL) de la clé.

Ainsi que précédemment suggéré, le profil de sécurité attribué à certains composants sensibles de la base de registre empêche l'accès par un compte autre que le compte système local. Un moyen rudimentaire d'accéder à ces données consiste simplement à réinitialiser leur sécurité, ce qui en plus de n'être pas conforme aux bonnes pratiques en matière d'administration, pourrait avoir des résultats malencontreux sur la sécurité d'ensemble du système. Une autre façon consiste à exécuter l'utilitaire Regedit.exe, ou tout autre outil offrant la possibilité d'éditer le registre, sous le compte système local. L'utilitaire PsExec prend en charge une option permettant de procéder ainsi.

Callbacks de registre

Windows donne aux composants de l'exécutif, pilotes et autres codes système l'occasion de surveiller les activités se déroulant dans le registre, ce par l'intermédiaire de routines de rappel (callbacks) spéciales. Un thread mode noyau emploie la fonction CmRegisterCallbackEx en vue de déclarer auprès du gestionnaire de configuration un point d'entrée RegistryCallback, que Windows exécute chaque fois qu'une opération liée au registre a lieu (comprendre chaque fois qu'un service système apparenté au registre est sollicité). Ce genre de fonctionnalités s'illustre, par exemple, dans divers produits de sécurité, qui examinent les données du registre afin de détecter d'éventuelles malveillances.

Registre et droits d'accès

Tableau 5.7. Droits d'accès concernant clés et valeurs de registre

Constante	Valeur	Description
KEY_QUERY_VALUE	0x0001	Requis pour interroger les valeurs présentes dans une clé. Voir service NtEnumerateValueKey, NtEnumerateKey (KeyInformationClass = KeyNameInformation), NtQueryValueKey.
KEY_ENUMERATE_SUB_KEYS	0x0008	Requis pour énumérer des sous-clés.
KEY_NOTIFY	0x0010	Requis pour effectuer un audit (recevoir des notifications de modification) d'un clé.
KEY_SET_VALUE	0x0002	Requis pour créer, supprimer ou modifier une valeur. Voir service NtDeleteValueKey.
KEY_CREATE_SUB_KEY	0x0004	Requis pour créer une sous-clé.
KEY_CREATE_LINK	0x0020	Requis pour créer un lien symbolique.

Tableau 5.8. Mappage générique des droits d'accès au registre

Droit générique	Signification
KEY_ALL_ACCESS	STANDARD_RIGHTS_ALL
	KEY_QUERY_VALUE
	KEY_SET_VALUE
	KEY_CREATE_SUB_KEY
	KEY_ENUMERATE_SUB_KEYS
	KEY_NOTIFY
	KEY_CREATE_LINK
KEY_EXECUTE	KEY_READ
KEY_READ	STANDARD_RIGHTS_READ
	KEY_QUERY_VALUE
	KEY_ENUMERATE_SUB_KEYS

Droit générique	Signification
	KEY_NOTIFY.
KEY_WRITE	STANDARD_RIGHTS_WRITE
	KEY_SET_VALUE
	KEY_CREATE_SUB_KEY

NtQueryKey

L'acquisition de métadonnées associées à une clé spécifique, telles que son nom, sa classe ou son horodatage, peut être conduite à l'aide de la fonction `NtQueryKey`, laquelle se présente comme suit.

```
NTSTATUS NtQueryKey(
    _In_ HANDLE KeyHandle,
    _In_ KEY_INFORMATION_CLASS KeyInformationClass,
    _Out_writes_bytes_opt_(Length) PVOID KeyInformation,
    _In_ ULONG Length,
    _Out_ PULONG ResultLength
);
```

Tableau 5.9. KEY_INFORMATION_CLASS

ID	Classe	Type de données associé
0x0000	KeyBasicInformation	KEY_BASIC_INFORMATION
0x0001	KeyNodeInformation	KEY_NODE_INFORMATION
0x0002	KeyFullInformation	KEY_FULL_INFORMATION
0x0003	KeyNameInformation	KEY_NAME_INFORMATION
0x0004	KeyCachedInformation	KEY_CACHED_INFORMATION
0x0005	KeyFlagsInformation	KEY_FLAGS_INFORMATION
0x0006	KeyVirtualizationInformation	KEY_VIRTUALIZATION_INFORMATION
0x0007	KeyHandleTagsInformation	KEY_HANDLE_TAGS_INFORMATION
0x0008	KeyTrustInformation	KEY_TRUST_INFORMATION
0x0009	KeyLayerInformation	KEY_LAYER_INFORMATION

```
typedef struct _KEY_BASIC_INFORMATION {
  LARGE_INTEGER LastWriteTime;
  ULONG         TitleIndex;
  ULONG         NameLength;
  WCHAR         Name[1];
} KEY_BASIC_INFORMATION, *PKEY_BASIC_INFORMATION;

typedef struct _KEY_CACHED_INFORMATION {
  LARGE_INTEGER LastWriteTime;
  ULONG         TitleIndex;
  ULONG         SubKeys;
  ULONG         MaxNameLen;
  ULONG         Values;
  ULONG         MaxValueNameLen;
  ULONG         MaxValueDataLen;
  ULONG         NameLength;
} KEY_CACHED_INFORMATION, *PKEY_CACHED_INFORMATION;

typedef struct _KEY_FULL_INFORMATION {
    LARGE_INTEGER LastWriteTime;
    ULONG   TitleIndex;
    ULONG   ClassOffset;
    ULONG   ClassLength;
    ULONG   SubKeys;
    ULONG   MaxNameLen;
    ULONG   MaxClassLen;
    ULONG   Values;
```

```
    ULONG   MaxValueNameLen;
    ULONG   MaxValueDataLen;
    WCHAR   Class[1];
} KEY_FULL_INFORMATION, *PKEY_FULL_INFORMATION;

typedef struct _KEY_NAME_INFORMATION {
  ULONG NameLength;
  WCHAR Name[1];
} KEY_NAME_INFORMATION, *PKEY_NAME_INFORMATION;

typedef struct _KEY_NODE_INFORMATION {
  LARGE_INTEGER LastWriteTime;
  ULONG         TitleIndex;
  ULONG         ClassOffset;
  ULONG         ClassLength;
  ULONG         NameLength;
  WCHAR         Name[1];
} KEY_NODE_INFORMATION, *PKEY_NODE_INFORMATION;

typedef struct _KEY_VALUE_BASIC_INFORMATION {
  ULONG TitleIndex;
  ULONG Type;
  ULONG NameLength;
  WCHAR Name[1];
} KEY_VALUE_BASIC_INFORMATION, *PKEY_VALUE_BASIC_INFORMATION;

typedef struct _KEY_VIRTUALIZATION_INFORMATION {
  ULONG VirtualizationCandidate : 1;
  ULONG VirtualizationEnabled : 1;
  ULONG VirtualTarget : 1;
  ULONG VirtualStore : 1;
  ULONG VirtualSource : 1;
  ULONG Reserved : 27;
} KEY_VIRTUALIZATION_INFORMATION, *PKEY_VIRTUALIZATION_INFORMATION;
```

Le champ NameLength précise la taille, en octets, de la chaine de caractères véhiculée par le tableau Name, représentant le nom de clé dans le registre. Le champ Name constitue un tableau de caractères larges (dont la taille est indiquée par l'attribut NameLength) qui renferme le nom de la clé.

```
/*
** keyinfo.c
*/

#include "keyinfo.h"

#pragma comment(lib, "ntdll")

int     wmain(int argc, wchar_t **argv)
{
  NTSTATUS Status;
  HANDLE hKey;
  UNICODE_STRING strkeyName;
  OBJECT_ATTRIBUTES keyAttr;
  BYTE buffer[1024];
  ULONG len;
  PKEY_NAME_INFORMATION pKeyInfo;
  PKEY_FULL_INFORMATION pKeyFullInfo;

  RtlInitUnicodeString(&strkeyName, argv[1]);
  InitializeObjectAttributes(&keyAttr, &strkeyName, 0, NULL, NULL);

  Status = NtOpenKey(&hKey, KEY_QUERY_VALUE, &keyAttr);
  if (!NT_SUCCESS(Status)) {
    printf("ERROR : NtOpenKey (0x%X)\n", Status);
    return (-1);
```

```
  }

  Status = NtQueryKey(hKey, KeyNameInformation, buffer, sizeof(buffer), &len);
  if (!NT_SUCCESS(Status)) {
    printf("ERROR : NtQueryKey (0x%X)\n", Status);
    return (-1);
  }
  pKeyInfo = (KEY_NAME_INFORMATION *) buffer;
  wprintf(L"Name: %.*s\n",
          pKeyInfo->NameLength / sizeof(WCHAR),
          pKeyInfo->Name);

  Status = NtQueryKey(hKey, KeyFullInformation, buffer, sizeof(buffer), &len);
  if (!NT_SUCCESS(Status)) {
    printf("ERROR : NtQueryKey (0x%X)\n", Status);
    return (-1);
  }
  pKeyFullInfo = (KEY_FULL_INFORMATION *) buffer;
  printf("Subkeys: %u\n", pKeyFullInfo->SubKeys);
  printf("Values: %u\n", pKeyFullInfo->Values);
  printf("Max class length: %u\n", pKeyFullInfo->MaxClassLen);
  printf("Max name length: %u\n", pKeyFullInfo->MaxNameLen);
  printf("Max value name length: %u\n", pKeyFullInfo->MaxValueNameLen);
  printf("Max data length: %u\n", pKeyFullInfo->MaxValueDataLen);

  return (0);
}

c:\>keyinfo "\registry\machine\system\currentcontrolset\services"

Name: \REGISTRY\Machine\System\ControlSet001\Services
Subkeys: 26
Values: 0
Max class length: 0
Max name length: 32
Max value name length: 0
Max data length: 0
```

NtSetInformationKey

Tableau 5.10. KEY_SET_INFORMATION_CLASS

ID	Classe	Type de données associé
0x0000	KeyWriteTimeInformation	KEY_WRITE_TIME_INFORMATION
0x0001	KeyWow64FlagsInformation	KEY_WOW64_FLAGS_INFORMATION
0x0002	KeyControlFlagsInformation	KEY_CONTROL_FLAGS_INFORMATION
0x0003	KeySetVirtualizationInformation	KEY_SET_VIRTUALIZATION_INFORMATION
0x0004	KeySetDebugInformation	KEY_CONTROL_FLAGS_INFORMATION
0x0005	KeySetHandleTagsInformation	KEY_HANDLE_TAGS_INFORMATION
0x0006	KeySetLayerInformation	KEY_SET_LAYER_INFORMATION

Ruches

Les données s'apparentant au Registre Windows sont le disque enregistrées sous la forme de plusieurs fichiers appelées *ruches*. Chaque ruche fait miroir à une arborescence de registre, avec une clé servant de racine (autrement dit de point de départ) à cette arborescence.

Lorsque le Gestionnaire de configuration manipule (charge ou décharge) des ruches, il le fait à l'aide des informations contenues sous la clé HKLM\SYSTEM\CurrentControlSet\Control\hivelist, dont les valeurs internes définissent les noms de chemins de toutes les ruches.

Quelques-uns des sous-ensembles définis dans le Registre sont volatiles, et ne sont en conséquence pas associés à des fichiers. Le système crée et gère ces ruches en mémoire uniquement ; elles sont donc temporaires et ne perdurent pas au-delà de l'extinction de l'ordinateur. Windows recrée les ruches volatiles à chaque démarrage. Exemple de ruche volatile : HKLM\HARDWARE, qui stocke des informations sur les périphériques matériels et sur les ressources qui leur sont assignées. La détection des périphériques et l'assignation des ressources ayant lieu à chaque réinitialisation du système, il est alors tout à fait logique de ne pas pérenniser les données issues de ces procédures et, partant, de ne pas les enregistrer sur disque.

Tableau 5.11. Correspondance entre fichiers disque et chemins de registre

Chemin registre	Chemin fichier
HKEY_LOCAL_MACHINE\BCDOOOOOOOO	\Boot\BCD
HKEY_LOCAL_MACHINE\COMPONENTS	%SystemRoot%\System32\Config\Components
HKEY_LOCAL_MACHINE\HARDWARE	Ruche volatile
HKEY_LOCAL_MACHINE\SAM	%SystemRoot%\System32\Config\Sam
HKEY_LOCAL_MACHINE\SECURITY	%SystemRoot%\System32\Config\Security
HKEY_LOCAL_MACHINE\SOFTWARE	%SystemRoot%\System32\Config\Software
HKEY_LOCAL_MACHINE\SYSTEM	%SystemRoot%\System32\Config\System
HKEY_USERS\.DEFAULT	%SystemRoot%\System32\Config\Default

Tout comme le système de gestion de fichiers, qui s'appuie pour afficher un espace de noms virtuels sur un certain nombres d'artifices, le gestionnaire de configuration procède sensiblement de la même manière en utilisant des liaisons symboliques (type de données REG_LINK) pour relier les ruches, donnant ainsi au Registre la forme structurelle qu'on lui connaît. Les branches HKEY_CLASSES_ROOT et HKEY_CURRENT_USER, par exemple, correspondent en réalité à des emplacements de HKLM et HKU.

Services

Presque tous les systèmes d'exploitation disposent d'un mécanisme pensé pour des tâches non attachées explicitement à l'utilisateur interactif. Sous Windows, ce genre de processus porte le nom de service ou service Windows. Analogues des "démons" UNIX, qui fonctionnent en arrière-plan plutôt que sous le contrôle direct d'un utilisateur, les services Windows sont des programmes qui se chargent automatiquement dans le cadre du processus de démarrage d'une application ou du processus de démarrage de système pour prendre en charge les différentes tâches nécessaires au système d'exploitation.

Les fonctionnalités implémentées au niveau service, souvent orientés réseau et dont la durée d'exécution est généralement longue, rendent les services parfaitement adaptés à une utilisation sur un serveur. Ils sont aussi particulièrement utiles lorsqu'une fonctionnalité du système doit n'être pas tributaire d'une identité numérique directe (utilisateurs comme administrateurs). Les exemples de composants Windows existants sous forme de services incluent le spouleur d'impression, le journal des événements et le planificateur de tâches.

Les services Windows sont constitués de trois composants : une application service, un programme de contrôle de service (SCP), et le gestionnaire de contrôle de service (SCM). Dans cette section, nous commencerons par d'écrire les applications et le fonctionnement du SCM. Nous expliquerons ensuite comment se fait le démarrage des services automatiques lors de l'amorçage du système. Nous verrons aussi les mesures que prend le SMC lorsqu'un service échoue lors de son démarrage et la façon dont le SMC arrête les services.

Le tableau qui suit énumère les fonctions Windows utilisées dans le contexte des services.

Tableau 5.12. Fonctions de gestion de services

Fonction	Description
ChangeServiceConfig/2	Modifie la configuration d'un service
ControlService/Ex	Envoie un code de contrôle à un service
CreateService	Crée un service (l'ajoute au Registre)

Fonction	Description
DeleteService	Supprime un service (du Registre)
EnumDependentServices	Énumère les dépendances de service
EnumServicesStatusEx	Énumère les services sur la base d'un niveau d'information
ControlService/Ex	Envoie une requête de contrôle à un service
GetServiceDisplayName	Obtient le nom d'affichage d'un service
GetServiceKeyName	Obtient le nom de la clé de service d'un service
OpenSCManager	Établit une connexion au gestionnaire de contrôle de service sur l'ordinateur spécifié
QueryServiceConfig	Retourne les paramètres de configuration d'un service
SetServiceObjectSecurity	Modifie la sécurité d'un service
StartService	Démarre un service

Applications service

Une application service est un ensemble logiciel dont au moins un composant fonctionne en tant que service Windows.

Quand vous installez une application qui inclut un service, le programme d'installation de cette application a pour obligation préalable de déclarer ledit service auprès du système, chose qu'il accomplit en sollicitant CreateService, laquelle fonction a dans ce contexte pour rôle de partager au SCM l'ensemble des informations nécessaires (pas moins de 13 paramètres) à une future prise en charge. Comme tant d'autres des fonctions de gestion de services, CreateService est implémentée dans Advapi32.dll, DLL qui recueille en réalité toutes les API coté client du SCM.

Le fonctionnement des applications service diffère de celui des autres logiciels à plusieurs égards :

- Elles sont dotés de code spécial leur permettant de recevoir des commandes du SCM et de lui communiquer leur état.
- Elles s'exécutent dans une station fenêtre différente de la station interactive, raison pour laquelle les boites de dialogue émanant d'une application service ne sont par conséquent pas visibles. De la même façon, les messages d'erreur doivent être consignées dans le journal des événements Windows, plutôt que montrées à l'aide d'une interface utilisateur.
- Tout utilisateur qui interagit avec un service le fait par non pas directement mais par le biais d'un programme de contrôle de service (SCP). Windows fournit des SCP intégrés dotés de fonctionnalités génériques de démarrage, d'arrêt, de suspension et de reprise. Certaines applications service incluent toutefois leurs propres SCP, cela essentiellement en vue de pouvoir instaurer des dispositions spécifiques (des paramètres de configuration particuliers) au service concerné.

Base de données des services

Le gestionnaire de contrôle de service déploie dans le Registre une représentation non volatile de quelques-unes de ses données internes, dont certaines contenant les propriétés définitoires de l'ensemble des services installés. Quand un programme d'installation déclare un service en appelant CreateService, il y a envoi d'un message au SCM de la machine appelée à héberger le dit service (l'ordinateur local dans une large majorité des cas). Le SCM crée alors une clé de registre pour le service sous HKLM\SYSTEM\CurrentControlSet\Services.

HKLM\SYSTEM\CurrentControlSet\Services.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\.NET CLR Data
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\.NET CLR Networking
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\.NET CLR Networking 4.0.0.0
```

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\.NET Data Provider for Oracle
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\.NET Data Provider for SqlServer
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\.NET Memory Cache 4.0
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\.NETFramework
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\1394ohci
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\3ware
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AarSvc
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AarSvc_d5d0c
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\ACPI
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AcpiDev
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\acpiex
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\acpipagr
.
.
.
.
```

Le SCM mémorise chaque caractéristique propre à un service sous forme de valeur dans la clé de registre associée au service. L'exemple suivant montre par exemple les options de configuration apparentées à Windows Defender (à quoi correspond le service nommée WinDefend).

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\WinDefend
DependOnService    REG_MULTI_SZ    RpcSs
Description    REG_SZ    @%ProgramFiles%\Windows Defender\MpAsDesc.dll,-240
DisplayName    REG_SZ    @%ProgramFiles%\Windows Defender\MpAsDesc.dll,-310
ErrorControl    REG_DWORD    0x1
FailureActions    REG_BINARY    80510100000000000100000003000000140000000300000064000000000000006400000000000000640000000
ImagePath    REG_EXPAND_SZ    "C:\ProgramData\Microsoft\Windows Defender\platform\4.18.2008.9-0\MsMpEng.exe"
LaunchProtected    REG_DWORD    0x3
ObjectName    REG_SZ    LocalSystem
RequiredPrivileges    REG_MULTI_SZ    SeImpersonatePrivilege\0SeBackupPrivilege\0SeRestorePrivilege\0SeDebugPrivilege
\0SeChangeNotifyPrivilege\0SeLoadDriverPrivilege\0SeSecurityPrivilege\0SeShutdownPrivilege\0SeIncreaseQuotaPrivilege
\0SeAssignPrimaryTokenPrivilege\0SeTcbPrivilege\0SeIncreaseBasePriorityPrivilege\0SeSystemEnvironmentPrivilege
\0SeTakeOwnershipPrivilege
ServiceSidType    REG_DWORD    0x1
Start    REG_DWORD    0x2
Type    REG_DWORD    0x10
FailureCommand    REG_SZ    C:\WINDOWS\system32\mrt.exe /EHB /ServiceFailure "CAMP=4.18.2008.9;approximate-\>
 Engine=1.1.17400.5;AVSIG=1.323.410.0;ASSIG=1.323.410.0" /StartService /Defender /q
```

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\WinDefend\Security

Les options admissibles en la matière sont celles que voici :

- **Type de démarrage (Start)** Indique si le démarrage de ce service est effectué automatiquement, défini de manière manuelle ou s'il est désactivé. Les services automatiques sont démarrés à l'amorçage. Les services manuels sont démarrés à la demande par les utilisateurs ou par d'autres services. Les services désactivés ne peuvent être démarrés tant qu'ils sont en l'état.

- **Niveau de contrôle d'erreur (ErrorControl)** Oriente la manière dont le système sanctionne avec plus ou moins de sévérité les ratés et dysfonctionnements émanant d'un service ou d'un pilote : ignorer le problème, afficher un avertissement, faire s'effondrer et réamorçer le système avec la dernière bonne configuration connue, etc. Pour plus d'informations, consultez la section Erreurs des services.

- **Type de composant (Type)** Type auquel se rapporte un composant. Comme les services et les pilotes de périphériques Windows stockent tous deux leurs paramètres de configuration sous la même clé de registre, il est de ce fait nécessaire de faire une distinction entre les deux. Voir tableau plus loin.

- **Nom de groupe (Group)** Désigne le groupe auquel appartient le pilote ou le service. Cela permet à des pilotes ou services apparentés de démarrer conjointement (par exemple, des pilotes de système de fichiers). L'entrée de registre List de la sous-clé HKLM\SYSTEM\CurrentControlSet\Control\ServiceGroupOrder spécifie l'ordre de démarrage du groupe.

- **Numéro de tag (Tag)** Désigne l'ordre de démarrage d'un pilote au sein d'un groupe de pilotes. Ce paramètre ne concerne pas les services.

- **Dépendance de service (DependOnService)** Enumère les services spécifiques qui doivent être chargés avant le chargement de ce service. Ce paramètre ne s'applique pas aux pilotes autres que ceux ayant le type de démarrage SERVICE_AUTO_START.

- **Dépendance de groupes (DependOnGroup)** Au moins un élément de ce groupe doit démarrer avant le chargement de ce service.

- **Chemin de fichier exécutable (ImagePath)** Chemin du fichier exécutable sous-jacent au service ou pilote. Dans l'éventualité où ImagePath n'est pas spécifié, le gestionnaire d'E/S recherche les pilotes dans \Windows \System32\Drivers et le SCM dans les différents emplacements du système de fichier auxquels conduit la variable PATH.

- **Compte de service (ObjectName)** Nom du compte sous lequel le service est exécuté. Si ObjectName n'est pas spécifié, le système emploie le compte système local (LocalSystem). Ce paramètre ne s'applique pas aux pilotes.

- **Nom de service (DisplayName)** Nom par lequel se fait connaitre le service auprès des différents logiciels, par exemple l'application Services. Dans l'éventualité où aucun nom n'a été donné, c'est le nom de la clé de registre renfermant les informations du service qui est utilisé.

- **Description de service (Description)** Contient une brève description textuelle du service et de son but (32767 octets maximum)

- **Actions en cas d'échec (FailureActions)** Description des actions que doit exercer le SCM dans l'éventualité où le processus service se termine de manière inattendue ; plusieurs cas de figure ici : le redémarrage du processus service, l'exécution d'un programme spécifié et le réamorçage du système. Ce paramètre ne concerne pas les pilotes.

- **Commande de processus (FailureCommand)** Le SCM ne s'appuie sur cette valeur que si FailureActions a enjoint le SCM à exécuter un programme spécifique en cas d'échec du service. Ce paramètre ne concerne pas les pilotes.

- **Descripteur de sécurité (Security)** Contient le descripteur de sécurité qui définit qui a accès à l'objet service créé en interne par le SCM.

Type de services

Chaque service a parmi ses propriétés définitoires un certain nombre d'attributs qui le caractérisent auprès du SCM, par exemple si le service est exécuté dans un seul processus ou au contraire partage un processus avec d'autres composants, s'il peut interagir avec le bureau actif, etc. En interne, toutes ces informations sont regroupées sous la valeur Type de la clé associée au service. Le tableau suivant décrit les valeurs susceptibles d'apparaitre à ce niveau.

Tableau 5.13. Valeurs de registre Type

Constante	Valeur	Description
SERVICE_ADAPTER	0x00000004	Spécifie un adaptateur. Cet attribut est désormais obsolète.
SERVICE_FILE_SYSTEM_DRIVER	0x00000002	Spécifie un pilote de système de fichiers mode noyau.
SERVICE_KERNEL_DRIVER	0x00000001	Spécifie un pilote de périphérique.
SERVICE_RECOGNIZER_DRIVER	0x00000008	Spécifie un pilote reconnaisseur de système de fichiers.

Constante	Valeur	Description
SERVICE_WIN32_OWN_PROCESS	0x00000010	Spécifie un service qui s'exécute dans un processus qui héberge un seul service
SERVICE_WIN32_SHARE_PROCESS	0x00000020	Spécifie un service qui s'exécute dans un processus qui héberge plusieurs services.
SERVICE_INTERACTIVE_PROCESS	0x00000100	Spécifie un service autorisé à afficher des fenêtres sur la console et à recevoir des saisies de la part de l'utilisateur.

Chaque service peut éventuellement avoir une description qui le caractérise auprès des autres composants du système, lui permettant de la sorte de mettre en avant son but ou les diverses tâches qu'il accomplit. Pour associer une description à un service, sollicitez ChangeServiceConfig2 en lui spécifiant la valeur SERVICE_CONFIG_DESCRIPTION. Pour voir la description qui accompagne tel ou tel service, employez la commande sc qdescription, suivie d'un nom de service.

États d'un service

Comme d'autres composants opérationnels dans Windows, dont par exemple processus et threads, les services voient leur dynamique marquée par une succession de phases, à quoi ramène en interne différents états.

Tableau 5.14. État des services

Constante	Valeur	Description
SERVICE_STOPPED	0x00000001	Le service est arrêté.
SERVICE_START_PENDING	0x00000002	Le service est en cours de démarrage.
SERVICE_STOP_PENDING	0x00000003	Le service est en cours d'arrêt.
SERVICE_RUNNING	0x00000004	Le service est démarré.
SERVICE_CONTINUE_PENDING	0x00000005	Le service est en cours de reprise après une pause.
SERVICE_PAUSE_PENDING	0x00000006	La mise en pause du service est en cours.
SERVICE_PAUSED	0x00000007	Le service est en pause.

Chaque service a individuellement la responsabilité de signaler les modifications de son état interne au SCM. Un service signale son état actuel par le biais de la fonction SetServiceStatus, spécifiant alors auprès de ladite fonction une structure SERVICE_STATUS dont l'attribut dwCurrentState est établi conformément aux circonstances. (Les programmes de contrôle de service et le système n'ont pour prendre connaissance de l'état et de l'évolution d'un service pas d'autre alternative que le SCM, il est donc important que ce dernier dispose des informations les plus pertinentes.)

Tous les services entament leur cycle de vie dans l'état SERVICE_STOPPED. Lorsque le SCM démarre le service, il modifie l'état de ce dernier en le faisant passer à SERVICE_START_PENDING et appelle la fonction principale du service (ServiceMain). Ce type de fonctions n'à normalement qu'un seul but : terminer l'initialisation du service et préparer le terrain pour répondre aux demandes de contrôle ultérieures. Une fois que le service est prêt, il le signale au moyen de la fonction SetServiceStatus et de la constante SERVICE_RUNNING. La transition de SERVICE_START_PENDING à SERVICE_RUNNING indique au SCM (entres autres outils) que le service a démarré avec succès.

Les changements d'état que connaît un service lors de son exécution résultent le plus souvent de demandes de contrôle spécifiques, dont SERVICE_CONTROL_STOP, SERVICE_CONTROL_PAUSE et SERVICE_CONTROL_CONTINUE.

Les informations qu'émet un service concernant son état déterminent la manière dont le SCM interagit avec lui. Par exemple, quand un service retourne SERVICE_STOP_PENDING, lequel code ramène à une procédure d'arrêt en cours, le SCM ne transmet alors plus au service les requêtes ultérieures. Dans l'ordre légitime des choses, l'état suivant SERVICE_STOP_PENDING devrait être SERVICE_STOPPED, signe que l'arrêt s'est déroulé sans incident et a effectivement eu lieu.

Démarrage des services

La façon dont gestionnaire de contrôle des services (Scm) orchestre le démarrage est en interne piloté par l'entrée de registre Start.

Tableau 5.15. Valeurs de registre Start

Valeur	Type de démarrage	Description
SERVICE_BOOT_START(0)	Démarrage	Spécifie un pilote chargé (mais pas démarré) par le programme d'amorçage (Winload). En l'absence d'erreur, le pilote est démarré pendant l'initialisation du noyau, avant le chargement de tout pilote non-démarrage.
SERVICE_SYSTEM_START (1)	Système	Spécifie un chargé et initialisé pendant l'initialisation du noyau.
SERVICE_AUTO_START (2)	Chargement automatique	Spécifie un pilote ou un service initialisé lors du démarrage système par le gestionnaire de session (Smss.exe) ou le contrôleur de services (Services.exe).
SERVICE_DEMAND_START (3)	Chargement à la demande	Spécifie un pilote ou un service que le SCM démarre uniquement à la demande.
SERVICE_DISABLED (4)	Désactivé	Spécifie un pilote ou un service désactivé.
5	Démarrage retardé	Spécifie un service démarrant peu après le démarrage de la session utilisateur.

Une activité qui engage l'une des plus importantes, sinon la plus importante, charge de responsabilité au sein de la fonction initiale du SCM (SvcCtrlMain) est d'orchestrer le lancement des services et des pilotes dont les paramètres de configuration (valeur Start sous la clé idoine) stipulent que le démarrage est automatique. En interne, SvcCtrlMain sollicite pour ce genre d'opération la fonction SCM ScAutoStartServices.

L'algorithme de ScAutoStartServices procède par phases, chaque phase correspondant à un groupe et les phases se déroulant selon une chronologie définie par la valeur de registre HKLM\SYSTEM\CurrentControlSet \Control\ServiceGroupOrder\List. La valeur List inclut les noms des groupes dans l'ordre où le SCM doit les démarrer.

Certains pilotes et services exigent que des conditions, connues sous le nom de dépendances, soient satisfaites, comme spécifiées par la présence des entrées DependOnGroup, pour ce qui concerne les dépendances relatives à des groupes, et DependOnService, pour ce qui est des dépendances de service, dans la clé associée au service. Dans l'éventualité où une dépendance de niveau groupe est constatée, il faut que le groupe dont dépend le service ait déjà été initialisé et qu'au moins un service parmi ce groupe ait eu l'occasion de démarrer. Si le service dépend d'un groupe qui démarre après le groupe auquel appartient le service dans l'ordre de démarrage des groupes, le SCM enregistre une erreur de type dépendance circulaire pour le service.

Le SCM prend en charge le démarrage d'un service Windows ou d'un pilote par le biais de la fonction ScStartService, laquelle se comporte différemment selon qu'elle a affaire à l'une ou à l'autre sorte. Quand ScStartService démarre un service, elle commence par déterminer le nom du fichier qui exécute le processus du service en examinant la valeur ImagePath de la clé de registre associée. Elle consulte ensuite la valeur Type, toujours sous le même emplacement, et, si cela correspond à un service destiné à intégrer un processus qui en héberge plusieurs (SERVICE_WIN32_SHARE_PROCESS), vérifie alors que le processus dans lequel fonctionne le service, s'il a déjà démarré, est exécuté sous le même compte que celui spécifié pour le service à démarrer, comme indiquée par la valeur de registre ObjectName. Un service sans ObjectName est exécuté sous le compte système local.

Si le composant dont il entreprend le démarrage a une entrée de registre valant SERVICE_KERNEL_DRIVER ou SERVICE_FILE_SYSTEM_DRIVER, c'est que le SCM est en réalité confronté non à un service mais à un pilote ; auquel cas ScStartService appelle ScLoadDriver pour charger le pilote. ScLoadDriver active le privilège de sécurité SeLoadDriverPrivilege pour le processus SCM, puis passe le relais au service noyau NtLoadDriver.

Acceptation de l'amorçage

Outre lui confier la prise en charge des services, Windows place le SCM au centre de la gestion des configurations de démarrage, et tout particulièrement de la dernière bonne connue (last known good config). Particulièrement utile dans dans les cas où des modifications apportées au système l'empêchent de démarrer correctement, la dernière bonne configuration connue est en interne constituée par les paramètres de registre correspondant au dernier démarrage réussi de l'ordinateur. Par défaut, un tel succès se compose de deux critères : un démarrage réussi des services automatiques, et une ouverture de session utilisateur réussie. Il y a échec de démarrage si le système se bloque parce qu'un pilote de périphérique fait s'effondrer le système lors de l'amorçage ou si un service automatique ayant une valeur ErrorControl définie à SERVICE_ ERROR_SEVERE ou SERVICE_ERROR_CRITICAL fait état d'une erreur lors de son démarrage.

Les configurations de démarrage enregistrées sous HKLM\SYSTEM ont des noms de la forme ControlSet_nnn_, où *nnn* est un numéro du genre 001 ou 002. La clé HKLM\SYSTEM\Select contient des valeurs qui identifient le rôle de chaque configuration (voir figure ci-après). La Valeur Current sous Select se rapporte au jeu de contrôle actuellement utilisé. Par exemple, si CurrentControlSet pointe vers ControlSet001, la valeur vaut 1. La Valeur LastKnownGood sous Select contient le numéro de la dernière bonne configuration connue, laquelle fait en l'occurence tremplin vers le dernier jeu de contrôle ayant donné lieu à un démarrage réussi. La valeur Default renvoie au jeu de contrôle à utiliser par défaut si aucun autre n'est désigné. Autre valeur que la clé Select est susceptible de renfermer : Failed, qui pointe vers la dernière configuration pour laquelle le démarrage a échoué.

Clé de sélection de configuration de démarrage.

```
HKEY_LOCAL_MACHINE\System\Select
Current    REG_DWORD    0x1
Default    REG_DWORD    0x1
Failed    REG_DWORD    0x0
LastKnownGood    REG_DWORD    0x1
```

Comme nous l'avons déjà dit, que le SCM ai été en mesure de traiter correctement l'ensemble des services automatiques ne représente qu'un des facteurs requis pour tirer un bilan positif de la phase d'amorçage du système, dans la mesure où reste dans ce contexte à s'assurer du bon fonctionnement des mécanismes participant à l'ouverture de session utilisateur. Aussi, quand un utilisateur ouvre une session, Winlogon appelle la fonction NotifyBootConfigStatus, laquelle informe le SCM du succès de l'opération. Après démarrage réussi des services automatiques ou réception du message émanant de NotifyBootConfigStatus (de ces deux événements, c'est celui arrivé en dernier qui compte), le SCM appelle la fonction NtInitializeRegistry pour enregistrer la configuration courante en tant que dernière bonne connue.

SCM (*Service Control Manager*)

Le gestionnaire de contrôle de service (SCM, Service Control Manager) est un processus système spécial responsable du démarrage, de l'arrêt et de l'interaction la gestion des processus de service. Lancé dès l'amorçage du système et exécutant l'image \Windows\System32\Services.exe, le SCM est implémenté sous forme d'un serveur RPC, de sorte à pouvoir manipuler des services sur des machines distantes.

Parmi les responsabilités qui incombent au SCM, citons : (1) le stockage des paramètres et options de configuration concernant chaque service ; (2) le démarrage des services, soit lors de la phase d'amorçage ou à la demande ; (3) l'acheminement de requêtes de contrôle aux services appropriés.

Objectifs de conception des services Windows

Voici quelques-uns des objectifs et règles de fonctionnement auxquels les services Windows doivent se plier.

- **Démarrage immédiat** Les services lancés lorsque l'ordinateur est mis sous tension ou redémarré (services à démarrage automatique, qui sont pour la plupart essentiels à la prise en charge de l'environnement) ne devraient pas interférer de manière significative sur les performances du système.

- **Fermeture immédiate** Arrêter des services en cours d'exécution devrait n'avoir qu'une incidence mineure sur la capacité du système d'exploitation à s'éteindre rapidement.
- **Consommation des ressources** Chaque service devrait être optimisé dans le but de consommer le moins de mémoire et de ressources processeur. Dans la mesure du possible, chaque service devrait éviter d'être un fardeau supplémentaire sur les performances, la fiabilité, la stabilité et la robustesse du système.
- **Chemin de performance** Chaque service devrait viser un chemin de performance idéal, ce qui signifie être respectueux des conditions dans lesquelles le système d'exploitation et les autres services s'exécutent. Empêcher le système d'entrer dans un des états d'économie d'énergie, monopoliser une ressource, ou au contraire en abandonner une sans l'avoir libéré (un mutex) sont des exemples de situations préjudiciables pour tous les acteurs du système.
- **Exécution avec le compte à moindres privilèges** Chaque service ne devrait posséder que les privilèges et ne réclamer que les ressources nécessaires à son exécution. Sur le plan de la sécurité, les services ayant besoin d'être exécutés avec les données d'identification d'un compte utilisateur devraient choisir celui avec le moins de capacités, autrement dit le moins susceptible d'augmenter la surface d'attaque de l'ensemble du système.

Les règles mentionnés ci-dessus guide la démarche d'une bonne gestion des services. Les sections suivantes décrivent pour la plupart *comment* les services peuvent les respecter.

Services Windows et tâches planifiées

Pour l'essentiel, la différence entre services Windows et tâches planifiées tient au fait de quand les uns et les autres peuvent agir, et pour quelle longévité.

Les services Windows mettent en œuvre une fonctionnalité particulière du système d'exploitation, généralement pour le compte d'autres processus, et à un assez bas niveau. De telles applications n'interagissent pas avec l'utilisateur et ne nécessitent pas qu'un utilisateur ait ouvert une session pour s'exécuter.

Les tâches planifiées n'effectuent généralement pas non plus d'interaction utilisateur, mais nécessitent qu'un utilisateur ait ouvert une session. Les tâches planifiées sont déclenchées en fonction d'un calendrier ou par des conditions système, et sont gérées par le Planificateur de tâches. Les services ont un cycle de vie orchestré par le Gestionnaire de contrôle des services.

Comptes de services

Le contexte de sécurité d'un service, dans la mesure où c'est par son intermédiaire que sont définies les ressources auxquelles le processus peut accéder, tient lieu de sujet important tant pour les concepteurs de logiciels que pour les administrateurs. Dans la configuration par défaut, chaque service exécuté sous Windows peut l'être en fonction de l'une ou l'autre de trois identités : Système local, Service réseau et Service local. Les permissions de chacun de ces comptes sont ajustées précisément afin de leur octroyer uniquement les permissions nécessaires sur le système, ceci dans le but de limiter l'exposition du système d'exploitation en cas de faille de sécurité ou d'utilisation non autorisée.

Tableau 5.16. Privilège des comptes de service

Privilège	Système local	Service réseau	Service local
SeAssignPrimaryTokenPrivilege	{y}	{y}	{y}
SeAuditPrivilege	{y}	{y}	{y}
SeBackupPrivilege	{y}		
SeChangeNotifyPrivilege	{y}	{y}	{y}
SeCreateGlobalPrivilege	{y}	{y}	{y}
SeCreatePagefilePrivilege	{y}		
SeCreatePermanentPrivilege	{y}		
SeCreateTokenPrivilege	{y}		

Privilège	Système local	Service réseau	Service local
SeDebugPrivilege	{y}		
SeEnableDelegationPrivilege			
SeImpersonatePrivilege	{y}	{y}	{y}
SeIncreaseBasePriorityPrivilege	{y}		
SeIncreaseQuotaPrivilege	{y}	{y}	{y}
SeLoadDriverPrivilege	{y}		
SeLockMemoryPrivilege	{y}		
SeMachineAccountPrivilege			
SeManageVolumePrivilege	{y}		
SeProfileSingleProcessPrivilege	{y}		
SeRemoteShutdownPrivilege			
SeRestorePrivilege	{y}		
SeSecurityPrivilege	{y}		
SeShutdownPrivilege	{y}	{y}	{y}
SeSyncAgentPrivilege			
SeSystemEnvironmentPrivilege	{y}		
SeSystemProfilePrivilege			
SeSystemTimePrivilege	{y}	{y}	{y}
SeTakeOwnershipPrivilege	{y}		
SeTcbPrivilege	{y}		
SeUndockPrivilege	{y}	{y}	{y}

Compte système local

Plus haut gradé de la hiérarchie des comptes de services Windows, le compte système local représente auprès des autorités concernés l'identité, par extension le profil de sécurité, du système d'exploitation. Possédant de manière intrinsèque toutes les autorisations et tous les droits requis pour un accès complet à l'ordinateur, le compte système local est ainsi par nature extrêmement puissant, plus puissant que n'importe quel compte local ou domanial quand il s'agit de privilèges de sécurité locaux. Peu de surprise, donc, à ce que ce soit sur cette identité que s'appuie la plupart des services, y compris ceux représentant des composants clé : gestionnaire de sessions (Smss.exe), sous-système d'authentification de sécurité locale (Lsass.exe), processus d'ouverture de session (Winlogon.exe), sous-système Windows (Csrss.exe), et quelques autres encore.

Le compte Système Local peut être utilisé pour exécuter un service lorsque celui-ci requiert des permissions locales élevées, comme par exemple pour écrire dans les quelques sous-ensembles du système (système de fichiers ou base de registre) où seul ce compte a le droit de le faire.

Les caractéristiques principales du compte Système Local sont les suivantes :

- Il a pour véritable nom NT AUTHORITY\System.
- Il est membre du groupe Administrateurs locaux. Le tableau x indique les groupes auxquels appartient le compte système local.
- Il a le droit d'activer presque tous les privilèges (même ceux qui ne sont pas en principe accordés au compte administrateur local, par exemple la création de jetons de sécurité). Voir tableau x pour la liste des privilèges attribués au compte système local.
- Il possède plus de droits que le compte administrateur, pour lequel certaines parties du système d'exploitation demeurent dans tous les cas invisibles. Par exemple, alors que la ruche de sécurité HKEY_LOCAL_MACHINE \SECURITE est interdite à l'administrateur (puisque réservée à un rôle particulier), elle est autorisée pour le compte système.

- Les processus exécutés sous le compte système local le sont avec le profil utilisateur par défaut (HKU \.DEFAULT). Ils ne peuvent par conséquent pas accéder aux données de configuration stockées dans les profils utilisateur d'autres comptes.

Compte service réseau

Autre compte duquel une instance peut se réclamer, Service réseau est utilisé par des services ayant besoin d'accéder à des ressources sur le réseau sans pour autant nécessiter de privilèges particulièrement élevés.

Représentant également le système d'exploitation, Service Réseau possède cependant moins de permissions sur le système que Système Local. En premier lieu, son profil de sécurité englobe une étendue de moindre ampleur - en termes plus techniques, le jeton d'accès attribué à Service Réseau possède moins de privilèges que celui distribué pour son confrère Système Local. D'autre part, les listes de contrôles d'accès (ACL) de nombreux objets du système sont plus restrictives pour Service Réseau qu'elles ne le sont envers Système Local, cela avec comme conséquence de donner moins de latitude aux processus exécutés sous ce compte. Ainsi, ces processus ne peuvent pas par exemple, charger de pilotes dans le système d'E/S, ou ouvrir arbitrairement d'autres processus.

Le compte Service réseau bénéficie du même niveau d'accès aux ressources et aux objets que les membres du groupe Utilisateurs. Les services qui s'exécutent sous le compte Service réseau s'identifient auprès des machines présentes sur le réseau à l'aide des informations d'identification du compte d'ordinateur.

Les processus exécutés sous le compte service réseau utilisent le profil du compte service réseau, chargé dans le registre sous HKU\S-1-5-20, "AUTORITE NT\SERVICE RÉSEAU" en tant que service réseau. (Vous aurez compris que S-1-5-20 est une valeur sous forme de SID).

Le compte Service réseau dispose de privilèges minimaux sur l'ordinateur local. Cela empêche quiconque d'utiliser le compte pour accéder à des ressources protégées du système. Aucun mot de passe n'est associé à ce compte.

Compte service local

Le compte service local est quasiment identique au compte service réseau, à cette différence majeure près qu'il ne peut accéder qu'aux ressources réseau autorisant l'accès anonyme (session nulle).

Tout comme le compte service réseau, dont il se rapproche de bien des manières du point de vue de la sécurité, service local dispose de permissions réduites sur le système d'exploitation, équivalentes à celles d'un utilisateur standard non-administrateur. Ledit compte est par conséquent majoritairement employé parallèlement à des processus qui se satisfont facilement d'un cadre étroit en ce qui concerne les droits et les privilèges qui leur sont accordés, ou qui n'ont tout simplement pas besoin d'accéder au réseau.

Les processus exécutés sous le compte service local le sont par l'intermédiaire du profil se trouvant sous HKU\S-1-5-19, lequel est chargé au niveau du système de fichiers depuis %SystemRoot%\ServiceProfiles \LocalService.

Exemples de services que Windows exécute sous le compte service local : Registre à distance (service RemoteRegistry), qui permet d'accéder à distance à la base de registre du poste ; LmHosts, qui effectue la résolution de noms NetBIOS.

Exécution de service sous un compte suppléant

Compte tenu des restrictions précédemment mentionnées, certains services ont besoin de fonctionner sous un compte de connexion suppléant, compte dont les données d'identification viendront alors alimenter le contexte de sécurité dudit service. Vous pouvez configurer un service pour qu'il soit soit exécuté sous un compte de remplacement, soit lors de la création du service soit en spécifiant un compte et un mot de passe dans l'outil

d'administration Services. Pour ce faire, ouvrez la console Services (Services.msc), cliquez avec le bouton droit de la souris sur un service, sélectionnez Propriétés, allez dans l'onglet Connexion, puis cochez l'option Ce compte.

Démarrage automatique différé

Jusqu'à Windows Vista, la gestion des services ne reconnaissait que deux modes de démarrage : manuel et automatique. Du fait qu'un service à démarrage manuel exige l'intervention de l'utilisateur, l'alternative habituelle était le mode automatique, lorsque le service devait démarrer lors du démarrage du système, avant même l'ouverture d'une quelconque session utilisateur. Quel que soit le soin porté à leur optimisation, les services à démarrage automatique ont une incidence non négligeable sur les performances de démarrage du système. Pour complexifier encore ce paysage, il faut aussi noter que les utilisateurs sous Windows tendent généralement à employer la machine immédiatement après l'avoir démarré et ouvert une session, ce qu'ils estiment être antérieur à la fin des préparatifs nécessaires pour une expérience complète du système. Les services à démarrage automatique, qui dans ce contexte effectuent leur initialisation en même temps que l'utilisateur entame diverses activités, contredisent par nature ce schéma.

Pour remédier à ce problème, Microsoft avait incorporé à son système d'exploitation un mécanisme nouveau, appelé démarrage différé, dont le principe consiste en l'introduction d'une courte pour la mise en route des services concernés. Cela permet à l'utilisateur de commencer à interagir avec ses applications sans ressentir l'impact des dits services lors de l'ouverture d'une session.

Un certain nombre de services distribués avec Windows sont automatiquement mis en œuvre avec un départ retardé par rapport aux autres services. Parmi ceux-ci figurent Windows Update, le partage réseau Media Player, Windows Defender, etc.

Les concepteurs de logiciel ont la possibilité pour marquer un service comme à démarrage automatique différé d'employer l'API de contrôle de service *ChangeServiceConfig2*, accompagnée de la structure SERVICE_DELAYED_AUTO_START_INFO.

Soulignons, pour en terminer à leur sujet, que les services à démarrage automatique, s'ils constituent une amélioration notable, ne règlent pas tous les problèmes liés aux performances de démarrage des composants s'exécutant en marge du système d'exploitation. Il n'existe, par exemple, aucune garantie quant au moment où sera démarré un service, ni quand aura lieu la commande de contrôle émise dans ce but par le SCM. (Notez que ces limitations, loin d'être imputable à une mauvaise conception des services, découlent du fait que Windows ne soit pas un système d'exploitation temps réel.) De plus, et encore une fois du fait qu'ils n'obéissent pas à une temporalité bien définie, les services à démarrage automatique différé sont difficilement conciliables, voire incompatibles, avec les règles communes en matière de dépendances et d'ordre de démarrage des services Windows. De tels services ne peuvent en conséquence rejoindre aucune des hiérarchies prévues pour gérer les enchaînements de démarrage de services, conçues pour donner le départ à des services ayant besoin entre eux de respecter un ordre donné. Un service à démarrage automatique peut par contre dépendre d'un service service à démarrage automatique différé, cela établissant néanmoins le problème du lien qui unit entre les deux services. Ainsi, comme un service ne peut démarrer que lorsque les services dont il dépend ont eux même démarrés, le SCM lancera le service à démarrage automatique différé lors de la séquence d'initialisation du système, avec comme effet de réduire à néant tous les avantages que comptait procurer le dit service.

Démarrer et arrêter des services

Tous les services n'offrent pas les mêmes possibilités concernant leur statut. Certains empêchent leur arrêt aussi bien que leur démarrage ; d'autres autorisent ces opérations, mais font obstacle dès qu'il s'agit de les suspendre ou de reprendre leur exécution. Certains services n'acceptent des interactions qu'avec un nombre restreint d'identités (utilisateurs ou groupes). Par exemple, la plupart des services ne permettent qu'aux membres des groupes Administrateurs et Utilisateurs avec pouvoir de les démarrer ou de les arrêter. La liste de contrôle d'accès discrétionnaire de chaque service, définie lors de l'enregistrement auprès du système, détermine quels moyens de contrôle sont permis, et qui est susceptible de les mettre en oeuvre.

La marche à suivre à modifier le statut d'un service est la suivante :

1. Ouvrez la console Services (Services.msc depuis une invite de commandes ou le contrôle Exécuter), laquelle offre en premier lieu une vue de l'ensemble des services installés. La colonne État indique si un service est ou non cours d'exécution.

2. Pour démarrer, arrêter, suspendre ou redémarrer un service, sélectionnez-le parmi la liste en effectuant un clic droit. Il vous sera à ce moment proposé ces options. Une autre méthode consiste à faire appel aux contrôles situés dans la barre d'outils ou à la boite de dialogue menant aux propriétés du service (effectuez auquel cas un double clic).

Gérer les services

Windows propose quatre outils principaux avec lesquels gérer des services :

- L'outil d'administration Services (Services.msc)
- Le Gestionnaire des tâches
- La stratégie de groupe
- La commande sc.exe

L'ensemble est intégré en standard dans les diverses versions et déclinaisons des systèmes Microsoft.

Service Control (Sc.exe)

La commande Sc.exe permet d'interagir avec les services (démarrage, suspension et reprise, arrêt) et de gérer les différents aspects les concernant : type de démarrage, dépendances, actions en cas d'échec, etc.

Le tableau qui suit inventorie les différents commutateurs, et leurs effets, disponibles auprès de la commande Sc.

Tableau 5.17. Commandes SC

Commande	Action
query	Interroge l'état d'un service ou énumère l'état des types de services
queryex	Interroge l'état étendu d'un service ou énumère l'état des types de services
start	Démarre un service
pause	Envoie une demande de contrôle PAUSE à un service
interrogate	Envoie une demande de contrôle INTERROGATE à un service
continue	Envoie une demande de contrôle CONTINUE à un service
stop	Envoie une demande STOP à un service
config	Modifie la configuration d'un service (persistant)
description	Modifie la description d'un service
failure	Modifie les actions entreprises par un service en cas d'échec
failureflag	Modifie l'indicateur des actions d'échec d'un service
sidtype	Modifie le type de SID d'un service
privs	Modifie les privilèges requis d'un service
managedaccount	Modifie le service pour marquer le mot de passe du compte de service comme étant géré par LSA
qc	Interroge les informations de configuration d'un service
qdescription	Interroge la description d'un service
qfailure	Interroge les actions entreprises par un service en cas d'échec

Commande	Action
qfailureflag	Interroge l'indicateur des actions d'échec d'un service
qsidtype	Interroge le type de SID d'un service
qprivs	Interroge les privilèges requis d'un service
qtriggerinfo	Interroge les paramètres de déclenchement d'un service
qpreferrednode	Interroge le noeud NUMA favori d'un service
qmanagedaccount	Interroge si un service utilise un compte avec un mot de passe géré par LSA
qprotection	Interroge le niveau de protection de processus
quserservice	Interroge une instance locale d'un modèle de service utilisateur
delete	Supprime un service (du Registre)
create	Crée un service (en l'ajoutant au Registre)
control	Envoie un contrôle à un service
sdshow	Affiche le descripteur de sécurité d'un service
sdset	Définit le descripteur de sécurité d'un service
showsid	Affiche la chaîne du SID de service correspondant à un nom arbitraire
triggerinfo	Configure les paramètres de déclenchement d'un service
preferrednode	Définit le noeud NUMA favori d'un service
getdisplayname	Obtient le nom complet DisplayName d'un service
getkeyname	Obtient le nom de clé d'un service (ServiceKeyName)
enumdepend	Énumère les dépendances d'un service
boot	Indique si le dernier démarrage doit être enregistré comme la dernière configuration de démarrage valide connue
lock	Verrouille la base de données des services
querylock	Interroge l'état de verrouillage (LockStatus) d'une base de données du Gestionnaire de contrôle des services

Onglet Services du Gestionnaires des tâches

L'onglet Services du Gestionnaires des tâches répertorie un certain nombre d'informations concernant les services : nom, ID du processus hôte, description, état (arrêté ou en cours d'exécution) et groupe.

Un processus héberge potentiellement plusieurs services. Pour trier les services par numéro d'identification de processus (PID) associé, cliquez sur l'en-tête de la colonne appropriée. Cliquez sur l'en-tête de la colonne État pour trier les services en fonction de leur état.

En effectuant un clic droit sur un service, vous disposez d'un menu contextuel à partir duquel les actions suivantes sont proposées :

- Démarrer un service arrêté
- Arrêter l'exécution d'un service démarré
- Aller dans le processus associé au service (sur l'onglet Détails)

Services.msc

Le composant enfichable Services de la console MMC permet une gestion souple des services Windows. Pour afficher ce complément, saisissez services.msc à l'intérieur d'une invite de commandes, ce qui devrait donner lieu à l'apparition d'une nouvelle fenêtre graphique.

Notez que des privilèges d'administrateur sont mobilisés pour donner toutes ses capacités à l'outil. En l'exécutant en tant qu'utilisateur standard, vous afficherez les paramètres des services, mais ne pourrez ni démarrer ni

arrêter la plupart d'entre eux, pas plus que modifier le type de démarrage ou procéder à d'autres modifications de configuration.

Erreur de démarrage

Dans l'éventualité où un service ou un pilote signale une erreur en réponse à la commande de démarrage du SCM, la réaction de ce dernier est conditionnée par la valeur ErrorControl de la clé de registre associée au service.

- Si ErrorControl vaut SERVICE_ERROR_IGNORE (0) ou n'est pas spécifiée, le SCM ignore l'erreur et continue le démarrage des services. Tout se passe dans cette configuration comme si aucun problème n'était advenu.
- Si ErrorControl vaut SERVICE_ERROR_NORMAL (1), le SCM enregistre l'événement dans le journal Système et poursuit le démarrage.
- Si ErrorControl vaut SERVICE_ERROR_SEVERE (2), le SCM enregistre l'événement dans le journal Système, bascule la configuration registre du système de sorte à employer la dernière bonne configuration connue (autrement dit les réglages avec lesquels Windows a démarré correctement pour la dernière fois), puis redémarre le système. Si le système était déjà en train de s'amorcer conformément à la dernière bonne configuration connue, il se contente de redémarrer.
- Si ErrorControl vaut SERVICE_ERROR_CRITICAL (3), le SCM enregistre l'événement dans le journal Système (Il se peut à ce stade qu'il n'y parvienne pas), restaure la dernière bonne configuration connue et redémarre l'ordinateur. Si le système était déjà en train de s'amorcer conformément à la dernière bonne configuration connue, il s'arrête, la procédure de démarrage n'allant pas plus loin.

Sécurité des services

Tableau 5.18. Mappage générique des droits d'accès aux fichiers

Droit générique	Signification
GENERIC_ALL	SC_MANAGER_ALL_ACCESS
GENERIC_EXECUTE	SC_MANAGER_CONNECT
	SC_MANAGER_LOCK
	STANDARD_RIGHTS_EXECUTE
GENERIC_READ	SC_MANAGER_ENUMERATE_SERVICE
	SC_MANAGER_QUERY_LOCK_STATUS
	STANDARD_RIGHTS_READ
GENERIC_WRITE	SC_MANAGER_CREATE_SERVICE
	SC_MANAGER_MODIFY_BOOT_CONFIG
	STANDARD_RIGHTS_WRITE

Contrôle des services

Programmes d'installation et d'administration interagissent avec le SCM, indirectement avec les applications service, via une technologie de passage de messages. Dans ce scénario, une application en nécessité d'un service quelconque envoie un code de contrôle, non à l'application service elle-même, mais au SCM, puisque c'est à lui que revient la responsabilité de prendre en charge la commande et de la relayer au service approprié.

La fonctionnalité d'envoi d'un code de contrôle vers à un service est rendu visible dans l'API Windows par la fonction ControlService. Une fois établie une connexion avec le SCM, et négociés les droits d'accès requis pour une commande (les permissions nécessaires dépendent d'elle), un programme appelle ControlService et spécifie, entre autre, un code de contrôle qui représente l'opération souhaitée. Il fournit de même un pointeur vers une structure SERVICE_STATUS, alimentée par une application service pendant le traitement et qui reçoit les dernières informations sur l'état du service. Les constantes et valeurs admissibles au titre de code de contrôle sont énumérées dans le tableau suivant.

Tableau 5.19. Code de contrôle des services

Code	Valeur	Signification
SERVICE_CONTROL_STOP	0x00000001	
SERVICE_CONTROL_PAUSE	0x00000002	
SERVICE_CONTROL_CONTINUE	0x00000003	
SERVICE_CONTROL_INTERROGATE	0x00000004	
SERVICE_CONTROL_SHUTDOWN	0x00000005	
SERVICE_CONTROL_PARAMCHANGE	0x00000006	
SERVICE_CONTROL_NETBINDADD	0x00000007	
SERVICE_CONTROL_NETBINDREMOVE	0x00000008	
SERVICE_CONTROL_NETBINDENABLE	0x00000009	
SERVICE_CONTROL_NETBINDDISABLE	0x0000000A	

Le SCM envoie un code de contrôle à un service que si ce dernier a explicitement mentionné prendre en charge la commande, et que si son état le lui permet. Le SCM traite les notifications à destination des services de manière sérialisé, et bloquera un certain temps si un service est occupé. Si le service n'a toujours pas retourné de sa fonction gestionnaire de contrôle, que le service implémente pour traiter les diverses commandes qu'il reçoit du SCM, ControlService échoue avec le code d'erreur ERROR_SERVICE_REQUEST_TIMEOUT.

Les services en cours de fonctionnement acceptent par défaut le code de contrôle SERVICE_CONTROL_INTERROGATE, qui avertit un service qu'il doit communiquer son état au SCM. Les pilotes n'acceptent pas de codes autres que SERVICE_CONTROL_STOP and SERVICE_CONTROL_INTERROGATE. Chaque service indique les codes de contrôle supplémentaire qu'il accepte avec un appel de la fonction SetServiceStatus.

Erreur des services

Chaque service peut au travers sa configuration de base orienter le comportement de l'ordinateur en cas de défaillance. Le SCM s'entend avec le système de sorte que ce dernier lui signale quand un processus de services se termine. Dans l'éventualité où cette fin résulte d'un dysfonctionnement, le SCM détermine quels sont les services que le processus exécutait et effectue les mesures de récupération spécifiées par les valeurs FailureActions et FailureCommand enregistrées sous la clé de registre idoine.

Quand un service entre dans l'état SERVICE_STOPPED et que le code d'erreur retourné au SCM n'est pas synonyme de succès (ERROR_SUCCESS), le SCM vérifie si l'indicateur des actions en cas d'échec (FailureActionsOnNonCrashFailures) associé au service est actif et, s'il l'est, applique la même procédure de récupération que si le service était bloqué. Pour tirer profit de ce paramètre, les concepteurs de logiciels ont à leur disposition l'API ChangeServiceConfig2, et les administrateurs système la commande Sc.exe et ses commutateurs -failureflag et -qfailureflag.

Les actions qu'un service a la possibilité de configurer pour le SCM sont le redémarrage du service, l'exécution d'un programme, le redémarrage de l'ordinateur et, enfin, l'inaction (qui devient dans ce contexte à la fois une action et la conséquence attendue d'un choix). Il est en outre envisageable de définir des actions différentes selon qu'il s'agit de la première fois où le processus de services échoue, de la seconde, et des fois suivantes. À ce panel de mesures s'ajoute un compteur de défaillances.

Pour spécifier les options de récupération d'un service, allez dans l'onglet Récupération de la boite de dialogue des propriétés du service dans le composant enfichable MMC Services.

Codes d'erreur d'E/S service

Le liste qui suit répertorie les codes d'erreur que Windows emploie en cas d'erreur dans un service. .Codes d'erreur d'E/S service

- ERROR_ACCESS_DENIED Voir fonctions LockServiceDatabase et SetServiceObjectSecurity.
- ERROR_DEPENDENT_SERVICES_RUNNING Le service ne peut être stoppé car d'autres services en cours d'exécution dépendent de lui.
- ERROR_INVALID_HANDLE Le handle spécifié n'a pas été obtenu à l'aide des fonctions opportunes (CreateService ou OpenService), ou n'est plus valide. Voir fonctions LockServiceDatabase et SetServiceObjectSecurity.
- ERROR_INVALID_PARAMETER Voir fonction SetServiceObjectSecurity.
- ERROR_INVALID_SERVICE_CONTROL La commande demandée dans pas valide pour ce service.
- ERROR_SERVICE_CANNOT_ACCEPT_CTRL
- ERROR_SERVICE_NOT_ACTIVE Le service n'a pas été démarré. Voir fonction ControlService.
- ERROR_SERVICE_REQUEST_TIMEOUT Le service n'a pas satisfait la requête dans les temps requis.
- ERROR_SHUTDOWN_IN_PROGRESS Le système est en cours d'extinction, et n'honore par conséquent plus aucune requête de contrôle supplémentaire. Voir fonction ControlService.
- ERROR_SERVICE_DOES_NOT_EXIST Le service spécifié n'existe pas en tant que service installé. Voir fonction OpenService.
- ERROR_INVALID_NAME Le nom de service spécifié n'est pas valide. Voir fonction OpenService.
- ERROR_INVALID_SERVICE_ACCOUNT Le compte sous lequel il est prévu que le service s'exécute n'existe pas. Voir fonctions CreateService et ChangeServiceConfig.
- ERROR_DUPLICATE_SERVICE_NAME Le nom d'affichage existe déjà dans la base de données des services. Voir fonction ChangeServiceConfig.
- ERROR_SERVICE_MARKED_FOR_DELETE Le service est marqué pour suppression, et ne se trouve par conséquent plus en mesure de répondre à une requête. Voir fonctions ChangeServiceConfig et SetServiceObjectSecurity.
- ERROR_FAILED_SERVICE_CONTROLLER_CONNECT Impossible de se connecter au controleur de service. Voir fonction StartServiceCtrlDispatcher.
- ERROR_SERVICE_ALREADY_RUNNING Une instance du service est déjà en cours d'exécution. Voir fonction StartServiceCtrlDispatcher.
- ERROR_SERVICE_DATABASE_LOCKED La base de données des services est verrouillée. Voir fonction LockServiceDatabase.
- ERROR_INVALID_DATA Voir fonction SetServiceStatus.
- ERROR_SERVICE_LOGON_FAILED Le service n'a pu être démarré en raison d'un échec de connexion. Cette erreur se produit si le service est configuré pour s'exécuter sous un compte dépourvu du droit "Se connecter en tant que service"(SeServiceLogonRight). Voir fonction StartService.
- ERROR_SERVICE_EXISTS Le service spécifié existe déjà dans la base de données des services. Voir fonction CreateService.

Arrêt des services

Lors de la phase d'extinction, l'un des comportements attendus de la part du sous-système Windows, est de de boucler sur tous les processus actifs en vue de leur signaler que le système est en cours d'arrêt. Le

processus du SCM, compte tenu des composants avec lesquels il interagit, fait en les circonstances l'objet d'un traitement à part. Ainsi, quand Csrss rencontre ledit processus, il attend un délai prédéfini qui diffère de celui des autres processus, cela afin de prendre en compte les éventuelles opérations de nettoyage que des services auraient à effectuer. Le délai que Csrss laisse au SCM pour se terminer est donnée par la valeur HKLM\SYSTEM \CurrentControlSet\Control\WaitToKillServiceTimeout, qui est de 20 secondes par défaut. (Pour ne pas impacter négativement la procédure d'arrêt, ce qui viendrait dégrader l'expérience utilisateur, il est recommandé de ne pas augmenter cette valeur de manière trop significative.)

Le gestionnaire d'arrêt du SCM a la charge d'envoyer des notifications d'arrêt à tous les services ayant demandé ces notifications lors de leur initialisation. Pour chaque service auquel il envoie une commande d'arrêt, le SCM tient compte de l'*indication d'attente* (*wait hint*)du service. Après l'envoi des messages d'arrêt, le SCM attend soit qu'un des services auxquels il a notifié l'arrêt se termine, soit que le délai spécifié par l'indication d'attente soit écoulé.

Tandis que le SCM est occupé à mettre fin aux services, Csrss attend que le SCM se termine. Sur les versions de Windows antérieures à Vista, dans l'éventualité où l'attente de Csrss se terminait sans que le SCM se soit arrêté (signe que le délai WaitToKillServiceTimeout avait expiré), Csrss tuait le processus et faisait aller de l'avant le processus d'arrêt. Les concepteurs du noyau ont par la suite estimé que cette méthode n'était pas la meilleure, et introduit des notifications de pré-arrêt et l'application d'un ordre d'arrêt.

Les notifications de pré-arrêt sont transmises, en utilisant le même mécanisme que les notifications d'arrêt, aux services les ayant demandé via l'API SetServiceStatus, suite à quoi le SCM attend leurs accusés de réception respectifs. L'idée motrice sous-jacente aux notifications de pré-arrêt est de donner aux services les plus susceptibles de passer du temps dans une procédure d'arrêt (tels qu'un service faisant la passerelle vers des fonctionnalités de base de données) plus de temps pour la mener à bien. Dans cette optique, le SCM envoie une requête de progression et attend un certain laps de temps que le service réponde à cette notification. Si le service ne répond pas dans ce délai imparti, il sera tué pendant la procédure d'arrêt ; autrement, le service peut continuer à fonctionner aussi longtemps qu'il le faut tant qu'il continue de répondre au SCM. Un service peut augmenter le délai de réponse, 3 minutes dans la configuration par défaut, par le biais de l'API ChangeServiceConfig2.

En plus des notifications, le SCM offre aux services la possibilité de redéfinir l'ordre dans lequel il émet des commandes d'arrêt. Les services qui dépendent d'autres services au niveau de l'arrêt peuvent spécifier leurs dépendances dans la valeur de registre HKLM\SYSTEM\CurrentControlSet\Control\PreshutdownOrder.

Processus partagés entre des services

Si exécuter chaque service dans son propre processus est une solution particulièrement adaptée en termes de robustesse (par extension, de fiabilité), procéder ainsi n'en constitue pas moins un éventuel gaspillage des ressources système. Afin de l'empêcher, Windows permet à plusieurs services de résider sous un même toit (comprendre de partager un même processus), et de la sorte mutualiser les aspects techniques qu'ils ont en commun.

Parmi les services prédéfinis de Windows, certains sont configurés pour s'exécuter dans leur propre processus, tandis que d'autres le sont pour se partager un processus. Par exemple, le processus du LSASS héberge différents services liées à la sécurité, tels le service Gestionnaire de comptes de sécurité (SamSs) ou le service Ouverture de session réseau (Netlogon).

Ainsi que nous l'avons sous-entendu plus haut, partager un processus entre plusieurs services n'a pas que des avantages puisque, dans une telle configuration, si l'un des services d'un processus partagé rencontre un dysfonctionnement qui force l'arrêt du processus, il s'ensuit que tous les services dudit processus prennent automatiquement fin. Il est en outre plus difficile dans ce contexte de mesurer l'utilisation des ressources d'un service spécifique.

Windows assure la complémentarité entre plusieurs services par le biais d'un processus générique appelé Service Host (\Windows\System32\Svchost.exe).

Bon nombre de services Windows s'appuient pour leur fonctionnement interne sur SvcHost, dont Appel de procédure distante (RpcSs), Mises à Jour Automatiques, Pare-Feu, Plug and Play, Client DNS, Partage de fichiers, et d'autres.

Windows implémente les services exécutés dans SvcHost par le biais de modules chargeables dynamiquement (DLL). Il exige des services concernées de se présenter par conséquent sous cette forme, et de montrer conformément à cela une valeur nommée ServiceDll, définie dans une sous-clé Parameters, qui pointe vers le fichier DLL apparenté au service.

Tous les services se partageant un même processus SvcHost mettent en évidence par l'intermédiaire de leur configuration l'appartenance au même ensemble, ce qu'ils font en définissant une valeur ImagePath de la forme \Windows\System32\Svchost.exe -k. Quand le SCM découvre, au démarrage des services, le premier service ayant une valeur ImagePath SvcHost dotée d'un paramètre particulier, il créé une nouvelle entrée dans la base de données d'images et exécute un processus SvcHost avec ce paramètre. Par la suite, ledit processus SvcHost recherche une valeur ayant le même nom que le paramètre sous HKLM\SOFTWARE\Microsoft\Windows NT \CurrentVersion\Svchost, qu'il interprète comme une liste de noms de service. Une fois ces éléments réunis, le processus SvcHost informe le SCM qu'il héberge ces services.

Dans l'éventualité où le SCM rencontre un service SvcHost dont la configuration (valeur ImagePath) renvoie à une entrée deja présente dans la base de données d'images, il ne crée pas un nouveau processus, mais envoie une commande de démarrage du service au processus SvcHost auquel il a déjà donné lieu pour cette rubrique. Le processus SvcHost existant lit le paramètre ServiceDll dans la clé de registre associé au service, puis charge la DLL correspondante.

Pour voir depuis une invite de commandes la liste des services exécutés dans des processus, utilisez la commande tasklist /svc.

Balises de service

L'un des inconvénients de partager un processus entre plusieurs services est qu'il est dans ce contexte difficile d'effectuer un bilan fidèle de la consommation isolée de chaque service. (Tous les services hébergés au sein d'un même processus partagent par extension les mêmes ressources, y compris l'espace d'adressage, la table des handles, mais encore différents compteurs qui, à l'échelle d'un processus donné, permettent de mesurer les performances.) Un autre inconvénient de cette approche tient au fait qu'elle peut éventuellement nuire à l'identification des threads appartenant à un même service - soit parce que des threads auxiliaires ont été employés, soit parce ce que l'adresse de début du thread ne met pas en évidence le nom de la DLL à laquelle s'apparente le service. Pour proposer une solution à ces difficultés, Windows intègre la notion de balises de service (service tags).

Le SCM génère la balise correspondante à un service à l'aide de la fonction ScGenerateServiceTag.

Noms des services

Les services ont trois noms (deux selon le point de vue) : le nom du processus exécuté dans le système, le nom interne dans le registre et un autre nom prévu pour l'affichage. Premier et second nom sont directement mis en correspondance, de telle sorte que tout processus de service tire son nom de la clé maintenue pour lui dans le Registre. Le nom d'affichage est un nom convivial qui apparaît et est utilisé dans divers outils d'administration, dont l'utilitaire Services et la commande net start. (Certains services n'ont pas de nom affiché, auquel cas c'est le nom interne qui est montré.)

Les fonctions Windows GetServiceDisplayName et GetServiceKeyName permettent au code mode utilisateur d'interroger les données nominatives relatives à un service. Dans un cas, l'application passe en paramètre de GetServiceDisplayName un nom de processus ou de clé de registre et obtient en retour le nom d'affichage du service sous-jacent. Dans un autre, l'application donne à GetServiceKeyName le nom d'affichage d'un service et reçoit comme résultat le nom de la clé de registre en lien avec le dit service.

Tâches planifiées

SchTasks

La commande schtasks offre une gestion souple du dispositif de planifications des tâches intégré à Windows. Elle permet de créer supprimer, effectuer des requêtes, modifier, exécuter et mettre fin à des tâches planifiées sur un système local ou distant.

Sur le plan pratique, schtasks effectue les mêmes opérations que l'utilitaire graphique Planificateur de tâches, en conséquence de quoi les deux sont utilisables ensemble et de manière interchangeable. A titre informatif, notez que schtasks est apparu dans Windows Server 2003 en remplacement de l'illustre commande at, dont l'utilisation est désormais déconseillée. (Si les raisons qui ont conduit Microsoft à ne pas supprimer ladite commande du programme d'installation de Windows restent à ce jour inconnues, il est raisonnable d'imputer cela à la volonté de prendre en charge les anciens scripts.)

Protection et restauration du système

Windows contient diverses fonctionnalités, parmi lesquels Protection du système et sa contrepartie Restauration du système, qui permettent aux utilisateurs et administrateurs de créer des sauvegardes et, le cas échéant, d'effectuer une récupération des composants névralgiques ou les plus importants de l'ordinateur, y compris les paramètres fondamentaux du système d'exploitation.

Windows crée automatiquement des points de restauration. Ces points sont planifiés de façon régulière par Windows ou sont créés automatiquement lors d'un changement notable du système. Des points de restauration sont automatiquement créés lors des événements que voici :

- Installation d'un pilote.
- Installation d'une application compatible avec la fonctionnalité de restauration système et qui va, en quelque sorte, commanditer la création d'un point de restauration.
- Installation d'une mise à jour via Windows Update.
- Lorsqu'un utilisateur restaure son ordinateur de sorte à le replacer à un état préalablement enregistré.
- Dans le cadre des outils intégrés à Windows pour la sauvegarde et la restauration de données.
- Relativement aux paramètres de la tâche intitulée SR dans le Planificateur de tâches.
- En fonction de la périodicité définie au niveau des clés de registre RPGlobalInterval et RPSessionInterval sous HKLM\Software\Microsoft\Windows NT\CurrentVersion\SystemRestore.

Créer un point de restauration

Windows offre la possibilité de créer manuellement des points de restauration. Cela est potentiellement utile avant d'effectuer une modification susceptible d'impacter négativement le système d'exploitation, en vue de le restaurer plus tard au cas où un problème survenait. Avant toute chose, vérifiez que la fonctionnalité Protection du système de Windows est activée. Pour ce faire, faite apparaitre la boîte de dialogue des propriétés systèmes, cliquez sur le bouton Configurer, puis activez, si elle ne l'est déjà, l'option Activer la protection système. Procédez ensuite comme indiqué dans ce qui suit :

1. Dans Propriétés système, onglet Protection du système, cliquez sur le bouton Créer se trouvant en bas de la fenêtre.
2. Donnez un nom à votre point de restauration. Dans l'idéal, celui-ci doit être représentatif soit de l'état actuel de l'ordinateur ou de l'action que vous comptez mener à terme après la création du point de restauration.

3. Une fois le point de restauration créé, cliquez sur OK pour quitter l'assistant de création de point de restauration.

Pour voir concrètement comment se matérialise un point de restauration sur le système de fichiers, rendez-vous dans le répertoire nommé System Volume Information à la racine du disque.

Restaurer les fichiers et paramètres système

Si vous estimez que votre système n'a plus sa stabilité d'antan ou offre des performances moindres des suites de l'installation d'un logiciel, d'un pilote ou d'une mise à jour de Windows, il est très simple de remédier à ce problème en effectuant une restauration du système.

L'utilitaire de restauration du système permet de restaurer l'ordinateur à n'importe lequel des points de restauration disponibles. Ceux-ci peuvent être de trois types :

- **Manuels** Ils ont été créés en réponse à un besoin expressément formulé par l'utilisateur.
- **Système** Ils ont été planifiés automatiquement par le système d'exploitation.
- **Installation** Ils ont automatiquement été créés lors de l'installation de certaines applications.

Pour effectuer une restauration du système, dirigez-vous d'abord vers l'onglet Protection du système de la boite de dialogue Propriétés système. Ensuite, cliquez sur Restauration du système, sous Restaurer le système. Une nouvelle boite de dialogue intitulée Restauration du système s'affiche. Cliquez sur Suivant, choisissez un point de restauration dans la liste puis validez. (La quantité de points disponibles dépend de vos paramètres pour la création de telles sauvegardes.)À partir de là, Windows vous laisse une dernière occasion de confirmer, au contraire d'infirmer, vos choix. Cliquez sur Terminer pour procéder à la restauration.

Windows Management Instrumentation (WMI)

Windows Management Instrumentation (WMI) est une infrastructure de commande et contrôle intégrée pour Windows.

WMI est une implémentation du modèle Web-Based Enterprise Management (WBEM) qui a pour objectif de définir des standards de gestion pour les équipements informatiques (systèmes, réseaux et périphériques) sur plusieurs plates-formes. S'appuyant en la matière sur une pluralité de normes industrielles régies par le consortium DMTF (Distributed Management Task Force), WBEM a été pensé de telle sorte à couvrir la mise en oeuvre d'un système d'acquisition et de partage des données évoluant dans un milieu distribué. WBEM, par extension WMI, autorise l'accès aux données à partir de nombreuses technologies sous-jacentes, parmi lesquelles Win32, DMI (Desktop Management Interface) et le protocole SMTP (Simple Network Management Protocol).

De nombreux utilitaires d'administration des systèmes et réseaux Windows exploitent WMI, par exemple les composants Propriétés système, Informations système. En outre, la plateforme de script Windows PowerShell possède des commandes spéciales en ce qui concerne l'interaction avec des objets WMI.

Les technologies incorporées à WMI sont pour la plupart fondées sur COM et sur DCOM, ce qui permet aux objets WMI de recourir à des techniques de programmation standard comme l'héritage, la composition ou la surcharge.

Fonctionnalités de WMI

Conçu comme une réponse au besoin d'uniformisation de la gestion des systèmes, des applications et des réseaux d'entreprise, WMI propose une infrastructure de gestion évolutive autour de laquelle se développe un riche panel de fonctionnalités.

- **API d'écriture de script uniforme** Tous les objets gérés sont définis dans un cadre objet commun fondé sur le modèle objet CIM. Les scripts n'ont besoin d'utiliser qu'une seule API, WMI, pour accéder aux informations concernant des sources très diverses.

- **Administration à distance** Par définition, les objets gérés dans WMI sont disponibles pour les applications et les scripts aussi bien localement qu'à distance. La gestion à distance des objets ne requiert aucune tâche supplémentaire.

- **Recherche et navigation** Les applications et les scripts peuvent découvrir les informations disponibles sur un système en énumérant les classes disponibles. Les relations entre objets liés peuvent être détectées et parcourues afin de déterminer l'influence d'une entité gérée sur une autre.

- **Fonctionnalité de requête** L'infrastructure WMI traite les données qu'elle gère comme une base de données relationnelle et permet l'utilisation de requêtes SQL pour filtrer et concentrer les requêtes uniquement sur les données pertinentes.

- **Fonctionnalités de publication d'événements et d'abonnements** Des événements peuvent être demandés pour pratiquement toute modification apportée aux objets gérés du système, que ces objets disposent ou non de fonctionnalités internes d'événement. Les abonnés à des événements peuvent demander la notification d'événements très spécifiques en fonction de leurs intérêts particuliers (et non uniquement des événements prédéfinis par les développeurs). Grâce à une architecture très souple, toute action définie par l'utilisateur peut en principe être effectuée à réception d'un événement donné.

Architecture WMI

WMI se compose de quatre grands composants : applications d'administration, infrastructure WMI, fournisseurs et objets administrés.

- **Applications d'administration** Les applications d'administration sont des applications Windows ou des scripts qui interagissent avec l'infrastructure WMI. Elles accèdent à et affichent ou traitent les données concernant les objets administrés, font s'exécuter des méthodes de fournisseurs WMI, et s'abonnent à des événements en appelant l'API COM pour WMI ou l'API de script pour WMI.

- **Infrastructure WMI** L'infrastructure WMI relie fournisseurs et consommateurs WMI et comporte deux volets : le service WMI (Winmgmt), agissant en tant que médiateur entre les applications de gestion et les fournisseurs de données WMI, et le magasin WMI, qui sert de zone de stockage pour les données statiques liées à WMI, comme les classes définies par les fournisseurs. (Seules les données statiques sur des objets sont stockés dans cette base de données. WMI obtient la plupart des données de façon dynamique à partir du fournisseur lorsqu'un client en fait la demande.) Le magasin WMI utilise un espace de noms qui contient plusieurs sous-espaces que WMI arrange de manière hiérarchisée. Le service WMI crée des espaces de noms, tels que root \default, root\cimv2, pendant l'amorçage du système et préinstalle un ensemble de définitions des données et des classes, y compris les classes Win32, les classes système WMI, et d'autres. Les espaces de noms restants sont créés et alimentés par les fournisseurs WMI. Pour plus d'informations et une liste de fournisseurs WMI, consultez Fournisseurs WMI.

- **Fournisseurs WMI et objets administrés** Les fournisseurs WMI sont des serveurs COM qui surveille un ou plusieurs objets administrés pour WMI. Un objet peut modéliser un composant logique ou physique, par exemple un disque dur, une carte réseau ou un processus, ou un ensemble de composants, par exemple un ordinateur. (L'objet ordinateur contiendrait peut-être l'objet carte réseau, et certainement l'objet disque dur.) En plus de définir des objets, les fournisseurs doivent interfacer WMI avec les objets, et gérer les messages de WMI aux objets administrés.

Chaque fournisseur WMI, qui peut se présenter sous la forme d'une DLL chargée dans le processus gestionnaire de WMI, sous la forme d'une application Windows autonome ou sous la forme d'un service Windows, écrit et publie les définitions des classes pour lesquelles il renvoie les données et effectue des opérations dans le langage

MOF (Managed Object Format). MOF est un langage qui conçoit sous forme de classes des représentations d'éléments.

Un exemple de fournisseur est le fournisseur Registry préinstallé, qui accède aux données du Registre. Le fournisseur Registry dispose d'une classe WMI, StdRegProv, avec de nombreuses méthodes mais aucune propriété. D'autres fournisseurs préinstallés, comme le fournisseur Win32, ont généralement des classes avec de nombreuses propriétés, mais peu de méthodes, telles que Win32_Process ou Win32_LogicalDisk. Le fichier DLL du fournisseur, Stdprov.dll, contient le code qui renvoie dynamiquement des données à la demande des scripts ou des applications clientes.

WMIC

Pour permettre des interactions poussées avec WMI, Windows incluent Wmic.exe (Windows Management Instrumentation Command-line). Tous les objets WMI et leurs propriétés, méthodes comprises, sont de la sorte accessibles à l'aide d'une interface simple et relativement accessible, faisant dudit utilitaire une console d'administration de systèmes avancée.

A l'instar d'autres outils intégrés, par exemple Netsh, la console WMIC honore deux modes de fonctionnement : interactif et script. Elle peut qui plus est être facilement étendue par des scripts, voire d'autres logiciels du même type. Pour invoquer la console WMIC, entrez wmic dans une une invite de commande ou le contrôle Exécuter.

Sécurité WMI

WMI, comme d'autres fonctionnalités internes à Windows, prend en compte la sécurité au niveau de l'espace de noms. Dès lors qu'une application est en mesure de se connecter à un espace de noms, elle peut accéder aux propriétés de tous les objets qui y résident. Un administrateur peut dans l'optique d'interagir sur ces aspects employer l'application Contrôle WMI.

PowerShell

Windows PowerShell, anciennement Microsoft Command Shell (MSH), nom de code Monad, est une interface en ligne de commande et un langage de script au coeur de la stratégie Microsoft pour l'administration de ses nouveaux produits et systèmes. Basé sur la programmation orientée objet, créé à partir du Framework Microsoft .NET, PowerShell aide utilisateurs et administrateurs à contrôler la configuration de Windows, tant coté système qu'applicatif, et fournit une structure logicielle complète via laquelle automatiser les tâches d'administration.

Les buts de PowerShell sont multiples : fournir un environnement à partir duquel gérer l'ordinateur et accéder aux ressources associées, telles que processus, systèmes de fichiers, magasins de certificats, Registre, variables d'environnement, etc., être au moins égal sinon meilleur que le Shell UNIX, direct mètre étalon en matière d'interprétation de commandes, avoir une interopérabilité et une compatibilité avec les commandes et les scripts existants, une meilleure sécurité, et bien d'autres encore.

Combinant interactivité, souplesse et de nombreuses possibilités d'extension, PowerShell ancre dans Windows des technologies qui jusqu'alors faisait défaut, ou ne répondait pas aux critères de qualité requis - il faut à ce sujet reconnaitre que, antérieurement à PowerShell, l'environnement en ligne de commande fourni nativement avec Windows fait pâle figure en comparaison du Shell Unix. Utile pour quiconque souhaite accélérer les tâches répétitives, PowerShell met la puissance du .NET Framework, duquel il est issu, à la portée de personnes qui utilisent la ligne de commande et écrivent des scripts. Il introduit un certain nombre de concepts nouveaux et puissants qui vous permettent d'étendre les connaissances que vous avez acquises et les scripts que vous avez créés dans les environnements de l'invite de commandes Windows et Windows Script Host.

[NOTE]() .Sur l'origine du nom de code Monad # Le nom de code Monad vient du livre Monadologie, une œuvre de 1714 écrite en français par le philosophe allemand Leibniz. Elle affirme que la réalité est composée d'éléments

fondamentaux appelés monades, qui sont intégrés ensemble dans une harmonie pré-établie. De façon analogue, Windows PowerShell est composé d'une série de tâches exécutées par une série de composants. Dans le cas présent, ces composants sont appelés *commandlets* (abrégées en *cmdlets* dans la console de commande), on rencontre aussi l'anglicisme *commandelettes*. #

Considérations sur les langages de script dans Microsoft Windows

Les systèmes d'exploitation Microsoft, même si à des degrés et selon des modalités divers, ont de tout temps incorporé la logistique nécessaire à l'interprétation des commandes informatiques et au traitement par lots. (Un traitement par lots, *batch processing* en anglais, désigne un enchaînement automatique d'une suite de commandes sur un ordinateur sans intervention d'un opérateur.)

C'est en 1981 que le MS-DOS fait son apparition avec les premiers scripts batch. L'interpréteur de commandes standard (command.com) dispose de deux modes de travail distincts. Le premier est le mode interactif, dans lequel l'utilisateur saisit des commandes qui sont ensuite exécutées immédiatement. Le second est le mode de traitement par lots, qui exécute une séquence prédéfinie d'instructions stockées dans des fichiers à l'extension .bat. S'ils possèdent effectivement un jeu de commandes spécifiques pour le traitement par lots (structures de boucle, sauts conditionnels et sauts catégoriques, etc.), les programmes batch pour MS-DOS sont très limités en termes de fonctionnalités.

Après DOS vient Windows, dont l'interface présentée à l'utilisateur est assurée par le programme Explorateur Windows, à la fois shell graphique et gestionnaire de fichiers. Les premières versions de Windows, lesquelles s'appuient encore largement sur MS-DOS, conserve l'interpréteur de commande livré par défaut avec ces systèmes, et du même coup les limitations inhérentes. Cette situation perdurera jusqu'au déclin de la famille de systèmes d'exploitation basée sur Windows 95, incluant Windows 98 et Windows Me, ce dernier se positionnant comme la cinquième, dernière et ultime version de Windows 9x.

Accessible via toutes les versions de Windows à partir de Windows NT, Le programme Cmd (fichier exécutable cmd.exe) remplace COMMAND.COM au titre d'interpréteur de commandes standard. Les fichiers batch du Windows NT sont les fichiers dotés de l'extension .BAT ou .CMD. Leur syntaxe est héritée de COMMAND.COM de MS-DOS, mais est nettement plus puissante.

Apparu avec Windows 98 et l'Option Pack de Windows NT 4.0, Windows Script Host (WSH) dote les systèmes d'exploitation Microsoft d'une puissante technologie de scripts. Concrètement, WSH se présente sous la forme de programmes hôtes, indépendants du langage de programmation utilisé, chargés d'activer l'interpréteur approprié pour un script. Ces interpréteurs peuvent ensuite accéder au système par l'intermédiaire des objets COM/ActiveX scriptables installés sur le système. (Ils peuvent en particulier faire appel à WMI pour interagir avec le système d'exploitation). WSH n'étant pas un langage en soi, il fallait un langage de scripts compatible pour l'exploiter. Les systèmes Windows incluent deux interpréteurs : VBScript et JScript. Des interpréteurs pour d'autres langages, comme Perl ou Python, ont également été réalisés.

Si la standardisation de WSH a offert d'indéniables atouts aux systèmes d'exploitation Windows, l'outil présentait de nombreux défauts. L'absence de langage et d'environnement standard pour l'écriture de scripts d'une part, puisque rien ne garantissait que le langage de scripts compatible WSH choisi par l'administrateur serait disponible. La seule manière d'être certain que le langage de scripts ou la version de WSH était compatible avec le système administré consistait à utiliser un langage de scripts natif, beaucoup moins puissants, et à accepter les problèmes afférents. Par ailleurs, WSH ouvrait une porte aux attaques par code malveillant. Cette vulnérabilité a donné naissance à un flot de virus, de vers et autres programmes indélicats qui ont causé de nombreux dégâts sur les systèmes informatiques, et cela à cause de l'accent mis sur l'automation sans intervention de l'utilisateur. En outre, WSH ne permettait pas une utilisation en ligne de commande. Très clairement, une approche différente était nécessaire pour l'administration des systèmes.

Avec le développement de Windows Vista, Microsoft, conscient du besoin d'un shell en ligne de commande robuste et sûr pour l'administration de ses systèmes d'exploitation, crée Windows PowerShell. PowerShell a été conçu comme un shell, avec un accès complet aux composants de Windows, par l'intermédiaire de .NET Framework, des objets COM (Component Object Model) et d'autres méthodes. D'abord distribué séparément,

il est intégré nativement dans Windows 7. Depuis Windows 8, PowerShell dispose d'une place plus prononcée au sein du système d'exploitation avec un raccourci dans toutes les fenêtres de l'Explorateur de fichiers, dans le menu Fichier. Dans le futur, PowerShell pourrait remplacer les interfaces actuelles, comme cmd.exe, WSH, les outils en ligne de commande, etc., et être totalement intégré aux systèmes Windows.

Vue d'ensemble de Windows PowerShell

Environnements interactif et de script

Avant toute chose un environnement au service des utilisateurs, Windows PowerShell se présente à ces derniers sous la forme d'un *shell*. Ce que l'on nomme shell dans le contexte de cet ouvrage est l'interface en ligne de commande, ou CLI (_Command Line Interface), fonctionnant à partir d'instructions en mode texte. Cela dit, et seconde forme au travers laquelle une couche logicielle peut fournir une interface à un système d'exploitation, un shell peut aussi être de type graphique. L'explorateur Windows est un exemple de shell graphique, unifiant dans un ensemble visuel commun différentes perspectives pour interagir avec Windows. Beaucoup d'utilisateurs réduisent la signification du mot shell au sens d'interface en ligne de commande, ignorant que ce mot a d'autres acceptations. Pour en revenir à PowerShell, il est aussi intéressant de noter quelle place l'outil a su se frayer dans l'écosystème Windows, quelles inflexions il a produit sur les stratégies de Microsoft pour l'administration système de ses produits, toutes maintenant clairement orientée ligne de commandes.

Le langage de script PowerShell, basé sur des tâches, spécialement conçu pour l'administration, permet de contrôler tout un système et ses ressources, que ce soit localement ou à distance. Couplant les avantages d'un langage orienté commande, comme le font les shells Unix, et la gestion d'objets, il a le mérite d'être simple à comprendre, bien documenté et de permettre d'atteindre rapidement des résultats.

Les paramètres initiaux de Windows PowerShell, réglés au plus strict, sont conçus dans l'optique d'un maximum de sécurité. Reflet des conditions dans lesquelles PowerShell charge des fichiers de configuration et exécute des scripts (cf. section suivante : Stratégies d'exécution), l'application de la stratégie d'exécution Restricted par défaut, laquelle bloque systématiquement l'exécution de tout script et active l'interpréteur uniquement pour un usage interactif.

Orientation objet

Contrairement aux commandes d'interface de ligne de commande traditionnelles, Windows PowerShell permet d'utiliser des scripts développés en orienté objet (OOP, *Object-oriented Programming*).

La plupart des shells opèrent dans un environnement de type texte. Cela signifie généralement que la sortie doit être traitée pour effectuer des tâches. Par exemple, si les données d'une commande doivent être envoyées à une autre commande, la sortie de la première doit généralement être remise en forme afin de répondre aux exigences d'entrée de la seconde. A contrario, les commandes Windows PowerShell sont conçues pour traiter des objets, informations structurées allant au-delà de simples chaînes de caractères.

Au lieu de transporter les données sous forme de texte brut, PowerShell le fait sous forme d'objets .NET Framework, ce qui permet aux applets de commande d'accéder directement aux propriétés et aux méthodes d'un objet. Cette évolution simplifie également l'usage du shell, puisque plutôt que de modifier des données textuelles, il est possible de faire référence aux données requises simplement par leur nom, et pour chaque objet voir le type auquel il appartient ainsi que les membres dont il dispose, tout cela depuis l'environnement de commande.

Types de commandes PowerShell

Lorsqu'une commande PowerShell est sollicitée depuis l'interface en ligne de commandes, l'interpréteur en examine le nom afin de définir la tâche à effectuer. Ce processus détermine le type de la commande et la manière

de l'exécuter. Il existe quatre types de commandes PowerShell : applets de commande, fonctions shell, scripts et commandes natives.

Les applets de commande, fonctions shell et scripts, dépendants des technologies mises en oeuvre par lui, sont exécutés dans le processus PowerShell (PowerShell.exe). Les commandes natives, des programmes exécutable autonomes, sont exécutés en tant que processus distincts.

Applets de commande

Ressemblant aux commandes natives de divers autres interpréteurs de commandes, telles que les commandes dir et cd de cmd.exe, les applets de commande constituent le type de commande fondamental (fondamental au sens d'intrinsèque) de Windows PowerShell. Les commandes Windows PowerShell intégrées sont appelées commandlets, pour command applets en anglais, abrégées en cmdlets dans l'interface en ligne de commande. On rencontre parfois l'anglicisme commandelettes.

Les cmdlets utilisent un système de dénomination composé à l'aide de paires verbe-substantif, tous deux souvent anglais et toujours séparés par un tiret. Le verbe indique ce que fait la cmdlet en général. Le substantif précise ce avec quoi la cmdlet fonctionne. Par exemple, la cmdlet get-variable récupère toutes les variables de l'environnement PowerShell et retourne leurs valeurs ou récupère une variable d'environnement spécifiquement nommée et retourne ses valeurs. La liste suivante énumére les verbes couramment associés aux cmdlets :

Get Interroge un objet spécifique ou un sous-ensemble d'un type d'objet.

Set Modifie les paramètres spécifiques à un objet.

Enable Active un paramètre.

Disable Désactive un paramètre activé.

New Crée une nouvelle instance d'un élément.

Remove Supprime une instance d'un élément.

Vous pouvez répertorier toutes les commandes qui incluent un verbe particulier au moyen du paramètre -Verb de Get-Command. Par exemple, pour consulter toutes les applets de commande qui utilisent le verbe Get, tapez Get-Command -Verb Get. Vous devriez voir quelque chose comme ce qui suit :

Le paramètre -Noun est plus utile encore, car il vous permet de consulter une famille des commandes affectant un même type d'objet. Par exemple, si vous souhaitez consulter les commandes disponibles pour la gestion des services, tapez Get-Command -Noun Service.

Une commande n'est pas une applet de commande simplement parce que son nom obéit au modèle verbe-substantif. La commande Clear-Host par exemple, laquelle permet d'effacer le contenu de la fenêtre de console, n'est pas une applet de commande, mais en réalité une fonction interne.

Fonctions shell

Le langage de script défini avec avec PowerShell permet l'organisation du code source en unités de travail logiques, nommées fonctions, avec lesquelles découper un problème global en fractions plus simples. Elément fondamental de PowerShell en tant que shell, mais aussi en tant que language, les fonctions sont dans l'écosystème PowerShell considérées comme des commandes à part entière, au même titre que les scripts, cmdlets et les commandes natives Windows. Elles sont en outre des passerelles vers la création de modules.

Une fonction shell permet d'affecter un nom à une liste de commandes. Les fonctions sont comparables aux sous-routines et aux procédures dans d'autres langages de programmation. Un script diffère d'une fonction en

cela qu'une nouvelle instance du shell est démarrée pour chaque script alors que les fonctions s'exécutent dans l'instance courante du shell.

Commandes natives

PowerShell permet l'exécution depuis l'interface en ligne de commande des programmes et outils natifs intégrés à Windows. Ce que l'on désigne par le terme de "commandes natives Windows" est constitué de tous les exécutables ou commandes qui ne relèvent pas de l'environnement de PowerShell, comme les cmdlets, et qui ont pour le coup une existence autonome, par exemple Calculatrice (calc.exe), Bloc-notes (notepad.exe), Robocopy (robocopy.exe), et des milliers d'autres.

Fonctionnant hors de toute considération pour PowerShell, les commandes natives sont exécutées comme des processus séparés. Leur exécution est donc plus lente que celles des autres commandes, à cause de la séparation des processus. De plus, le traitement de leurs paramètres est différent selon que l'on se trouve dans un environnement Command Prompt ou PowerShell. Aussi, conséquemment à ces raisons, il est recommandé d'éviter la sollicitation de commandes tierces si l'objectif recherché peut être accompli via seulement PowerShell.

Fonctionnant hors de toute considération pour PowerShell, les commandes natives sont exécutées comme des processus séparés. Puisque l'invocation d'une commande native déclenche la création d'un nouveau processus, elles sont moins efficaces que les autres types. Elles possèdent également leurs propres paramètres pour le traitement des commandes, qui diffèrent généralement selon que l'on se trouve dans un environnement Command Prompt ou PowerShell. Aussi, conséquemment à ces raisons, il est recommandé d'éviter la sollicitation de commandes tierces si l'objectif recherché peut être accompli via seulement PowerShell.

Scripts

Si l'environnement interactif (comprendre l'interprète de commandes)livré avec PowerShell suffit pour lancer quelques commandes occasionnelles, son utilisation devient vite inconfortable s'il s'agit de lancer toute une série de commandes ; le recours à des scripts semble auquel cas plus appropriée. PowerShell considère les scripts comme un type de commandes à part entière. Ils sont constitués d'au moins une commande (ou groupe de commandes) et sont contenus dans un fichier texte, qui doit avoir l'extension .ps1.

[NOTE]() .Scripts et sécurité # A l'époque de VBScript, à cause surtout de ce dont ce langage profitait en matière d'interactions avec les applications (notamment de messagerie), il était relativement facile de créer des virus performants. (Le lecteur intéressé à ce sujet peut consulter les diverses documentations - techniques ou non - concernant I Love You, un des premiers vers ayant pris une ampleur mondiale, du même coup très célèbre de par son exposition médiatique.) Une fois le code écrit et enregistré dans un fichier idoine (extension .vbs), il suffisait à la victime de double-cliquer sur ce dernier que le code soit exécuté. Aujourd'hui, maintenant que plus d'attention est porté au domaine de la sécurité informatique, si vous essayez de double-cliquer sur un script PowerShell, cette action lancera un éditeur de texte ou de script. #

Les scripts peuvent être très simples, mais aussi très complexes avec des paramètres, une documentation intégrée et la possibilité d'être lancés à distance.

Activer le support des scripts. Les paramètres de sécurité implémentés dans PowerShell, communément et collectivement connus comme *stratégie d'exécution*, désactivent par défaut le support des scripts. Aussi, si vous tentez d'exécuter un script PowerShell sans avoir au préalable modifié la configuration à cet égard, une erreur survient et le script ne s'exécute pas. En l'absence de définition d'une stratégie d'exécution, un message d'erreur s'affiche à l'écran.

Cette position est qualifiée de *stratégie d'execution resteinte*. En la matière, il s'agit de la stratégie la plus restrictive, au sens où elle bloque systématiquement l'exécution de tout script. Seules les commandes entrées depuis l'invite de commandes PowerShell seront exécutées.

Pour remédier à l'inexécution de script, PowerShell requiert que l'utilisateur change le mode d'exécution avec la commande Set-ExecutionPolicy \<mode d'exécution\>. Il faut pour ce faire être administrateur de la machine. Dans l'exemple ci-après, nous modifions le niveau autorisation affecté aux scripts pour passer à une stratégie où les scripts locaux sont autorisés :

```
PS C:\> Set-ExecutionPolicy RemoteSigned
```

Vous pouvez vérifier que le changement ai bien été pris en compte à l'aide de la commande Get-ExecutionPolicy, laquelle devrait donner lieu à la sortie de commande suivante :

```
PS C:\> Get-ExecutionPolicy
RemoteSigned
PS C:\>
```

WER (*Windows Error Reporting*)

Quand une exception non traitée est interceptée par le filtre idoine, la procédure qui s'ensuit génère des informations de contexte concernant l'application problématique et initie un échange LPC avec le service WER. Ledit service effectue alors une analyse superficielle du programme et détermine à partir de la configuration de rapport d'erreurs (clé AeDebug) le comportement adéquat vis à de l'utilisateur. Dans la plupart des cas, il s'agit d'exécuter l'utilitaire WerFault.exe, qui affiche une boîte de dialogue mettant en évidence l'incident et, éventuellement, proposant une option pour le déboggage.

Sur les systèmes qui présentent la configuration de rapport d'erreurs par défaut, le rapport (un mini dump plus un fichier XML contenant des divers détails, tels les numéros de version des DLL chargées dans le processus), est envoyé au serveur d'analyse en ligne des effondrements de Microsoft. A l'issue de cet examen, si une solution au problème est connue, elle est portée à l'attention de l'utilisateur, l'informant des mesures à prendre en vue résoudre l'incident.

Dans les environnements non connectés à Internet, ou pour lesquels l'administrateur a parmi ses prérogatives de contrôler les informations soumises à Microsoft, il est possible de configurer la destination du rapport d'erreur de telle sorte à ce qu'elle se ramène à un serveur de fichiers interne.

Pour plus de flexibilité, WER reconnait un vaste ensemble de paramètres, configurables via l'éditeur de stratégie de groupe ou en opérant les modifications appropriées dans le Registre. La liste qui suit répertorie les options de configuration associées à WER, stockées sous HKLM\SOFTWARE\Microsoft\Windows\Windows Error Reporting en ce qui concerne la station de travail locale, et HKCU\Software\Microsoft\Windows\Windows Error Reporting pour ce qui est de l'utilisateur interactif.

Description	Type	Valeurs
ConfigureArchive		
Si l'archive est appelée à stocker uniquement les paramètres ou toutes les données	DWORD	1 = Les paramètres seulement 2 = Toutes les données
CorporateWerServer		
Nom du serveur d'entreprise auquel WER enverra des rapports (au lieu de les envoyer à Microsoft).	REG_SZ	Nom de serveur
CorporateWERUseSSL		
Emploi ou non de SSL.	DWORD	0 = Désactivé (valeur par défaut) 1 = Activé
Disabled		
Active les rapports d'erreur	DWORD	0 = Rapports d'erreur activés 1 = Rapports d'erreur désactivés
DontSendAdditionalData		

Description	Type	Valeurs
Contrôle la facon dont Windows fait suite aux demande de données supplémentaires émanant de Microsoft. Prioritaire sur les réglages de consentement.	DWORD	0 = Désactivé (valeur par défaut) 1 = Activé
DontShowUI		
Empêche l'affichage de toute boite de dialogue issue de l'interface utilisateur WER.	DWORD	0 = Désactivé (valeur par défaut) 1 = Activé
DebugApplications\NomExe		
Liste des applications qui obligent l'utilisateur à choisir entre Déboguer et Continuer.	REG_SZ	
DisableQueue		
Désactive la mise en file d'attente des rapports	DWORD	0 = Désactivé (file d'attente activée) 1 = Activé (file d'attente désactivée)
Consent\DefaultConsent		
Choix de consentement par défaut de l'utilisateur.	DWORD	0 = Toujours demander 1 = Paramètres uniquement 2 = Paramètres et données 4 = Tout envoyer
Consent\DefaultOverrideBehavior		
Si le consentement par défaut outrepasse le consentement vertical	DWORD	0 = Ne pas outrepasser (valeur par défaut) 1 = Outrepasser
ExcludedApplications\<NomApplication>		
Liste des applications dont n'émanent pas de rapports d'erreur	REG_SZ	
ForceAdminQueue		
Force l'acheminement des rapports vers la file d'attente de l'administrateur	DWORD	0 = Désactivé (valeur par défaut) 1 = Activé
ForceQueue		
Force l'acheminement des rapports vers la file d'attente de l'utilisateur	DWORD	0 = Désactivé (valeur par défaut) 1 = Activé
LoggingDisabled		
Désactive la journalisation.	DWORD	0 = Activé (valeur par défaut) 1 = Désactivé
MaxArchiveCount		
Taille maximale du stockage d'archive.	DWORD	Nombre de fichiers (minimum 1, valeur par défaut 1000, maximum 5000)
MaxQueueCount		
Profondeur maximale de la file d'attente	DWORD	Nombre de fichiers (minimum 1, valeur par défaut 50, maximum 500)
QueuePesterInterval		
Intervalle entre les invites de vérification de solution adressées à l'utilisateur.	DWORD	Nombre de jours (par défaut, 14)

Tableau 5.20. Fonctions WER

Fonction	Description
WerAddExcludedApplication	Exclut une application donnée du processus de rapports d'erreurs
WerGetFlags	Retourne les paramètres de rapports d'erreurs s'appliquant à un processus donné

Fonction	Description
WerRegisterFile	
WerRegisterMemoryBlock	
WerRemoveExcludedApplication	
WerReportAddDump	
WerReportAddFile	Ajoute un fichier à un rapport
WerReportCloseHandle	
WerReportCreate	Crée un rapport d'erreurs
WerReportSetParameter	
WerReportSetUIOption	
WerReportSubmit	
WerSetFlags	Définit les paramètres de rapports d'erreurs s'appliquant au processus actuel
WerUnregisterFile	
WerUnregisterMemoryBlock	

Journaux d'événements

Il existe deux types généraux de fichiers journaux :

- **Journaux Windows** Journaux employés par le système d'exploitation pour enregistrer les événements système généraux relatifs aux applications, à la sécurité, à l'installation et aux composants système.
- **Journaux des applications et des services** Journaux employés par des applications et des services spécifiques pour enregistrer des événements spécifiques aux applications ou aux services.

Les Journaux Windows sont les suivants :

- **Application** Enregistre les événements consignées par les applications. Son emplacement par défaut est \SystemRoot\System32\Winevt\Logs\Application.evtx.
- **Evénements transférés** Lorsque la transmission d'événements est configurée (et les abonnements à des fournisseurs d'événements paramétrés en ce sens), ce journal enregistre les événements transmis depuis des ordinateurs distants. Son emplacement par défaut est \SystemRoot\System32\Winevt\Config\ ForwardedEvents.evtx.
- **Installation** Enregistre les événements qui résultent de l'instalation de logiciels, donnant des détails sur les tâches d'installation terminées et sur les éventuelles erreurs liées à l'installation. Son emplacement par défaut est \SystemRoot\System32\Winevt\Logs\Setup.evtx.
- **Sécurité** Enregistre les événements relatifs à la protection et à la sécurité de la station de travail, tels que les tentatives valides et non valides d'ouverture de session, ainsi que les événements liés à l'utilisation des ressources, comme la création, l'ouverture ou la suppression de fichiers ou d'autres objets. Son emplacement par défaut est \SystemRoot\System32\Winevt\Logs\Security.evtx.
- **Système** Contient les événements journalisés par le système d'exploitation ou ses composants, par exemple l'échec du démarrage d'un service à l'amorçage, la mise en veille de l'ordinateur, et bien d'autres. Son emplacement par défaut est \SystemRoot\System32\Winevt\Logs\System.evtx.

Parmi les Journaux des applications et des services, citons :

- **Windows PowerShell** Enregistre les activités relatives à l'utilisation de Windows PowerShell. Son emplacement par défaut est \SystemRoot\System32\Winevt\Logs\ Windows PowerShell.evtx.
- **Microsoft\Windows** Journaux qui suivent les événements relatifs aux services et fonctionnalités spécifiques à Windows.

- **Microsoft\Windows\Diagnostic-Performance** Consigne les événements dont résulte une dégradation des performances, par exemple la surexploitation des ressources mémoire, vidéo, processeur, etc.

Windows intègre deux utilitaires majeurs en ce qui concerne les journaux : l'observateur d'événements et l'outil en ligne de commande wevtutil.

Types d'événements

Les informations contenues dans les journaux d'événements sont classées par rang d'importance, allant de faits mineurs comme la réussite d'une opération dans un programme, à des effets et impacts majeurs pour l'utilisation de Windows (arrêt inopiné de l'ordinateur, dysfonctionnement d'un pilote de périphériques, etc.). Les événements sont en outre organisés selon qu'ils concernent, ou non, la sécurité du système d'exploitation. L'ensemble de ces critères donne lieu à une hiérarchie à cinq niveaux :

- **Critique** un événement étiqueté critique rend compte d'un avis négatif sur le fonctionnement global de Windows, ou au moins d'une partie de son écosystème. Par exemple, quand un dispositif matériel génère des erreurs, un événement de type Critique est enregistré, avertissant l'utilisateur que des mesures sont nécessaires.

- **Erreur** Une erreur indique un problème important tel qu'une perte de données ou une perte de fonctions à l'échelle du système. Par exemple, si un service n'a pas pu être chargé au démarrage, un événement de type Erreur est enregistré.

- **Avertissement** Un avertissement est un événement qui n'est pas nécessairement significatif mais qui peut annoncer des problèmes ultérieurs. Par exemple, lors d'une baisse importante de l'espace disque disponible, un événement de type Avertissement peut être enregistré.

- **Information** Un événement d'informations décrit le bon fonctionnement d'un programme, d'un pilote ou d'un service. Par exemple, lorsqu'un pilote est chargé, ou un service démarré correctement, un événement de type Information est enregistré.

- **Audit des succès** Indique un événement de sécurité qui a réussi. Par exemple, si l'audit des succès pour les événements d'ouverture de session des comptes est activé, une entrée d'audit est consignée pour chaque utilisateur validé auprès du système.

- **Audit des échecs** Événement de sécurité qui a échoué. Par exemple, si un utilisateur tente d'accéder au système sans y parvenir, la tentative est enregistrée en tant qu'événement de type Audit des échecs.

Chapitre 6. Gestionnaire d'objets

Windows suit une conception orientée objet, et ce faisant réalise une notion fondamentale des sciences de l'informatique, l'abstraction, par laquelle fait force le caractère sémantique des différentes couches du système (rappelez-vous que les services fournis par l'exécutif de Windows peuvent être groupés en catégorie : gestionnaire de processus, gestionnaire d'E/S, gestionnaires de plug-and-play et d'alimentation, etc.). En utilisant un modèle abstrait pour penser l'environnent local, Windows, tant dans sa structure interne que dans les dispositifs visibles au concepteur d'application et pilote tierce partie, rend possible une manipulation aisée (facilitée au moins) des éléments multiples et complexes sous-jacents. Dans ce chapitre, nous allons voir la plastique du modèle objet.

> **Note**
>
> Le chapitre que vous allez découvrir à une place importante, tant dans la structure narrative de ce livre que ce dont il fait son sujet. Il est le premier à user de détails techniques, à rendre compte des fonctions et routines implémentant telle ou telle fonctionnalités ; le premier à démêler l'intrication entre données, attributs, structures de données et services.

Gestionnaire d'objets

Windows est un système d'exploitation à dominante objet et traite comme tel la plupart des ressources de l'environnement informatique. Il met pour ce faire un oeuvre un composant spécialisé, le gestionnaire d'objets, dont les primitives permettent de traiter tout objet (processus, threads, sémaphores, pilotes de périphériques, et bien d'autres) indépendamment de sa signification physique ou technique.

La raison fondamentale et la vocation première premiere du gestionnaire d'objet est de centraliser les opérations de contrôle et de suivi de ressources qui, sans lui, serait éparpillées dans l'ensemble du système d'exploitation.

Sur une vue globale, le gestionnaire d'objets fournit un ensemble générique d'interfaces pour gérer les entités en mode noyau et les manipuler depuis le mode utilisateur. Le dispositif ainsi constitué permet : (1) la distinction des objets ; (2) la recherche d'un objet particulier ; (3) le partage d'un objet entre plusieurs processus.

Outre proposer des méthodes faisant office de tremplin vers les diverses ressources hébergées au sein de l'environnement, le gestionnaire d'objets implémente des opérations telles que la maintenance d'un espace de noms, pour permettre à tout objet de se voir attribuer un nom ; l'application de la sécurité, pour effectuer des contrôles sur les opérations concernant des objets ; ou encore la mémorisation des comptes de référence, qui servent en interne à gérer le cycle de vie de chaque objet.

Le Gestionnaire d'Objets se compose des éléments que voici :

- Un ensemble de services système de l'exécutif chargés de créer, supprimer, protéger et suivre les objets ; quelques-uns exposés via l'API Windows ou les interfaces mode noyau des pilotes de périphérique.
- Un ensemble de règles et de services à partir desquelles le noyau, les gestionnaires de l'exécutif et sous-systèmes d'environnement peuvent construire leurs propres versions des objets, ce faisant imaginer d'autres ressources.
- Un ensemble de primitives génériques, fonctionnelles en différents points du cycle de vie des objets (ouverture, fermeture, interrogation, modification, etc.), qui rendent possible un mécanisme commun uniforme pour l'utilisation des ressources système.
- Un petit nombre d'interfaces qui font contact avec le sous-système de sécurité pour permettre le contrôle d'accès discrétionnaire et l'audit sur les objets qui doivent être protégés.
- Un modèle d'espace de noms pour la catégorisation des objets, avec en arrière-plan les objets répertoires d'objets et liens symboliques pour l'organisation de ce lieu (les deux types d'objet appartiennent au gestionnaire d'Objets).

Structure des objets

Chaque objet Windows se présente sur le plan structurel sous la forme d'un entête et d'un corps. Les entêtes renferment des informations d'état sous l'égide du gestionnaire d'objet, par exemple le nom de l'objet, la catégorie à laquelle il appartient (processus, threads, etc), et d'autres. Les composants de l'exécutif contrôlent les corps des types d'objets dont ils assurent la supervision.

```
lkd> dt nt!_OBJECT_HEADER
   +0x000 PointerCount     : Int8B
   +0x008 HandleCount      : Int8B
   +0x008 NextToFree       : Ptr64 Void
   +0x010 Lock             : _EX_PUSH_LOCK
   +0x018 TypeIndex        : UChar
   +0x019 TraceFlags       : UChar
   +0x019 DbgRefTrace      : Pos 0, 1 Bit
   +0x019 DbgTracePermanent : Pos 1, 1 Bit
   +0x01a InfoMask         : UChar
   +0x01b Flags            : UChar
   +0x01b NewObject        : Pos 0, 1 Bit
   +0x01b KernelObject     : Pos 1, 1 Bit
   +0x01b KernelOnlyAccess : Pos 2, 1 Bit
   +0x01b ExclusiveObject  : Pos 3, 1 Bit
   +0x01b PermanentObject  : Pos 4, 1 Bit
   +0x01b DefaultSecurityQuota : Pos 5, 1 Bit
   +0x01b SingleHandleEntry : Pos 6, 1 Bit
   +0x01b DeletedInline    : Pos 7, 1 Bit
   +0x01c Reserved         : Uint4B
   +0x020 ObjectCreateInfo : Ptr64 _OBJECT_CREATE_INFORMATION
   +0x020 QuotaBlockCharged : Ptr64 Void
   +0x028 SecurityDescriptor : Ptr64 Void
   +0x030 Body             : _QUAD

lkd> dt nt!_OBJECT_CREATE_INFORMATION
   +0x000 Attributes       : Uint4B
   +0x008 RootDirectory    : Ptr64 Void
   +0x010 ProbeMode        : Char
   +0x014 PagedPoolCharge  : Uint4B
   +0x018 NonPagedPoolCharge : Uint4B
   +0x01c SecurityDescriptorCharge : Uint4B
   +0x020 SecurityDescriptor : Ptr64 Void
   +0x028 SecurityQos      : Ptr64 _SECURITY_QUALITY_OF_SERVICE
   +0x030 SecurityQualityOfService : _SECURITY_QUALITY_OF_SERVICE
```

La liste qui suit répertorie les attributs que comporte l'entête d'un objet.

- *Corps d'objet (Body)* Données spécifiques à l'objet.
- *ExclusiveObject* L'objet ne peut être utilisé que par le processus lui ayant donné lieu. Voir macro InitializeObjectAttributes (a = OBJ_EXCLUSIVE).
- *Flags* Modificateurs facultatifs qui définissent le comportement ou les caractéristiques de l'objet.
- *Compteur de références (HandleCount)* Compte le nombre de fois qu'un handle a été ouvert pour l'objet.
- *Objet noyau (KernelObject)* Détermine les conditions d'accessibilité du handle : mode noyau exclusivement ou mode noyau et mode utilisateur. Voir routines InitializeObjectAttributes (a = OBJ_KERNEL_HANDLE) et ObpAllocateObject (OwnershipMode = KernelMode).
- **Accès noyau uniquement (KernelOnlyAccess)** Détermine si l'objet peut être accédé uniquement en mode noyau ou s'il peut aussi l'être depuis un processus mode utilisateur.
- *Lock* Encapsule une primitive de verrouillage (type push lock) employée de telle sorte à protéger la structure des accès concurrents.

- *Etat d'insertion (NewObject)* Indique que l'objet a été créé mais n'est pas encore inséré dans l'espace de noms.
- *Objet permanent (PermanentObject)* Définit le comportement de rétention d'objet lié au décompte des références. Les objets permanents, contrairement aux temporaires, ne sont pas automatiquement supprimées de la mémoire quand le nombre de handles ouverts sur eux tombe à zéro. Voir services NtMakePermanentObject et NtMakeTemporaryObject ; routine ObMakeTemporaryObject.
- *Compteur de pointeurs (PointerCount)* Compte le nombre de fois qu'un composant mode noyau a référencé l'adresse de l'objet. Voir routine ObpIncrPointerCount.
- *Quota facturés (QuotaBlockCharged)* Énumère les facturations de ressource infligées à un processus quand il ouvre un handle vers l'objet.
- *Descripteur de sécurité (SecurityDescriptor)* Détermine qui peut utiliser l'objet et quelles opérations il est possible de réaliser dessus. Voir routines ObGetObjectSecurity et ObReleaseObjectSecurity ; voir attribut OB_FLAG_SECURITY.

Tableau 6.1. OBJECT_HEADER InfoMask

Bit	Type
0x01	OBJECT_HEADER_CREATOR_INFO
0x02	OBJECT_HEADER_NAME_INFO
0x04	OBJECT_HEADER_HANDLE_INFO
0x08	OBJECT_HEADER_QUOTA_INFO
0x10	OBJECT_HEADER_PROCESS_INFO

Tableau 6.2. OBJECT_HEADER Flags

Flags	Constante	Valeur
NewObject	OB_FLAG_NEW_OBJECT	0x01
KernelObject	OB_FLAG_KERNEL_OBJECT	0x02
KernelOnlyAccess	OB_FLAG_KERNEL_ONLY_ACCESS	0x04
ExclusiveObject	OB_FLAG_EXCLUSIVE_OBJECT	0x08
PermanentObject	OB_FLAG_PERMANENT_OBJECT	0x10
DefaultSecurityQuota	OB_FLAG_DEFAULT_SECURITY_QUOTA	0x20
SingleHandleEntry	OB_FLAG_SINGLE_HANDLE_ENTRY	0x40
DeletedInline	OB_FLAG_DELETED_INLINE	0x80

Catégories d'objet

En interne Windows a deux types d'objets : les objets exécutif et les objets noyau. Les premiers ont à charge les rôles centraux du système d'exploitation. Les objets noyau sont des objets plus primitifs qui servent aux opérations de bas niveau en lien avec le contrôle de l'environnement.

Objets noyau

Au même titre que l'exécutif dépeint en tant qu'objets les ressources, le noyau implémente un ensemble d'objets plus simples, appelés objets noyau, l'aidant à contrôler le traitement central et à créer les rudiments de la création d'objets par les divers autres composants du système (gestionnaire de processus, gestionnaire de mémoire, sous-système d'E/S, etc.)

Les objets noyau fournissent des fonctionnalités fondamentales comme la synchronisation, l'ordonnancement et la distribution (dispatching) des threads, sur lesquelles s'appuient les objets exécutif. De ce fait, nombre de ces structures encapsulent un ou plusieurs objets noyau. Chaque thread est par exemple défini au niveau de la sphère exécutive par une structure ETHREAD, laquelle renferme des informations stratégiques telle l'identifiant

ou le code de statut, et dans la perspective du noyau par une structure KTHREAD, contenant entre autre les données nécessaire pour un basculement de contexte (adresse de base et adresse supérieure de la pile noyau, données dispatcher, etc.).

Les objets noyau, conformément à ce à quoi ils invitent, se décomposent en deux sous-ensembles. Un premier ensemble, qui regroupe des objets dits de contrôle, sert de base pour le contrôle de diverses fonctionnalités du système d'exploitation. Cet ensemble inclut l'objet APC, l'objet DPC et l'objet interruption. Un autre ensemble d'objets, appelés objets dispatcher pilote les fonctionnalités de synchronisation qui affectent l'ordonnancement des threads. Les objets incluent le mutex noyau, le sémaphore, le thread (qui est sous Windows une entité que l'on peut attendre), le timer.

La tableau qui suit énumère les différents objets qui voient leur existence assurée par l'intermédiaire du noyau Windows.

Tableau 6.3. Objets noyau

Type d'objet	Symbole (KOBJECTS)	Structure
Interruptions APC	ApcObject	KAPC
Appel de procédure différée	DpcObject	KDPC
DPC threadé	ThreadedDpcObject	
Interruption	InterruptObject	KINTERRUPT
Événement (type notification)	EventNotificationObject	KEVENT
Événement (type synchronisation)	EventSynchronizationObject	KEVENT
Evénements appariés	EventPairObject	KEVENT_PAIR
Mutex gardé		KGUARDED_MUTEX
Mutex	MutantObject	KMUTEX, KMUTANT
Processus	ProcessObject	KPROCESS
Profil	ProfileObject	KPROFILE
File	QueueObject	KQUEUE
Sémaphore	SemaphoreObject	KSEMAPHORE
Spin lock		KSPIN_LOCK
Thread	ThreadObject	KTHREAD
Minuterie (type notification)	TimerNotificationObject	KTIMER
Minuterie (type synchronisation)	TimerSynchronizationObject	KTIMER
Bloc d'attente		KWAIT_BLOC
Gate	GateObject	KGATE

L'énumération KOBJECTS dresse une liste effective de tous les objets noyau qui renferment un entête dispatcher.

Objets exécutif

Les objets et services d'objet de l'exécutif sont des primitives que les divers composants de l'exécutif (gestionnaire de processus, gestionnaire de mémoire, sous-système d'E/S, et ainsi de suite) définissent afin de se les approprier au sein des technologies auxquelles ils donnent lieu. Le gestionnaire de processus, par exemple, met en oeuvre les processus, threads, et jobs sous la forme d'objets exécutif idoines.

Les objets et services d'objet de l'exécutif sont des primitives que les divers composants de l'exécutif (gestionnaire de processus, gestionnaire de mémoire, sous-système d'E/S, et ainsi de suite) définissent afin de se les approprier au sein des technologies auxquelles ils donnent lieu. Le gestionnaire de processus, par exemple, met oeuvre les processus, threads, et jobs sous la forme d'objets exécutif idoines.

Une autre clientèle, importante également, des objets exécutif est constituée des sous-système d'environnement, agissant auquel cas le plus souvent pour le compte d'une application utilisateur. Ainsi, pour créer un fichier, une application Windows appelle la fonction *CreateFile*, implémentée dans la DLL du sous-

système Windows et qui sollicite à son tour le service natif *NtCreateFile*, de sorte à entrainer la création d'un objet exécutif de type fichier.

Une large majorité des ressources exposées au niveau du sous-système Windows correspondent directement à des objets exécutifs (qui s'appuient eux-mêmes sur des objets noyau correspondants), par exemple les processus et les threads. Quelques autres objets, en comparaison, donnent lieu à plusieurs primitives Windows, par exemple les objets de type fichier, sur lesquelles sont basés les tubes nommés et les mailslots.

La liste qui suit énumère les objets existants parmi la couche exécutive, ainsi que les structures de données associées. Nombre de ces objets sont accessibles en l'état aux API de Windows, tandis que quelques autres servent uniquement aux composants de l'exécutif, et ne sont en l'occurrence pas exportés vers les autres sphères du système d'exploitation. Citons, comme exemples de ces objets, ceux sous-jacents aux périphériques, aux pilotes, ou aux événements à clé.

Tableau 6.4. Objets exécutif

Type d'objet	Défini par	Structure
Boite aux lettres (*mailslots*)	Gestionnaire d'E/S	FILE_OBJECT
Bureau	Sous-système graphique	
Clé	Gestionnaire de configuration	CM_KEY_BODY
Evénements appariés	Exécutif	EEVENT_PAIR
Fenêtre	Sous-système graphique	
Fichier	Gestionnaire d'E/S	FILE_OBJECT
Jeton d'accès	Sous-système sécurité	TOKEN
Job	Gestionnaire de processus	EJOB
LPC	Gestionnaire LPC	LPCP_PORT_OBJECT
Lien symbolique	Gestionnaire d'objet	OBJECT_SYMBOLIC_LINK
Minuterie	Exécutif	ETIMER
Processus	Gestionnaire de processus	EPROCESS
Section	Gestionnaire de mémoire	SECTION
Thread	Gestionnaire de processus	ETHREAD
Tube nommé	Gestionnaire d'E/S	FILE_OBJECT
Pilote	Gestionnaire d'E/S	DRIVER_OBJECT
Périphérique	Gestionnaire d'E/S	DEVICE_OBJECT

- **Bureau** Surface logique d'affichage où s'organisent des fenêtres, des menus, raccourcis et autres éléments d'interface utilisateur.
- **Clé** Ouvre la possibilité de manipuler les données du registre. Une clé est une entité hiérarchique qui contient des sous-clés et/ou des valeurs, permettant de structurer les données de configuration du système et des applications.
- **Événement** Objet à état (signalé ou non) avec lequel des processus/threads ont la possibilité de se synchroniser dans l'attente qu'une opération soit terminée ou qu'une condition soit remplie.
- **Fichier** Instance d'un fichier ouvert ou d'un périphérique d'E/S, par exemple un disque ou une imprimante, utilisée pour interagir de manière uniforme avec ces ressources.
- **Jeton** Profil de sécurité associé à un processus ou à un thread, contenant des informations sur les droits et privilèges de l'utilisateur pour le compte duquel il s'exécute.
- **Job** Collection de processus envisageables comme un tout, ce qui permet d'en faciliter la gestion, notamment pour imposer des restrictions ou surveiller leur exécution.
- **Lien symbolique** Mécanisme qui, partant d'un objet, permet de faire référence indirectement à un autre objet, comme un fichier ou un dossier, à l'aide d'un chemin alternatif.

- **Mutex** Primitive de synchronisation utilisée pour contrôler l'accès exclusif à une ressource partagée, par exemple un fichier ou une région de mémoire, garantissant qu'une seule entité à la fois peut y accéder.
- **Port** Mécanisme de communication interprocessus qui permet à des processus de s'envoyer des messages de manière synchrone ou asynchrone, que ce soit entre le mode utilisateur et le noyau ou entre deux processus utilisateur.
- **Processus** Informations nécessaires à la gestion d'un espace d'adressage virtuel et à l'exécution d'un ou plusieurs threads.
- **Section** Région de mémoire qui peut être partagée entre plusieurs processus, utilisée pour permettre des échanges de données ou pour mapper des fichiers.
- **Sémaphore** Compteur pour accès à une ressource permettant à un nombre maximal de threads d'accéder à la ressource à laquelle s'apparente le sémaphore.
- **Station fenêtre** Objet contenant un groupe d'objets bureau, associé à une session utilisateur, et gérant les interactions entre le système et les éléments graphiques qu'incorpore ladite session.
- **Thread** Entité exécutable logeant au sein d'un processus, représentant un flux d'instructions pouvant être exécuté indépendamment.

Etant donné l'étroite proximité entre les sphères hautes et basses du système d'exploitation, il n'est pas rare de voir un objet exécutif encapsuler un ou plusieurs objets noyau. Chaque processus, par exemple, représenté au niveau exécutif par une structure EPROCESS, héberge une sous-structure du type KPROCESS, elle établie sur le plan du noyau.

Autres objets

Divers objets courants utilisés dans Windows touchent à l'esthétique du système d'exploitation, et tirent auquel cas leur origine soit du sous-système de fenêtrage (objets d'interface utilisateur) soit de graphisme (objets GDI). Le système emploie des objets utilisateur afin de prendre en charge les composants visuels et interactifs qu'intègre une application - y compris fenêtres, boutons, boîtes de dialogue, etc. - et gérer les événements qui en découlent, tels que les clics de souris et les frappes clavier. Les objets GDI servent à réaliser les opérations fondamentales requises pour afficher des éléments à l'écran.

Objets d'interface utilisateur

Un objet d'interface utilisateur représente une entité graphique ou interactive qui fait partie de l'interface utilisateur d'une application. Il peut s'agir de fenêtres, de boutons, de zones de texte, de menus, etc. (Voir ci-après pour une liste exhaustive, accompagnée des éclairages et des précisions idoines.) Chacun de ces objets est généralement représenté par une classe et est manipulé à l'aide de fonctions spécifiques de l'API Windows. Par exemple, chaque fenêtre montrée au sein d'une application Windows dérive d'une classe donnée (`WNDCLASS`), laquelle contrôle le comportement et l'apparence générale de toutes les fenêtres associées à ladite classe. Ces attributs comprennent le style, la procédure de fenêtre (fonction de rappel), les handles d'icône et de curseur, la brosse de fond, etc. - **Fenêtre** Région affichée à l'écran qui peut contenir du contenu graphique, des contrôles interactifs et d'autres éléments d'interface utilisateur. Principale fonction associée : `CreateWindow`.

- **Menu** Liste d'options affichée à l'utilisateur, généralement dans la barre de menu d'une fenêtre, qui contient des éléments tels que des commandes, des sous-menus, des séparateurs, etc. Principale fonction associée : `CreateMenu`.
- **Icone** Représentation graphique d'une action envisageable depuis une interface utilisateur. Principale fonction associée : `CreateIconIndirect`.
- **Hook** Surveille différents événements en lien avec l'interface utilisateur : appuis de touches au clavier, activités de la souris, opérations sur une fenêtre, etc. Principale fonction associée : `SetWindowsHookEx`.

- **Curseur** Forme de curseur personnalisée utilisable en remplacement du curseur de souris standard. Principale fonction associée : `CreateCursor`.
- **Table d'accélérateurs** Associe des raccourcis clavier (accélérateurs) à des commandes spécifiques dans une application Windows. Principale fonction associée : `CreateAcceleratorTable`.
- **Caret** Forme de curseur personnalisée utilisable dans une fenêtre ou un contrôle. Principale fonction associée : `CreateCaret`.
- **Conversation DDE** Conversation entre deux applications conformes au modèle d'échange dynamique de données (DDE, Dynamic Data Exchange). Principale fonction associée : `DdeConnect`.

Les handles émanant de ces objets ont un petit nombre de particularités communes :

- **Non rétention** Contrairement à d'autres objets du système, qui encapsulent de manière sous-jacente un compteur de références, il n'existe pas en ce qui concerne les objets d'interface utilisateur de mécanisme global de suivi.
- **Limitation à une station-fenêtre** Les handles se rapportant à l'un ou l'autre des objets susmentionnés sont contraints à une seule station-fenêtre. Ils peuvent de le sorte être échangées librement entre toutes les applications partageant un bureau (par exemple, une session de bureau Windows), mais pas au-delà.

Objets GDI

L'interface de périphérique graphique (GDI, Graphics Device Interface) fait elle aussi usage d'objets et de handles. Entrent à cet égard en ligne de compte des objets tels que contexte de périphérique (`HDC`), stylet (`HPEN`), pinceau (`HBRUSH`), palette de couleurs (`HPALETTE`), et d'autres.

- **Métafichier** Séquence d'enregistrements GDI utilisables pour stocker des dessins vectoriels (lignes, formes, textes) plutôt que des images bitmap. Principale fonction associée : `CreateMetaFile`.
- **Brosse** Instrument graphique dédié à l'application de couleurs, au remplissage de zones, et à l'ajout de motifs sur des formes, des régions ou d'autres composants graphiques. Principale fonction associée : `CreateBrushIndirect .`
- **Contexte de périphérique** Caractéristiques et spécificités d'un périphérique de sortie, tel qu'une imprimante ou un écran, permettant l'exécution d'opérations graphiques. Principale fonction associée : `CreateDC`.
- **Police** Caractéristiques d'affichage du texte, telles que le nom de la police, la taille, le style (gras, italique, souligné), l'orientation, etc. Principale fonction associée : `CreateFont`.
- **Palette de couleurs** Ensemble structuré de couleurs définies pour une utilisation dans des opérations graphiques. Principale fonction associée : `CreatePalette`.
- **Stylet** Caractéristiques du trait lors du dessin dans un contexte graphique. Principale fonction associée : `CreatePen`.
- **Région** Partie délimitée d'un espace graphique. Principale fonction associée : `CreateRectRgn`.
- **Bitmap** Représentation d'une image sous forme de matrice de pixels. Principale fonction associée : `CreateBitmap`.

À noter que, à l'instar des objets d'interface utilisateur, les objets GDI n'honorent aucune solution automatique de comptage des références - il n'existe en réalité qu'un seul handle par objet - et sont privés à un processus. Dans la pratique, cela signifie que seul le processus ayant donné naissance à un objet GDI est susceptible de le manipuler.

Objets type

```
lkd> dt nt!_OBJECT_TYPE
   +0x000 TypeList         : _LIST_ENTRY
   +0x010 Name             : _UNICODE_STRING
   +0x020 DefaultObject    : Ptr64 Void
   +0x028 Index            : UChar
   +0x02c TotalNumberOfObjects : Uint4B
   +0x030 TotalNumberOfHandles : Uint4B
   +0x034 HighWaterNumberOfObjects : Uint4B
   +0x038 HighWaterNumberOfHandles : Uint4B
   +0x040 TypeInfo         : _OBJECT_TYPE_INITIALIZER
   +0x0b8 TypeLock         : _EX_PUSH_LOCK
   +0x0c0 Key              : Uint4B
   +0x0c8 CallbackList     : _LIST_ENTRY
```

- **Nombre maximal de handles (HighWaterNumberOfHandles)** Nombre maximal de handles associés à des objets d'un type spécifique depuis le dernier démarrage du système.
- **Nombre maximal d'objets (HighWaterNumberOfObjects)** Nombre maximal d'objets d'un type spécifique qui ont existé en même temps depuis le dernier démarrage du système.
- *Balise de pool (Key)* Séquence de quatre octets donnée à l'allocation. Voir routine ObCreateObjectType.
- **Nom du type (Name)** Noms des objets de ce type, par exemple « Process », « Event », « Desktop ». Voir routine ObCreateObjectType (attribut TypeName).
- *Nombre total des objets (TotalNumberOfObjects)* Nombre total des objects actifs de ce type.
- *Nombre total des handles (TotalNumberOfHandles)* Nombre total des handles actifs ouverts pour objets de ce type.
- *TypeLock* Verrou global posé sur les objets de ce type lors des opérations qui le nécessitent.

```
lkd> dt nt!_OBJECT_TYPE_INITIALIZER
   +0x000 Length           : Uint2B
   +0x002 ObjectTypeFlags  : UChar
   +0x002 CaseInsensitive  : Pos 0, 1 Bit
   +0x002 UnnamedObjectsOnly : Pos 1, 1 Bit
   +0x002 UseDefaultObject : Pos 2, 1 Bit
   +0x002 SecurityRequired : Pos 3, 1 Bit
   +0x002 MaintainHandleCount : Pos 4, 1 Bit
   +0x002 MaintainTypeList : Pos 5, 1 Bit
   +0x002 SupportsObjectCallbacks : Pos 6, 1 Bit
   +0x002 CacheAligned     : Pos 7, 1 Bit
   +0x003 ObjectTypeFlags2 : UChar
   +0x003 UseExtendedParameters : Pos 0, 1 Bit
   +0x003 Reserved         : Pos 1, 7 Bits
   +0x004 ObjectTypeCode   : Uint4B
   +0x008 InvalidAttributes : Uint4B
   +0x00c GenericMapping   : _GENERIC_MAPPING
   +0x01c ValidAccessMask  : Uint4B
   +0x020 RetainAccess     : Uint4B
   +0x024 PoolType         : _POOL_TYPE
   +0x028 DefaultPagedPoolCharge : Uint4B
   +0x02c DefaultNonPagedPoolCharge : Uint4B
   +0x030 DumpProcedure    : Ptr64     void
   +0x038 OpenProcedure    : Ptr64     long
   +0x040 CloseProcedure   : Ptr64     void
   +0x048 DeleteProcedure  : Ptr64     void
   +0x050 ParseProcedure   : Ptr64     long
   +0x050 ParseProcedureEx : Ptr64     long
   +0x058 SecurityProcedure : Ptr64     long
```

```
+0x060 QueryNameProcedure : Ptr64     long
+0x068 OkayToCloseProcedure : Ptr64     unsigned char
+0x070 WaitObjectFlagMask : Uint4B
+0x074 WaitObjectFlagOffset : Uint2B
+0x076 WaitObjectPointerOffset : Uint2B
```

- **Sensibilité à la casse (CaseInsensitive)** Détermine si les procédures de résolution de nom que le gestionnaire d'objets effectue pour ce type sont faites de manière sensible à la casse. Voir routine ObInitSystem ; variable noyau ObpCaseInsensitive.
- **Coût d'allocation (pool non paginé) (DefaultNonPagedPoolCharge)** Mesure ce que représente, en terme de charge facturée à la mémoire (nombre d'octets), le coût d'une allocation d'un objet de ce type à partir du pool non paginé. Voir routine ObQueryTypeInfo.
- **Coût d'allocation (pool paginé) (DefaultPagedPoolCharge)** Mesure ce que représente, en terme de charge facturée à la mémoire (nombre d'octets), le coût d'une allocation d'un objet de ce type à partir du pool paginé. Voir routine ObQueryTypeInfo.
- **Mappage générique des droits d'accès (GenericMapping)** Association entre les quatre droits d'accès génériques (lire, écrire, exécuter, tout) vers des droits spécifiques propres à un type d'objet. Voir structure `GENERIC_MAPPING`.
- **Attributs non conformes (InvalidAttributes)** Ensemble des indicateurs d'attributs (`OBJ_xxx`) inapplicables en regard de la nature des objets d'un type spécifique.
- **Taille (Length)** Taille en octets de la structure `OBJECT_TYPE_INITIALIZER`.
- **Procédure de validation de fermeture (OkayToCloseProcedure)** Pointeur vers une fonction dont le rôle est de déterminer si une application ou une fenêtre peut être fermée en toute sécurité.
- **Service d'ouverture (OpenProcedure)** Pointeur vers une routine que le gestionnaire d'objet exécute chaque fois qu'il crée un handle pour des objets de ce type.
- **Pool consommé (*PoolType*)** Détermine à partir duquel des pools généraux (pool paginé ou non paginé) sont effectuées les allocations pour des objets de ce type.
- **Sécurité obligatoire (SecurityRequired)** Détermine si la création des objets d'un type donné nécessite inéluctablement l'association d'un descripteur de sécurité.
- **Masque d'accès valide (*ValidAccessMask*)** Indique quels droits d'accès sont valides dans la perspective des objets de ce type.

Tableau 6.5. Routines concernant les Objets type

Routine	Description
ObCreateObjectType	Crée un nouveau type d'objet

Le tableau suivant dresse une liste des types d'objet.

Tableau 6.6. Types d'objet

Symbole	Nom	Balise	Stucture
AlpcPortObjectType	"ALPC Port"		ALPC_PORT
CmpKeyObjectType	"Key"	"Key "	
DbgkDebugObjectType	"DebugObject"		DEBUG_OBJECT
EtwpRegistrationObjectType			
ExCallbackObjectType	"Callback"	"Call"	CALLBACK_OBJECT
ExDesktopObjectType	"Desktop"	"Desk"	
ExEventObjectType	"Event"	"Even"	EEVENT

Symbole	Nom	Balise	Stucture
ExEventPairObjectType	"EventPair"	"Even"	KEVENT_PAIR
ExMutantObjectType	"Mutant"	"Muta"	KMUTANT
ExProfileObjectType	"Profile"	"Prof"	KPROFILE
ExSemaphoreObjectType	"Semaphore"	"Sema"	KSEMAPHORE
ExTimerObjectType	"Timer"	"Time"	KTIMER
ExWindowStationObjectType	"WindowStation"	"Wind"	
ExpKeyedEventObjectType	"KeyedEvent"		KEYED_EVENT_OBJECT
ExpWorkerFactoryObjectType			
IoAdapterObjectType	"Adapter"	"Adap"	ADAPTER_OBJECT
IoCompletionObjectType	"IoCompletion"	"IoCo"	IO_COMPLETION
IoControllerObjectType	"Controller"	"Cont"	CONTROLLER_OBJECT
IoDeviceHandlerObjectType			
IoDeviceObjectType	"Device"	"Devi"	DEVICE_OBJECT
IoDriverObjectType	"Driver"	"Driv"	DRIVER_OBJECT
IoFileObjectType	"File"	"File"	FILE_OBJECT
LpcPortObjectType	"Port"	"Port"	LPCP_PORT_OBJECT
LpcWaitablePortObjectType	"WaitablePort"	"Wait"	
ObpDirectoryObjectType	"Directory"	"Dire"	OBJECT_DIRECTORY
ObpSymbolicLinkObjectType	"SymbolicLink"	"Symb"	
ObpTypeObjectType	"Type"	"ObjT"	OBJECT_TYPE
PopPowerRequestObjectType			
PsJobType	"Job"	"Job "	EJOB
PsProcessType	"Process"	"Proc"	EPROCESS
PsThreadType	"Thread"	"Thre"	ETHREAD
SeTokenObjectType	"Token"	"Toke"	TOKEN
TmEnlistmentObjectType		"TmEn"	KENLISTMENT
TmResourceManagerObjectType		"TmRm"	KRESOURCEMANAGER
TmTransactionObjectType		"TmTm"	KTM
MmSectionObjectType	"Section"	"Sect"	SECTION
WmipGuidObjectType	"WmiGuid"	"WmiG"	GUID
		"TmTx"	KTRANSACTION

Vous pouvez consulter la liste des objets type déclarés au gestionnaire d’objets au moyen de la commande ! object du débogueur noyau.

```
lkd> !object \ObjectTypes
Object: ffffe78a17a15c50  Type: (ffff910a1d482570) Directory
    ObjectHeader: ffffe78a17a15c20 (new version)
    HandleCount: 0  PointerCount: 66
    Directory Object: ffffe78a17a18ea0  Name: ObjectTypes

    Hash Address          Type                     Name
    ---- -------          ----                     ----
     00  ffff910a1d516f20 Type                     TmTm
     01  ffff910a1d47abe0 Type                     Desktop
         ffff910a1d48d4b0 Type                     Process
     02  ffff910a1d595a80 Type                     EnergyTracker
         ffff910a1d517330 Type                     RegistryTransaction
     03  ffff910a1d47a1e0 Type                     DebugObject
     04  ffff910a1e6e0080 Type                     VRegConfigurationContext
         ffff910a1da34080 Type                     DmaDomain
         ffff910a1d478b80 Type                     TpWorkerFactory
     05  ffff910a1d47c4f0 Type                     Adapter
```

```
    ffff910a1d482080 Type                  Token
06  ffff910a1e6342e0 Type                  DxgkSharedResource
07  ffff910a1d494dc0 Type                  PsSiloContextPaged
08  ffff910a1e561d20 Type                  NdisCmState
    ffff910a1d494f20 Type                  ActivityReference
09  ffff910a1e3378c0 Type                  PcwObject
    ffff910a1d5a7320 Type                  WmiGuid
11  ffff910a1d464460 Type                  DmaAdapter
    ffff910a1d5a6dc0 Type                  EtwRegistration
12  ffff910a1d9d9f20 Type                  DxgkSharedBundleObject
    ffff910a1d51ddc0 Type                  Session
    ffff910a1d478e40 Type                  RawInputManager
    ffff910a1d479470 Type                  Timer
13  ffff910a1d476c80 Type                  Mutant
14  ffff910a1d48db80 Type                  IRTimer
16  ffff910a1d9d8b00 Type                  DxgkCurrentDxgProcessObject
    ffff910a1d47ed50 Type                  IoCompletion
17  ffff910a1d9d89a0 Type                  DxgkSharedProtectedSessionObject
    ffff910a1d9d8f20 Type                  DxgkSharedSyncObject
    ffff910a1d46d390 Type                  WindowStation
    ffff910a1d48da20 Type                  Profile
18  ffff910a1d50ef20 Type                  File
20  ffff910a1d48d080 Type                  Partition
21  ffff910a1d4795d0 Type                  Semaphore
22  ffff910a1d47a080 Type                  PsSiloContextNonPaged
23  ffff910a1d5a6b00 Type                  EtwConsumer
    ffff910a1d47aa80 Type                  Composition
24  ffff910a1d5a6c60 Type                  EtwSessionDemuxEntry
    ffff910a1d478ce0 Type                  CoreMessaging
25  ffff910a1d516dc0 Type                  TmTx
    ffff910a1d469f20 Type                  SymbolicLink
26  ffff910a1e428210 Type                  FilterConnectionPort
    ffff910a1d517490 Type                  Key
    ffff910a1d46d4f0 Type                  KeyedEvent
    ffff910a1d482a00 Type                  Callback
27  ffff910a1d47ebf0 Type                  WaitCompletionPacket
28  ffff910a1d46a8f0 Type                  UserApcReserve
    ffff910a1d46f9c0 Type                  Job
29  ffff910a1d9d8c60 Type                  DxgkDisplayManagerObject
    ffff910a1d9d8dc0 Type                  DxgkSharedSwapChainObject
    ffff910a1d47c390 Type                  Controller
    ffff910a1d46a790 Type                  IoCompletionReserve
30  ffff910a1d47c230 Type                  Device
    ffff910a1d482570 Type                  Directory
31  ffff910a1d51df20 Type                  Section
    ffff910a1d516b00 Type                  TmEn
    ffff910a1d48dce0 Type                  Thread
32  ffff910a1d47f600 Type                  Type
33  ffff910a1e429f20 Type                  FilterCommunicationPort
    ffff910a1d594b50 Type                  PowerRequest
35  ffff910a1d516c60 Type                  TmRm
    ffff910a1d46ea30 Type                  Event
36  ffff910a1d596b40 Type                  ALPC Port
    ffff910a1d47eeb0 Type                  Driver
```

Comptabilisation des handles ouverts pour chaque type d'objet

Pour voir combien d'objets de chaque type ont été ouverts, utilisez la commande Handle en spécifiant le commutateur -s. Vous devriez alors voir quelque chose qui ressemble à ce qui suit :

```
C:\>handle.exe -s

Nthandle v4.22 - Handle viewer
Copyright (C) 1997-2019 Mark Russinovich
```

```
Sysinternals - www.sysinternals.com

Handle type summary:
  ALPC Port       : 3478
  Composition     : 129
  CoreMessaging   : 17
  Desktop         : 247
  Directory       : 520
  DxgkDisplayManagerObject: 1
  DxgkSharedResource: 38
  DxgkSharedSyncObject: 7
  EnergyTracker   : 1
  EtwConsumer     : 10
  EtwRegistration : 21066
  Event           : 25476
  File            : 4739
  FilterCommunicationPort: 6
  FilterConnectionPort: 9
  IoCompletion    : 2632
  IoCompletionReserve: 87
  IRTimer         : 2334
  Job             : 291
  Key             : 11423
  Mutant          : 1609
  Partition       : 3
  PcwObject       : 20
  PowerRequest    : 10
  Process         : 2484
  RawInputManager : 36
  Section         : 3235
  Semaphore       : 11539
  Session         : 28
  SymbolicLink    : 250
  Thread          : 3874
  Timer           : 719
  TmRm            : 20
  TmTm            : 10
  Token           : 5563
  TpWorkerFactory : 1153
  UserApcReserve  : 14
  WaitCompletionPacket: 7833
  WindowStation   : 380
  WmiGuid         : 37
Total handles: 111328
```

Pour limiter la liste aux handles ouverts par un processus spécifique, ajoutez l’option -p suivie du nom (complet ou partiel) ou de l’ID d’un processus. L’exemple ci-après montre le résultat de la commande Handle, interrogée en ce concerne les processus de l’Explorateur Windows (explorer.exe).

```
C:\>handle.exe -s -p explorer

Nthandle v4.22 - Handle viewer
Copyright (C) 1997-2019 Mark Russinovich
Sysinternals - www.sysinternals.com

Handle type summary:
  ALPC Port       : 118
  Desktop         : 1
  Directory       : 2
  EtwRegistration : 689
  Event           : 2143
  File            : 186
  IoCompletion    : 316
  IoCompletionReserve: 11
  IRTimer         : 538
```

```
  Key             : 624
  Mutant          : 217
  Process         : 7
  Section         : 95
  Semaphore       : 100
  Thread          : 171
  Timer           : 25
  Token           : 4
  TpWorkerFactory : 269
  UserApcReserve  : 3
  WaitCompletionPacket: 1257
  WindowStation   : 2
  WmiGuid         : 1
Total handles: 6779
```

Méthodes d'objet

Les objets sont pour ce qui concerne leur manipulation traités par l'intermédiaire d'un ensemble de services génériques, communément appelées *méthodes*, que le gestionnaire d'objets anime en quelques occasions ciblées, habituellement quand il y a création, suppression ou modification d'un objet.

Quand un composant de l'exécutif donne lieu à un type d'objet, il peut déclarer des méthodes auprès du gestionnaire d'objets. Certains types d'objet spécifient l'ensemble des implémentations potentielles (comprendre que toutes les adresses des fonctions virtuelles sont renseignées), et d'autres seulement une partie, selon que telle ou telle opération ai un sens dans le contexte.

Le tableau suivant dresse la liste des méthodes prises en charge par le gestionnaire d'objets.

Tableau 6.7. Méthodes d'objet

Méthode	Moment de l'appel
Open	Quand le gestionnaire d'objets crée un handle d'objet
Close	Quand le gestionnaire d'objets ferme un handle d'objet
Delete	Quand le gestionnaire d'objets supprime un objet
Query Name	Quand un thread demande le nom d'un objet qui existe dans un espace de noms d'objets secondaires
Parse	Quand le gestionnaire d'objets recherche un nom d'objet qui existe dans un espace de noms d'objets secondaires
Security	Quand un processus lit ou modifie la protection d'un objet qui existe dans un espace de noms d'objets secondaires

Le gestionnaire d'objet appelle la méthode open chaque fois qu'il crée un handle pour un objet, ce qu'il fait lors de la création ou de l'ouverture d'un objet. Les objets du type processus, threads, fichiers, et quelques autres, montrent un exemple d'utilisation de méthode open.

Lorsque le handle associé à un objet est refermé, la procédure close correspondant à l'objet est appelée. Par exemple, le gestionnaire d'E/S déclare une méthode close pour le type d'objet fichier, et le gestionnaire d'objets sollicite ladite procédure chaque fois qu'il ferme un handle d'objet fichier. En règle générale, cette démarche donne le départ à un certain nombre d'opérations de nettoyage dont l'exécution entraine la suppression de tout état spécifique à l'utilisation de l'objet par l'appelant, y compris les verrous en instance et les restrictions de partage. Le gestionnaire d'objets n'ayant pas de visibilité sur la façon dont un processus emploie tel ou tel objet, il ne peut dès lors procéder lui-même à de tels travaux.

Le gestionnaire d'objets appelle une méthode delete, s'il y en a une de déclarée, avant de supprimer un objet temporaire en mémoire, créant de cette manière l'occasion (du reste, la dernière) de libérer les ressources qui ne l'ont pas été jusqu'à présent, par exemple les structures de données internes indispensables au cycle de vie de l'objet. Si l'on considère une nouvelle fois les objets fichiers comme exemple, le gestionnaire d'E/S déclare pour le type d'objet fichier une méthode delete dont l'appel entraine l'envoi d'une notification à l'objet périphérique représentant le système de système de fichiers. Le gestionnaire de mémoire, autre exemple, met en oeuvre pour le type d'objet section une méthode delete qui libère les pages physiques assignées à la section. Là encore, le

gestionnaire d'objets ne peut mener à bien ce travail sans un partenaire spécialisé. Les méthodes delete des autres types d'objets remplissent des fonctions semblables. Lorsque la procédure delete d'un objet temporaire se termine, l'objet disparait tout à fait.

Les opérations qui relèvent de la méthode parse ont lieu quand le gestionnaire d'objets recherche un objet qui existe dans un espace de noms secondaire. Quand le gestionnaire d'objets recherche un nom d'objet, il interrompt sa recherche quand il rencontre un objet du chemin qui a une méthode parse associée. Le gestionnaire d'objets appelle la méthode parse en lui passant le reste du nom de l'objet qu'il cherche à retrouver. Il existe deux espaces de noms dans Windows, en plus de celui du gestionnaire d'objets : l'espace de noms du registre, implémenté par le gestionnaire de configuration, et l'espace de noms du système de fichiers, que le gestionnaire d'E/S met en oeuvre avec l'aide des pilotes de système de fichiers.

La méthode query name est employée quand quand le gestionnaire d'objets recherche un nom d'objet existant dans un espace de noms secondaire.

Employée comme niveau supplémentaire de protection contre l'usage indu (ou incorrect) des compteurs que le gestionnaire d'objet maintient pour chaque objet, la méthode okay to close vient s'ajouter aux contrôles habituels effectués au préalable du démantèlement des ressources, et est l'occurrence mise en oeuvre chaque fois que la suppression d'une référence au compteur de références d'un objet serait susceptible d´entrer en conflit avec une stratégie système. La partie mode noyau du sous-système Windows (Win32k.sys) déclare par exemple une méthode okay to close pour les objets de type bureau et station-fenêtre, de sorte qu'aucun thread ayant des fenêtres visibles ne soit orphelin à ce niveau.

Vous pouvez observer à l'intérieur du débogueur noyau quelles fonctions virtuelles sont définies pour quel type d'objet. Commencez par repérer un type d'objet, ici par exemple le type objet Processus.

```
lkd> dt nt!_OBJECT_TYPE poi(nt!ObTypeIndexTable + 7 * @$ptrsize)
   +0x000 TypeList         : _LIST_ENTRY [ 0xfffffa80`0245db70 - 0xfffffa80`0245db70 ]
   +0x010 Name             : _UNICODE_STRING "Process"
   . . .
   +0x040 TypeInfo         : _OBJECT_TYPE_INITIALIZER

lkd> dt nt!_OBJECT_TYPE Name TypeInfo. poi(nt!ObTypeIndexTable + 7 * @$ptrsize)
   +0x010 Name      : _UNICODE_STRING "Process"
   +0x040 TypeInfo  :
      +0x000 Length    : 0x78
      . . .
      +0x030 DumpProcedure : (null)
      +0x038 OpenProcedure : 0xfffff802`79ff6320        long  nt!PspProcessOpen+0
      +0x040 CloseProcedure : 0xfffff802`7a01438c        void  nt!PspProcessClose+0
      +0x048 DeleteProcedure : 0xfffff802`7a02e780        void  nt!PspProcessDelete+0
      +0x050 ParseProcedure : (null)
      +0x058 SecurityProcedure : 0xfffff802`7a07c880        long  nt!SeDefaultObjectMethod+0
      +0x060 QueryNameProcedure : (null)
      +0x068 OkayToCloseProcedure : (null)
```

Noms des objets

Tous les systèmes d'exploitation offrent la possibilité de nommer certaines des ressources vers lesquelles ils fournissent un tremplin. Si l'on pense à cet égard instinctivement aux fichiers et aux répertoires, il faut aussi considérer les disques, les volumes, les tubes, les blocs de mémoire partagée, et quelques autres. L'exécutif Windows, en comparaison, permet de donner un nom à toute ressource représentée par un objet. Les noms d'objet permettent de répondre à plusieurs exigences fondamentales concernant la localisation et l'accès aux objets, mais aussi le partage d'objets entre processus.

Les noms d'objets permettent aux processus de se partager des objets. L'espace de noms des objets de l'exécutif est visible à tous les processus du système. Un processus peut dès lors créer un objet en lui conférant un nom et un autre processus ouvrir un handle vers l'objet en spécifiant le nom de l'objet. Un objet, s'il n'est pas censé être partagé (du moins pas de cette manière), peut ne pas avoir de nom.

Résoudre un nom peut sur le plan des performances s'avérer une procédure coûteuse, si ce n'est en temps en nombre d'opérations. De ce fait, pour plus d'efficacité, le gestionnaire d'objets ne recherche pas le nom d'un objet chaque fois qu'un composant utilise l'objet. En réalité, le gestionnaire d'objets ne recherche des noms que dans deux cas : création d'un objet nommé et création d'un handle vers un objet nommé. Quand un processus crée un objet nommé, le gestionnaire d'objets recherche le nom pour vérifier qu'il n'existe pas déjà et, le cas échéant, stocke le nouveau nom dans l'espace de noms global. Quand un processus ouvre un handle vers un objet nommé, le gestionnaire d'objets recherche le nom, trouve l'objet, puis retourne un handle à l'appelant. Toutes les opérations subséquentes sont réalisées à l'aide du handle.

Quand il recherche un nom, le gestionnaire d'objets permet à l'appelant de choisir entre recherche tenant compte de la casse et recherche ne tenant pas compte de la casse. Cette fonctionnalité assure la compatibilité avec POSIX et autres environnements sensibles sur ces aspects.

Où sont stockés les noms des objets dans l'espace de noms dépend du type auquel appartient l'objet. Le tableau suivant énumère les répertoires d'objets standard de Windows, en indiquant les types d'objets dont les noms sont enregistrées dans ces répértoires.

Tableau 6.8. Répertoires d'objets standard

\GLOBAL??	Liens symboliques vers les noms internes des périphériques sous Windows, y compris les lettres de lecteurs définies globalement et les noms de périphériques MS-DOS, tels que CON, PRN, AUX et NUL.
\Callback	Objets callback.
\Device	Objets périphérique.
\Driver	Objets pilote.
\FileSystem	Objets pilote de système de fichiers et objets périphérique reconnaisseur de système de fichier (*file system recognizer*)
\RPC Control	Objets port (ALPC) employés lors des mécanismes RPC.
\KnownDLLs et \KnownDLLs32	Noms de section et chemin pour les DLL connues (DLL chargées par le système au démarrage). \KnownDLLs32 existe uniquement sur les versions 64 bits de Windows et énumère les versions 32 bits des DLLs connues.
\Nls	Noms de section pour les tables mappées de support de langue nationale (NLS).
\ObjectTypes	Noms des types d'objets.
\Security	Objets spécifiques au sous-système de sécurité.
\Sessions	Données privées des différentes sessions maintenues sur la station de travail.

Les objets noyau de base, tels les événements, sémaphores, mutex, minuteries attendables (waitable timers) et sections, ont leurs noms enregistrés dans un même répertoire d'objets, en conséquence de quoi deux objets ne peuvent pas avoir le même nom même s'ils ne sont pas du même type. Cet aspect restrictif implique de bien penser les noms, de façon qu'ils n'entrent pas en collision les uns avec les autres.

Tableau 6.9. Routines connexes aux noms d'objet

Routine	Description
ObOpenObjectByName	Ouvre par le nom un objet.
ObQueryNameString	Retourne s'il en est pourvu le nom d'un objet.
ObReferenceObjectByName	Référence par le nom un objet.

Objets de base nommés

Voici quelques exemples d'objets de base nommés :

- *DBWinMutex* Contrôle l'exclusion mutuelle des opérations liées à la sortie de débogage.

- *SC_AutoStartComplete* Objet événement utilisé par le SCM de sorte à indiquer quand il a fini de démarrer tous les services et pilotes automatiques.

WinObj

L'utilitaire WinObj permet l'exploration de l'espace de noms interne du gestionnaire d'objets. Que vous soyez un administrateur scrupuleusement attentif du fonctionnement de ses systèmes, un concepteur de logiciel en prise avec des problèmes liés aux objets, ou un utilisateur simplement préoccupé de mieux connaitre les diverses ressources physiques et logiques gérées par Windows, WinObj se révèlera sans nul doute un outil particulièrement efficace, utilisable qui plus est à de nombreuses fins. Il est disponible gratuitement sur le site Microsoft TechNet.

Sur le plan fonctionnel, WinObj repose sur l'emploi d'un petit nombre de services appelables depuis le mode utilisateur, dont NtQuerySystemInformation. Il n'existe autrement dit pas de pilote tiers associé à l'application, laquelle s'exécute en l'occurrence comme n'importe quel autre logiciel prévu pour Windows.

WinObj procure une vue graphique sur la structure générale où sont rangés les objets Windows, elle-même organisée à la façon d'un système de fichiers. Lors du démarrage de l'application, celle-ci commence par ouvrir une fenêtre qui dévoile le niveau racine et les entrées de premier niveau de l'espace de noms du gestionnaire d'objets. Lorsque vous sélectionnez un répertoire dans le volet gauche, le volet de droite répertorie les objets contenus dans ce répertoire. Lorsque vous sélectionnez un répertoire dans volet de gauche ou un objet dans le volet de droite, la barre d'état indique le chemin complet de l'élément. Vous pouvez actualiser l'affichage à tout moment en appuyant sur F5.

Pour voir plus d'informations sur un répertoire ou un objet en particulier, effectuez un clic droit et choisissez Propriétés. Double-cliquer sur un objet permet également de faire apparaitre la boite de dialogue des attributs le concernant, sauf s'il s'agit d'un lien symbolique : le double-clic sur un lien déclenche la navigation vers la cible de celui-ci.

L'onglet Détails de la boîte de dialogue Propriétés affiche pour tout objet les attributs que voici.

- Le nom de l'objet et le type auquel il appartient.
- Si l'objet a un caractère permanent.
- Compteur de références et compteur de handles
- Quota facturés

La partie inférieure de la fenêtre énumère des valeurs pour des caractéristiques qui sont spécifiques à la nature de l'objet. Dans le cas d'un lien symbolique, par exemple, WinObj affiche des informations temporelles associées à la création de l'objet, de même bien entendu que les données de chemin auxquelles ledit lien fait référence. Si l'objet en question est de type événement, les données incluent la fonction de réinitialisation (automatique ou manuelle) et l'état (signalé ou non) de l'objet. En définitive, les informations présentées à ce niveau dépendent du contexte, et changent selon quelle place a un objet dans le système d'exploitation.

L'onglet Sécurité de la boîte de dialogue Propriétés liste les paramètres d'autorisation concernant un objet. Notez à cet égard que les autorisations en vigueur sur un objet peuvent empêcher l'affichage ou la modification de ses propriétés.

Afin d'aider les utilisateurs à mieux se repérer parmi toutes les informations de l'espace de noms, WinObj fait usage d'une signalétique où les icônes servent à mettre en évidence le type de n'importe quel objet.

- Les mutex sont dotés d'une icône qui ressemble à un cadenas.
- Les sections (objets mappage de fichier) sont dotées d'une puce mémoire.
- Les sémaphores sont munis d'une icône qui ressemble à des feux de signalisation.

- Les événements sont dotés d'un point d'exclamation à l'intérieur d'un triangle jaune.
- Les événements à clé ont la même icône que des événements standards, avec le dessin d'une clé en superposition.
- Les liens symboliques sont indiqués par une flèche incurvée.
- Les minuteries sont symbolisées par une horloge.
- Les stations fenêtre sont représentés par un moniteur d'ordinateur.
- Les pilotes sont représentés par une icône en forme d'engrenage. La même illustration est par ailleurs utilisée dans d'autres contextes, par exemple les objets de type port LPC ou groupes de processus (jobs).

Objet lien symbolique

Les objets lien symbolique implémentés par le gestionnaire d'objets de Windows servent à organiser l'espace de noms. Ils sont également utilisés pour associer un nom standard MS-DOS (a :, c :, etc.) aux lecteurs, ainsi que pour assurer la visibilité des périphériques aux applications (qui ne connaissent pas les noms internes des périphériques).

Un lien symbolique peut figurer n'importe où dans une chaîne de noms d'objet, charge au Gestionnaire d'Objets de le résoudre, et transformer auquel cas un nom abstrait en un autre nom, plus proche des racines du système. Quand un processus fait référence au nom d'un objet lien symbolique, le Gestionnaire d'Objets parcourt son espace de noms jusqu'à rencontrer (s'il s'y trouve) l'objet en question. Il se met alors en quête de la cible associée (représentée par le champ nommé LinkTarget dans la structure OBJECT_SYMBOLIC_LINK) et trouve une chaîne qu'il substitue au nom du lien symbolique. Après quoi il reprend sa recherche de nom, finalisée qu'au moment où toutes les références concernant des objets lien symbolique auront été résolues.

Tous les utilisateurs de Windows, sans le savoir peut-être, emploient des liens symboliques, habitués d'un certain style pour la désignation des lecteurs et autres périphériques (A:, B:, C:, etc. dans les volumes disque, COM1, COM2, etc. dans les ports série, etc.). En interne de ces associations se trouve le Gestionnaire d'Objets, chargé de la traduction des noms de périphérique tels qu'exprimés en standard MS-DOS vers leurs pendants dans l'espace de noms.

D'ordinaire, un lien symbolique est un objet du système de fichiers qui pointe vers un autre objet du système de fichiers. L'objet pointé est appelé la cible. Dans Windows, cette définition est à nuancer, puisque l'espace de noms est abstrait, et aussi parce que le Gestionnaire d'Objets est conçu pour faire peu de cas de la structure interne du système de fichiers. A l'inverse d'UNIX, qui enracine l'espace de noms système dans le système de fichiers, Windows emploie un espace de noms abstraits, et y connecte les systèmes de fichiers comme des périphériques.

Compte tenu des dangers que serait susceptible d'entraîner une mauvaise utilisation des liens symboliques, la création de tels objets est sévèrement controlé, et nécessite en l'occurence impérativement le privilège SeCreateSymbolicLink, attribué uniquement aux administrateurs. Le système de fichiers intègre en plus de cela une option appelée SymLinkEvaluation qui peut être configurée par l'intermédiaire de la commande fsutil.

Tableau 6.10. Services concernant les liens symboliques

Service	Description
NtCreateSymbolicLinkObject	Crée, ou éventuellement ouvre, un objet lien symbolique.
NtOpenSymbolicLinkObject	Ouvre un objet lien symbolique.
NtQuerySymbolicLinkObject	Retourne la cible d'un lien symbolique.

Tableau 6.11. Routines concernant les liens symboliques

Routine	Description

ObpCreateSymbolicLinkName	-
ObpDeleteSymbolicLink	-
ObpDeleteSymbolicLinkName	-
ObpParseSymbolicLink	-
ObpProcessDosDeviceSymbolicLink	-

OBJECT_SYMBOLIC_LINK.

```
lkd> dt nt!_OBJECT_SYMBOLIC_LINK
   +0x000 CreationTime     : _LARGE_INTEGER
   +0x008 LinkTarget       : _UNICODE_STRING
   +0x008 Callback         : Ptr64     long
   +0x010 CallbackContext  : Ptr64 Void
   +0x018 DosDeviceDriveIndex : Uint4B
   +0x01c Flags            : Uint4B
   +0x020 AccessMask       : Uint4B
```

- **Horodatage de création (*CreationTime*)** Date et heure à laquelle le lien symbolique a été créé. Voir service NtCreateSymbolicLinkObject.
- **Cible (*LinkTarget*)** Chemin réel ou l'emplacement vers lequel le lien symbolique pointe.

Heure de création de liens symbolique. Le service NtCreateSymbolicLink, à l'issue des opérations qui lui sont confiées, horodate automatiquement tout objet auquel il est parvenu à donner lieu. Les informations recueillies à cet égard le sont en interne par le biais d'un appel à la routine KeQuerySystemTime, et après quoi rendus visibles au moyen du champ CreationTime de la structure OBJECT_SYMBOLIC_LINK. Pour voir à quelle date et quelle heure un lien symbolique a été créé, tapez dt nt!_OBJECT_SYMBOLIC_LINK CreationTime dans le débogueur noyau, suivi de l'adresse d'un quelconque objet de ce type.

```
lkd> !object \DosDevices
Object: ffffba8a22215640  Type: (ffff930b8ba69f20) SymbolicLink
    ObjectHeader: ffffba8a22215610 (new version)
    HandleCount: 0  PointerCount: 1
    Directory Object: ffffba8a22218de0  Name: DosDevices
    Flags: 00000000 ( Local )
    Target String is '\??'

lkd> dt nt!_OBJECT_SYMBOLIC_LINK CreationTime ffffba8a22215640
   +0x000 CreationTime : _LARGE_INTEGER 0x01d3be56`629f8d63

lkd> .formats 0x01d3be56`629f8d63
Evaluate expression:
  Hex:     01d3be56`629f8d63
  Decimal: 131658092354964835
  Octal:   00072357453142477065 43
  Binary:  00000001 11010011 10111110 01010110 01100010 10011111 10001101 01100011
  Chars:   ...Vb..c
  *Time:    Sun Mar 18 02:13:55.496 2018 (UTC + 1:00)*
  Float:   low 1.47161e+021 high 7.77823e-038
  Double:  7.37036e-300
```

Répertoires d'objet

L'objet répertoire d'objets est le moyen par lequel le Gestionnaire d'Objets organise l'espace de noms, et agence avec lui une structure de noms hiérarchique analogue à celles de systèmes de fichiers. L'objet contient les noms d'autres objets, voire d'autres répertoires d'objets, qui mis en cascade permettent de structurer les noms d'objet tels des chemins d'accès MS-DOS.

Les composants de l'exécutif, pilotes de périphériques et les sous-systèmes font s'accroître l'espace de noms des objets. Par exemple, le gestionnaire d'E/S crée un répertoire d'objets nommé Device qui contient les noms

des objets créés par les pilotes pour représenter les périphériques d'E/S. Ce qui suit énumère les répertoires d'objets de niveau supérieur susceptibles d'être utilisé par les pilotes, les autres étant normalement réservé au système d'exploitation :

- \Callbacks
- \Device
- \KernelObjects
- \DosDevices

Tableau 6.12. Services connexes aux répertoires d'objets

Service	Description
NtCreateDirectoryObject	Crée un répertoire d'objets.
NtQueryDirectoryObject	Retourne des informations concernant un répertoire d'objets donné.
NtOpenDirectoryObject	Crée, ou éventuellement ouvre, un répertoire d'objets.

OBJECT_DIRECTORY.

```
lkd> dt nt!_OBJECT_DIRECTORY
   +0x000 HashBuckets      : [37] Ptr64 _OBJECT_DIRECTORY_ENTRY
   +0x128 Lock             : _EX_PUSH_LOCK
   +0x130 DeviceMap        : Ptr64 _DEVICE_MAP
   +0x138 ShadowDirectory  : Ptr64 _OBJECT_DIRECTORY
   +0x140 NamespaceEntry   : Ptr64 Void
   +0x148 SessionObject    : Ptr64 Void
   +0x150 Flags            : Uint4B
   +0x154 SessionId        : Uint4B
```

Tableau 6.13. Autorisations spécifiques à l'objet répertoire

Constante	Valeur	Description
DIRECTORY_QUERY	0x0001	Controlé par NtQueryDirectoryObject.
DIRECTORY_TRAVERSE	0x0002	
DIRECTORY_CREATE_OBJECT	0x0004	
DIRECTORY_CREATE_SUBDIRECTORY	0x0008	

Objets temporaires et objets permanents

Le gestionnaire d'objets de Windows donne lieu à deux sortes d'objet : ceux qui sont temporaires et ceux qui sont permanents. La plupart des objets sont temporaires, et disparaissent par conséquent quand plus aucun thread ne les utilise. Lorsque la procédure delete d'un objet temporaire se termine, le compteur de référence de l'objet chute à la valeur nulle (zéro), moment pour le Gestionnaire de supprimer l'objet de la mémoire. Un objet temporaire peut être rendu permanent en demandant au Gestionnaire d'Objets de compter une référence supplémentaire sur l'objet. Les objets permanents ne sont donc pas supprimés lorsque la dernière référence hors du Gestionnaire d'Objets est supprimée.

Le fait qu'un objet soit temporaire ou permanent est mutuellement exclusif : un objet est temporaire s'il n'est pas permanent (s'il n'a pas été marqué comme tel). Lorsqu'un objet permanent est rendu temporaire de nouveau, le Gestionnaire d'Objets supprime la référence supplémentaire, signe que plus aucun processus n'estime l'objet nécessaire, en conséquence de quoi il faille le démanteler. Les objets permanents sont rares, principalement utilisés pour les périphériques, les objets répertoires et liens symboliques.

Le service natif NtMakeTemporaryObject et la routine ObMakeTemporaryObject modifient les attributs d'un objet afin de le rendre temporaire.

Le tableau suivant répertorie les caractéristiques se rapportant au caractère temporaire ou permanent d'un objet.

Elément	Lieu	Type	Autres références
PermanentObject	OBJECT_HEADER	-	ObMakeTemporaryObject, OB_-FLAG_PERMANENT_OBJECT

Partage d'objets entre processus

Windows offre trois méthode de partage des objets entre processus. La première consiste pour un processus à hériter d'un handle d'objet, lequel fait éventuellement partie des caractéristiques que se transmettent les processus d'une même lignée (processus père/fils). Quand un processus fait appel à la fonction CreateXxx, il le fait en indiquant au système (plus précisément, au gestionnaire d'objets) quelles sont les règles à adopter en la matière, à quoi correspond une structure SECURITIES_ATTRIBUTES, dont le champ bInheritHandle indique s'il est à TRUE que le handle est héritable. Un processus peut alors en créer un autre en donnant la valeur TRUE au paramètre bInheritHandle de la fonction CreateProcess. Une autre manière de partager des objets est de donner un nom à l'objet lors de sa création, ce qui entraine de le rendre visible au sein de l'espace de noms global. L'avantage des objets nommés est que des processus sans aucun lien peuvent aisément se les partager. Dans ce scénario, le premier processus à animer un objet (CreateXxx) le fait en spécifiant lors de l'appel un nom dans le paramètre lpszName. Un autre processus a par la suite la possibilité d'obtenir une référence à cet objet en appelant la fonction OpenXxx (éventuellement CreateXxx) avec le même nom.

La troisième méthode de partage des objets repose sur la fonction DuplicateHandle, laquelle confère au processus appelant un duplicata d'un handle existant dans un autre processus. Par nature, vu que deux processus ne connaissent pas les valeurs des handles que l'un et l'autre utilisent, cette méthode requiert l'utilisation de procédé de communication inter processus comme la mémoire partagée, les messages, ou d'autres. En partant du handle d'un processus et du handle d'un objet manipulé par ce processus, un autre processus peut dans se contexte se voir octroyer un handle menant audit objet.

Handles d'objet et table des handles de processus

Quand un processus crée ou ouvre un objet par le nom, il reçoit un identificateur (*handle*) qui représente son accès à l'objet. Référencer un objet par son handle est plus rapide que d'utiliser le nom de l'objet, cela dans la mesure où le gestionnaire d'objets peut de la sorte localiser un objet sans effectuer de recherche de nom. Les processus peuvent aussi acquérir des handles vers des objets en héritant des handles de leur parent ou en recevant un duplication de handle d'un autre processus.

Tous les processus mode utilisateur doivent au préalable des interactions avec un objet en particulier posséder un handle vers ledit objet, sans quoi leurs threads ne peuvent y accéder. Dans la perspective de la prise en charge Windows de la sécurité, le principal avantage des handles est qu'ils servent de références indirectes vers les ressources système, et empêchent ainsi les applications de voir et de manipuler les structures de données sous-jacentes.

Les composants de l'exécutif, pilotes et autres codes système ne sont pas soumis aux mêmes obligations que les processus mode utilisateur en ce qui concerne les handles, mais peuvent en l'occurence accéder directement aux objets en mémoire. Si tel est le cas, cependant, ils doivent rendre compte de l'emploi de l'objet en incrémentant le compteur de références de façon que l'objet ne puisse pas être démantelé tandis qu'il est en cours d'utilisation.

Les handles d'objet procurent aussi d'autres avantages. Premièrement, exception faite de ce à quoi ils font référence, il n'y a aucune différence entre un handle de fichier, un handle de processus, un handle d'événements, et bien d'autres. Ces similitudes offrent par là même une interface cohérente pour référencer les objets indépendamment de leur type. Un second bénéfice garanti par les handles découle directement du fait que le gestionnaire d'objets soit le seul composant habilité à donner lieu à de telles références, et peut alors contrôler toute action mode utilisateur qui affecte un objet en vérifiant que le profil de sécurité de l'appelant autorise bel et bien l'opération demandée.

En interne, un handle d'objet est un index dans une *table de handles* propre à un processus, mappée dans l'espace d'adressage virtuel du noyau, allouée à partir du pool paginé et vers laquelle pointe le bloc EPROCESS. La table des handles d'un processus contient des pointeurs vers tous les objets pour lesquels le processus a ouvert un handle.

Chaque entrée de la sein de la table des handles d'un processus se présente sous la forme d'une structure HANDLE_TABLE_ENTRY.

- *Pointeur d'objet (Object)* Pointeur vers l'entête de l'objet pour lequel le handle a été créé.
- *Masque des accès autorisés (GrantedAccess)* Détermine les diverses opérations rendues possibles par le biais du handle.

Quand un processus se termine, le système appelle la routine ExSweepHandleTable de façon à refermer tous les handles vers des objets qui ont été ouvert lors de l'exécution.

```
lkd> dt nt!_HANDLE_TABLE_ENTRY ffffc48415a24010
   +0x000 VolatileLowValue : 0n-2698184060677586945
   +0x000 LowValue         : 0n-2698184060677586945
   +0x000 InfoTable        : 0xda8e1fe4`74b0ffff _HANDLE_TABLE_ENTRY_INFO
   +0x008 HighValue        : 0n2031619
   +0x008 NextFreeHandleEntry : 0x00000000`001f0003 _HANDLE_TABLE_ENTRY
   +0x008 LeafHandleValue  : _EXHANDLE
   +0x000 RefCountField    : 0n-2698184060677586945
   +0x000 Unlocked         : 0y1
   +0x000 RefCnt           : 0y0111111111111111 (0x7fff)
   +0x000 Attributes       : 0y000
   +0x000 ObjectPointerBits : 0y1101101010001110000111111110010001110100 1011 (0xda8e1fe474b)
   +0x008 GrantedAccessBits : 0y00001111100000000000000011 (0x1f0003)
   +0x008 NoRightsUpgrade  : 0y0
   +0x008 Spare1           : 0y000000 (0)
   +0x00c Spare2           : 0
```

Visualisation de la table des handles. La commande suivante affiche tous les détails de la table des handles du processus dont l'ID est 1110.

```
lkd> !handle 0 7 1110

PROCESS ffffda8e1ce04080
    SessionId: 1  Cid: 1110    Peb: 95f6a1a000  ParentCid: 0db4
    DirBase: 97bf8002  ObjectTable: ffffc484143f2e40  HandleCount: 521.
    Image: notepad.exe

Handle table at ffffc484143f2e40 with 521 entries in use

0004: Object: ffffda8e1fe02760  GrantedAccess: 001f0003 (Protected) (Inherit) Entry: ffffc48423af6010
Object: ffffda8e1fe02760  Type: (ffffda8e174a20c0) Event
    ObjectHeader: ffffda8e1fe02730 (new version)
        HandleCount: 1  PointerCount: 32768

0008: Object: ffffda8e1fe030e0  GrantedAccess: 001f0003 (Protected) (Inherit) Entry: ffffc48423af6020
Object: ffffda8e1fe030e0  Type: (ffffda8e174a20c0) Event
    ObjectHeader: ffffda8e1fe030b0 (new version)
        HandleCount: 1  PointerCount: 32769
.
.
.
```

Table des handles noyau

Les composants système, compte tenu des exigences de sécurité auxquelles ils doivent répondre, ont couramment besoin d'ouvrir des handles vers des objets auxquels les applications mode utilisateur ne doivent

pas avoir accès. Une telle configuration est en interne rendue possible par le biais de la *table des handles noyau*, laquelle structure renferme des handles dont la particularité est d'être accessibles, d'une part uniquement depuis le mode noyau, d'autre part dans n'importe quel contexte de processus.

Le gestionnaire d'objets reconnait les références aux handles de la table des handles noyau au fait que le bit de poids fort du handle est positionné, et ont incidemment des valeurs supérieures à 0x80000000.

```
lkd> ? poi(nt!ObpKernelHandleTable)
Evaluate expression: -65403497720448 = ffffc484`0fc05580

lkd> !process 0 0 System
PROCESS ffffda8e17470380
    SessionId: none  Cid: 0004    Peb: 00000000  ParentCid: 0000
    DirBase: 001ad002  ObjectTable: ffffc4840fc05580  HandleCount: 2789.
    Image: System

lkd> ? poi(nt!HandleTableListHead)
Evaluate expression: -65403497720424 = ffffc484`0fc05598
```

Démantèlement de handles

Il est possible à l'aide de la commande Handle de refermer les handles ouverts par un processus, de surcroit sans pour autant mettre fin à son exécution. Sachez toutefois qu'une telle façon de procéder n'est pas sans risque. Etant donné que le processus détenteur du handle n'a en cette occasion aucune connaissance du fait que ses accès à la ressource idoine ont été annulés, l'utilisation de cette fonctionnalité peut entrainer des dysfonctionnements plus ou moins sévères : immobilisation de l'application, corruption de données, etc. Avec cette mise en garde à l'esprit, procédez comme suit en vue de démanteler un handle donné :

```
handle -c handle -p pid [-y]
```

Avant toute chose, Handle affiche des informations concernant le handle, y compris son type et son nom, suivies d'une demande de confirmation quant à la suite à donner aux événements. Vous pouvez outrepasser cette sollicitation supplémentaire en ajoutant l'option -y sur la ligne de commande.

Pour finir, notez que certains handles d'objets sont de base protégées contre la fermeture. Les tentatives de refermer ces handles échouent silencieusement, ce qui implique que Handle indique que l'opération a eu lieu même si ce n'est pas le cas.

Rétention des objets

Le Gestionnaire d'Objets implémente la rétention d'objet, à savoir qu'il laisse perdurer les objets tant qu'ils sont utilisés, et cesse de les conserver sitôt qu'ils ne le sont plus. Ce genre de stratégie, qui vise essentiellement à optimiser l'occupation des ressources mémoire (centrale et auxiliaire), dérive du fait que le Gestionnaire piste les processus utilisateurs d'objets. Il sait instinctivement leur nombre, et peut rapidement connaître lesquels. Qu'un objet soit préservé ou démantelé dépend du nombre de références qui s'établissent sur lui.

Le Gestionnaire d'Objets garde trace de deux compteurs pour chaque objet. Le compteur de descripteurs est le nombre de handles ouverts sur l'objet. Chaque fois qu'un processus ouvre un handle vers un objet, le Gestionnaire d'Objets crée une entrée dans la table des handles du processus, insère le handle nouvellement crée dans cette table, et incrémente d'une unité le compteur de handles ouverts dans l'entête de l'objet. Quand un processus a fini d'utiliser l'objet et ferme son handle, cela supprime une référence du compteur de handles ouverts. Si le compteur atteint une valeur nulle (quand il tombe à zéro), c'est le moment pour le Gestionnaire de supprimer le nom éventuel de l'objet dans l'espace de noms global. Cette suppression de l'entrée de répertoire contenant le nom de l'objet empêche de nouveaux processus d'ouvrir des handles vers l'objet. À ce stade, seul le code du système d'exploitation peut accéder à l'objet. Comme les composants mode noyau accèdent plus volontiers aux objets à l'aide de pointeurs (et non de handles), le Gestionnaire doit également enregistrer le nombre des pointeurs d'objet qu'il a attribué aux processus système.

Le compteur de pointeurs est le nombre de références distinctes faites sur un objet, mis à jour chaque fois que le Gestionnaire d'Objets à fourni un pointeur vers l'objet (incrementation), et que des composants mode noyau ont fini d'utiliser le pointeur. Chaque descripteur est également comptabilisé par le compteur de références ; en effet, un handle est aussi une référence d'objet.

Au moment de la création de l'objet par la fonction ObCreateObject, le compteur de pointeurs est fixé à un pour tenir compte de la référence retournée à l'appelant. Si l'objet est nommé, le nombre est fixé à deux pour tenir compte du pointeur dans l'objet répertoire contenant le nom de l'objet.

Tableau 6.14. Eléments auxiliaires sous-jacents à la rétention des objets

Élément	Lieu	Type	Autres références
PointerCount	OBJECT_HEADER	LONG	-
HandleCount	OBJECT_HEADER	LONG	-

Sécurité des objets

Quand un processus ouvre un fichier, il doit le faire en précisant quelles options d'accès (lecture, écriture, etc.) il sollicite concernant le fichier en question. Dans la même veine, au sein de l'exécutif, quand un processus crée un objet ou ouvre un handle vers un objet existant, il doit spécifier un ensemble de droits d'accès souhaités (*desired acces rights*), lesquels représentent en définitive ce que le processus souhaite faire avec l'objet. Il peut demander soit un ensemble de droits d'accès d'accès standards (tels lire, écrire et exécuter) qui s'appliquent à tous les types d'objets, soit des droits d'accès spécifiques qui dépendent du type d'objet. Un processus peut par exemple demander des accès pour suppression sur un objet fichier, des accès pour suspension en ce qui concerne un objet thread, etc.

Quand un processus ouvre un handle vers un objet, le gestionnaire d'objets relaie au moniteur de références de sécurité l'ensemble des droits d'accès désirés par le processus. Le moniteur vérifie alors que le descripteur de sécurité de l'objet autorise le type d'accès demandé et, si tel est le cas, retourne un ensemble de droits d'accès accordés (granted access rights). Le gestionnaire d'objets stocke ces droits au sein du handle qu'il crée pour l'objet.

A partir du moment où un processus est dépositaire d'un handle se rapportant à un objet donné, le gestionnaire d'objets peut dès lors contrôler rapidement que l'ensemble des droits d'accès accordées qui est stocké dans le handle correspond à l'usage impliqué par le service d'objet auquel le processus tente d'accéder. Par exemple, si l'appelant s'est vu accordé un accès en lecture à une ressource mais qu'il sollicite un service pour écrire dans cet objet, ledit service échoue.

OBJECT_INFORMATION_CLASS

Tableau 6.15. OBJECT_INFORMATION_CLASS

ID	Classe	Type de données associé
0x0000	ObjectBasicInformation	OBJECT_BASIC_INFORMATION
0x0001	ObjectNameInformation	OBJECT_NAME_INFORMATION
0x0002	ObjectTypeInformation	OBJECT_TYPE_INFORMATION
0x0003	ObjectTypesInformation	OBJECT_TYPES_INFORMATION
0x0004	ObjectHandleFlagInformation	OBJECT_HANDLE_FLAG_INFORMATION
0x0005	ObjectSessionInformation	

OBJECT_BASIC_INFORMATION.

```
typedef struct _OBJECT_BASIC_INFORMATION {
    ULONG Attributes;
    ACCESS_MASK GrantedAccess;
    ULONG HandleCount;
```

```
    ULONG PointerCount;
    ULONG PagedPoolCharge;
    ULONG NonPagedPoolCharge;
    ULONG Reserved[3];
    ULONG NameInfoSize;
    ULONG TypeInfoSize;
    ULONG SecurityDescriptorSize;
    LARGE_INTEGER CreationTime;
} OBJECT_BASIC_INFORMATION, *POBJECT_BASIC_INFORMATION;
```

Tableau 6.16. Valeurs initiales des champs de la structure OBJECT_BASIC_INFORMATION

Champ	Valeur initiale
Attributes	OBJECT_HEADER Flags
HandleCount	OBJECT_HEADER HandleCount
PointerCount	OBJECT_HEADER PointerCount
CreationTime	OBJECT_SYMBOLIC_LINK CreationTime

OBJECT_NAME_INFORMATION.

```
typedef struct _OBJECT_NAME_INFORMATION
{
    UNICODE_STRING Name;
} OBJECT_NAME_INFORMATION, *POBJECT_NAME_INFORMATION;
```

OBJECT_TYPE_INFORMATION.

```
typedef struct _OBJECT_TYPE_INFORMATION {
    UNICODE_STRING TypeName;
    ULONG TotalNumberOfObjects;
    ULONG TotalNumberOfHandles;
    ULONG TotalPagedPoolUsage;
    ULONG TotalNonPagedPoolUsage;
    ULONG TotalNamePoolUsage;
    ULONG TotalHandleTableUsage;
    ULONG HighWaterNumberOfObjects;
    ULONG HighWaterNumberOfHandles;
    ULONG HighWaterPagedPoolUsage;
    ULONG HighWaterNonPagedPoolUsage;
    ULONG HighWaterNamePoolUsage;
    ULONG HighWaterHandleTableUsage;
    ULONG InvalidAttributes;
    GENERIC_MAPPING GenericMapping;
    ULONG ValidAccessMask;
    BOOLEAN SecurityRequired;
    BOOLEAN MaintainHandleCount;
    ULONG PoolType;
    ULONG DefaultPagedPoolCharge;
    ULONG DefaultNonPagedPoolCharge;
} OBJECT_TYPE_INFORMATION, *POBJECT_TYPE_INFORMATION;
```

OBJECT_TYPES_INFORMATION.

```
typedef struct _OBJECT_TYPES_INFORMATION {
    ULONG NumberOfTypes;
} OBJECT_TYPES_INFORMATION, *POBJECT_TYPES_INFORMATION;
```

Espace de noms d'objets

Vous pouvez voir les différents sous-ensembles que renferme l'espace de noms global du gestionnaire d'objets à l'aide de la commande !object.

```
lkd> !object \
```

```
Object: ffffb9894c8092a0  Type: (ffffa7049be74a60) Directory
    ObjectHeader: ffffb9894c809270 (new version)
    HandleCount: 0  PointerCount: 59
    Directory Object: 00000000  Name: \

    Hash Address          Type                      Name
    ``` ```---          ```
 01 ffffa704a0486550 Mutant   PendingRenameMutex
     ffffb9894c806e30 Directory   ObjectTypes
 02 ffffa7049be99990 FilterConnectionPort   storqosfltport
 03 ffffa7049dc51080 FilterConnectionPort   MicrosoftMalwareProtectionRemoteIoPortWD
 05 ffffb9894c80d430 SymbolicLink   SystemRoot
 06 ffffb9894daa0480 Directory   Sessions
     ffffa7049dc50370 FilterConnectionPort   MicrosoftMalwareProtectionVeryLowIoPortWD
 08 ffffb9894c80e060 Directory   ArcName
 09 ffffa704a03a4e40 FilterConnectionPort   WcifsPort
     ffffb9894c9c45a0 Directory   NLS
 10 ffffa704a5aa6160 Event   LanmanServerAnnounceEvent
     ffffa7049d852a20 ALPC Port   ThemeApiPort
     ffffa704a026b890 Device   UdfsCdRom
     ffffb9894d999ce0 Directory   Windows
     ffffb9894c809e10 Directory   GLOBAL??
 11 ffffb9894d9980c0 Directory   RPC Control
     ffffa7049d809490 ALPC Port   PdcPort
 13 ffffa704a046fc60 Event   EFSInitEvent
 14 ffffb9894d9bccd0 SymbolicLink   Dfs
     ffffa7049d8207d0 Device   clfs
 15 ffffa704a7dfe350 Event   CsrSbSyncEvent
     ffffa7049d559de0 ALPC Port   SeRmCommandPort
 16 ffffb9894c80a300 SymbolicLink   DosDevices
 17 ffffa704a054f7a0 Event   DSYSDBG.Debug.Trace.Memory.2b8
     ffffb9894d998660 Directory   KnownDlls32
 18 ffffb9894c810f10 Key   \REGISTRY
 19 ffffb9894daa0de0 Directory   BaseNamedObjects
 20 ffffb9894de94bb0 Section   Win32kCrossSessionGlobals
     ffffa7049be6dd60 ALPC Port   PowerPort
 21 ffffa704a5ad8c70 ALPC Port   SmSsWinStationApiPort
     ffffa704a62e2610 Event   UniqueInteractiveSessionIdEvent
     ffffb9894c8ad0a0 Directory   UMDFCommunicationPorts
 22 ffffb9894d998480 Directory   KnownDlls
     ffffa7049dfe4dc0 Device   FatCdrom
     ffffa7049dfecdc0 Device   Fat
     ffffa7049bfc5cb0 ALPC Port   PowerMonitorPort
 23 ffffa7049d85a2d0 Device   Ntfs
     ffffb9894c9c4ae0 Directory   FileSystem
     ffffb9894c809870 Directory   KernelObjects
 24 ffffa7049dc50f20 FilterConnectionPort   MicrosoftMalwareProtectionControlPortWD
 26 ffffa704a83c28b0 ALPC Port   SeLsaCommandPort
     ffffb9894c809470 Directory   Callback
 28 ffffa704a88792c0 FilterConnectionPort   BindFltPort
     ffffb9894c80cab0 Directory   Security
 29 ffffa704a0540d50 Device   UdfsDisk
 30 ffffa7049dc508f0 FilterConnectionPort   MicrosoftMalwareProtectionAsyncPortWD
     ffffb9894c80e4b0 Directory   Device
 32 ffffb9894d7edfa0 SymbolicLink   DriverData
 34 ffffb98951f79b60 Section   LsaPerformance
     ffffa704a0479090 ALPC Port   SmApiPort
 35 ffffa704a03a4c30 FilterConnectionPort   CLDMSGPORT
     ffffa7049dc509a0 FilterConnectionPort   MicrosoftMalwareProtectionPortWD
     ffffb9894c80da70 SymbolicLink   OSDataRoot
 36 ffffa704a05507e0 Event   SAM_SERVICE_STARTED
     ffffb9894c9c4920 Directory   Driver
     ffffb9894c89b380 Directory   DriverStores
```
```

Chapitre 7. Synchronisation

Les processus et threads fonctionnels au sein des systèmes informatiques modernes, pour la plupart multithread, multi-utilisateur à multi traitement symétrique, s'exécutent en *concurrence* les uns vis-à-vis des autres, et se font dès lors compétition pour accéder aux ressources de l'ordinateur, par exemple un processeur, un port d'imprimante, ou une région de mémoire partagée. Une problématique centrale qui découle de ces aspects est le besoin de coordonner, de *synchroniser*, les opérations. Dans ce chapitre, du reste comme autant de sujet qui le composeront, nous allons voir les moyens et les méthodes dont use Windows afin de concrétiser une entente intelligente entre différentes entités concurrentes. Nous expliquerons quels protocoles sont considérés en la matière, quelles primitives les représentent, ce qu'elles font, et comment chaque type de dispositif privilégiant de la synchronisation est utilisé.

Terminologie de synchronisation

- Une *ressource* désigne toute entité dont a besoin un processus en vue de poursuivre son exécution. Il peut s'agir d'une ressource matérielle, par exemple un processus ou un périphérique, ou d'une ressource logicielle, par exemple un fichier ou une variable. A tout instant, chaque ressource est caractérisée par son état (libre/occupée), éventuellement son nombre de points d'accès (à quoi correspond dans ce contexte le nombre de processus susceptibles d'utiliser en même temps la même ressource).

- Une *section critique* fait référence à une séquence d'instructions qui, du fait de manipuler des ressources partagées entre différents processus, est susceptible de donner lieu à des résultats imprévisibles.

- La solution naturelle à la résolution des problèmes d'accès concurrents consiste à interdire la modification de ressources partagées à plus d'un processus à la fois, à quoi se ramène le concept d'*exclusion mutuelle*. Il existe quatre conditions nécessaires à la validité d'une section critique : (1) une ressource critique ne peut effectivement être manipulée que par un seul processus à la fois (corolaire : deux processus ne peuvent jamais se trouver simultanément dans la même section critique) ; (2) aucune hypothèse n'est faite en ce qui concerne la vitesse potentielle des processus et le nombre de processeurs dont ils disposent ; (3) aucun processus suspendu en dehors de sa section critique ne doit bloquer les autres ; (4) un processus réclamant un accès à une section critique doit obtenir satisfaction au bout d'un temps borné.

- On parle de *situation de compétition* (race condition) pour décrire un ensemble d'opérations dont l'issue (le résultat) dépend de l'ordre dans lequel elles sont effectuées. Un tel contexte peut émerger dès que plusieurs acteurs ont un même potentiel d'action vis-à-vis d'une ressource partagée et qu'au moins l'un d'entre eux est susceptible de modifier son état.

Mécanismes de synchronisation

L'une des forces caractéristiques des systèmes Windows réside dans la profusion de mécanismes de synchronisation intégrés à l'environnement. Cette richesse, essentielle du fait de mener à un meilleur contrôle, peut toutefois s'avérer une difficulté supplémentaire notable. D'un côté, un ensemble aussi complet permet aux concepteurs de logiciels d'optimiser leur choix quant à la méthode la mieux adaptée en vue des circonstances. D'un autre côté, la gestion des accès concurrents est un phénomène complexe qui, tant il fait intervenir de facteurs et de considérations à saisir (en premier lieu les exigences liées à l'ordonnancement et à la configuration IRQL), a en définitive de quoi laisser perplexe.

Les systèmes d'exploitation modernes font place à deux grandes catégories de verrous : les uns basés sur l'attente active, les autres sur l'attente passive.

La méthode la plus adéquate pour synchroniser des accès dépend des opérations nécessitant de l'exclusion mutuelle. Les méthodes de synchronisation peuvent être rangées en plusieurs grandes catégories. Dans chaque catégorie, Windows fournit un ou plusieurs mécanismes spécifiques que les composants mode noyau ont loisir d'employer. (Certain de ces mécanismes ne sont pas documentés, ni exportés par le noyau, en conséquence de quoi leur usage est réservé au système d'exploitation.) La liste qui suit répertorie les solutions les plus courantes en matière de synchronisation.

- *Masquage des interruptions* Compte tenu des liens étroits entre ordonnancement et interruption, masquer quand nécessaire les interruptions permet de résoudre le problème : plus (au sens négatif, s'entend) d'interruption signifie plus de commutation de contexte, donc aucun risque de circonvenir l'exclusion mutuelle. Cette solution présente néanmoins deux graves inconvénients. D'une part, elle monopolise le processeur au bénéfice d'un seul processus, empêchant l'exécution de tous les autres, y compris ceux n'ayant aucun lien avec la ressource (et qui ne l'ont en l'occurence jamais réclamé). D'autre part, elle est inadaptée à une perspective multi processeur. (Nous reviendrons sur le sujet plus loin.) Exemple de mécanisme que Windows gère pour la catégorie exclusion mutuelle par masquage des interruptions : IRQL.
- *Opérations inter verrouillées*
- *Mutex* Exemple de mécanisme Windows où l'exclusion mutuelle est assurée à l'aide de mutex : spinlocks, mutex rapides, mutex gardés, mutex noyau dispatcher, événement noyau dispatcher.
- *Verrous*
- *Sémaphore* Exemple de mécanisme Windows où l'exclusion mutuelle est assurée à l'aide de sémaphores : sémaphore noyau dispatcher.

Tableau 7.1. Mécanismes noyau de synchronisation

| | | |

Non
Non
Non
Oui
Ressource exécutif
Oui
Oui
Non
Non
Oui

Problèmes classiques de synchronisation

Sous Windows, c'est le role du noyau de fournir des mécanismes qui empêcheront deux threads de modifier la même structure en même temps. Afin de mieux comprendre pourquoi ce problème est important, considérez une situation où deux threads incrémentent une variable globale. Au plus haut niveau d'abstraction, l'opération susmentionnée se traduit ainsi :

- Lecture de la variable vers un registre
- Incrément du registre
- Ecriture de la valeur du registre vers la variable

Dans l'éventualité où les deux threads s'exécutent simultanément sur un système multiprocesseur sans aucune forme de synchronisation, les résultats d'une mise à jour risquent d'être perdus. Par exemple, supposons que la valeur initiale de MyVar est 0 et que les opérations se déroulent dans l'ordre indiqué ci-dessous.

Thread A sur processeur 1	**Thread B sur processeur 2**
Lecture de Var dans un registre	
	Lecture de Var dans un registre
	Incrément du registre
	Affectation du registre vers Var
Incrément du registre	
Affectation du registre vers Var	

A l'issue de la séquence d'instructions qui précède, la valeur de Var devrait être égale à deux, correspondant au nombre de fois où elle a été actualisé. Cependant, comme le premier thread a lu variable avant que le second ait fini de la mettre à jour, il s'ensuit un partage incorrect de la mémoire. Au final, la variable vaut 1, alors que deux incréments la concernent.

L'exemple que nous venons de donner montre ce qu'il est susceptible d'advenir dans une perspective multi processeur, mais la même erreur pourrait avoir lieu sur un configuration mono processeur si le système d'exploitation devait effectuer un basculement de contexte. En effet, lors d'une telle procédure, le système sauvegarde l'état volatile du thread en cours et charge l'état sauvegardé du thread au bénéfice duquel s'effectue le basculement. L'exemple qui suit illustre une situation de compétition issue de la préemption.

Thread A sur processeur 1	**Thread B sur processeur 2**
Lecture de Var dans un registre	
Préemption du thread A et reprise de l'exécution du thread B	
	Lecture de Var dans un registre
	Incrément du registre

Thread A sur processeur 1	Thread B sur processeur 2
	Affectation du registre vers Var
Préemption du thread B et reprise de l'exécution du thread A	
Incrément du registre	
Affectation du registre vers Var	

Dans la perspective la plus large, tous les problèmes de synchronisation se ramènent à la gestion de l'accès concurrent à une ressource partagée. Il est dès lors nécessaire que le système fournisse un support efficace et opportun de l'exclusion mutuelle.

Exclusion mutuelle

Procurer aux logiciels des méthodes leur permettant d'accéder en toute sécurité aux ressources partagées de l'ordinateur (qui le sont du reste à peu près toutes, matérielles comme logicielles) fait sans aucun doute partie des entreprises les plus cruciales du système d'exploitation, qui doit à cet égard gérer le problème des accès concurrents, et de la sorte permettre l'exclusion mutuelle entre les différents processus.

Le concept d'exclusion mutuelle a au sein de tout système d'exploitation une importance logistique fondamentale, en ce qu'il assure une structure cohésive entre les différentes activités d'un même sous ensemble. Exclusion mutuelle signifie qu'un thread et un seul peut accéder à une certaine ressource à un moment donné. Il faut de l'exclusion mutuelle dès lors qu'une ressource ne se prête pas aux accès partagés, ou quand le partage donnerait des résultats imprévisibles. Par exemple, si deux threads envoient un fichier vers un port d'imprimante en même temps, un mélange des impressions pourrait avoir lieu. De même, si un thread lit un emplacement mémoire tandis qu'un autre écrit dessus, il existe une probabilité très forte que le premier thread obtienne des résultats différents de ceux attendues.

De manière générale, la problématique et les enjeux dont font l'objet l'exclusion mutuelle apparaissent chaque fois que plusieurs threads (ou à plus haut niveau, plusieurs processus) interviennent dans une chaine de traitement. Toutes les ressources modifiables sont concernées, et ne peuvent en l'occurence être partagées sans restrictions, contrairement aux ressources qui ne sont pas sujettes à modifications - par exemple une variable globale système dont la valeur serait définie une fois pour toute lors de la phase d'amorçage.

Les sections de code où s'effectue l'accès à des données partagées sont dites sections *critiques*. Pour assurer le bon fonctionnement du code, un seul thread à la fois doit être en mesure d'exécuter une section critique. Un programme ne pouvant faire aucune hypothèse sur l'ordre dans lequel les threads sont traités par l'ordonnanceur, des mécanismes gérant les accès concurrents doivent être mis en place.

Déjà complexe dans une perspective multitâche, le problème de l'exclusion mutuelle l'est d'autant plus sur les architectures à multi traitement symétrique, où le même code système est exécuté simultanément sur plusieurs processeurs et se partage des structures de données situés dans la mémoire globale. Le noyau Windows offre en conséquence diverses primitives d'exclusion mutuelle que lui, et le reste de l'exécutif, utilise pour synchroniser ses accès aux structures de données globales.

Si l'exclusion mutuelle est incorrectement prise en compte, il peut en résulter un interblocage (*deadlock*), à quoi correspond une situation dans laquelle deux processus concurrents s'attendent l'un l'autre, et s'empêchent ainsi mutuellement d'aller plus loin dans leur exécution.

Synchronisation au niveau IRQL haut

Un souci majeur culminant dans la gestion des accès concurrentiels se rapporte aux interruptions. Ainsi, supposons que le noyau soit en train de modifier une structure de données globale au moment où survient une interruption dont la routine de traitement associée modifie elle aussi la structure.

Sur une configuration mono processeur, le moyen le plus intuitif de parvenir à de l'exclusion mutuelle consiste à désactiver toutes les interruptions, empêchant de la sorte le code d'ordonnancement de s'exécuter. Windows,

de son coté, s'appuie en la matière sur une solution plus fine. Ainsi, avant d'utiliser une ressource globale, le noyau masque temporairement les interruptions dont la clientèle utilise les mêmes ressources. Pour ce faire, il augmente l'IRQL du processeur jusqu'au niveau maximal utilisé par une source d'interruptions potentielle. Par exemple, l'ordonnanceur synchronise les accès à ses structures de données, notamment la base de données du dispatcher, à l'IRQL DPC/dispatch. Dans l'éventualité où une autre partie du noyau doit accéder à ladite base, elle devra alors augmenter l'IRQL jusqu'au même niveau, masquant de ce fait les interruptions DPC/dispatch.

La stratégie décrite en amont, si elle convient parfaitement pour un système mono processeur, ne se prête pas à une configuration multi processeur, où augmenter l'IRQL sur un processeur n'isole en rien des interruptions émanant d'un autre processeur. Le noyau doit donc s'y prendre différemment en vue de garantir des chemins d'accès mutuellement exclusifs sur l'ensemble des processeurs.

Synchronisation au niveau IRQL bas

Sémaphores

Parmi les quelques objets du noyau rendus visibles en tant que primitives de synchronisation, les sémaphores se révèlent particulièrement utiles dans le but de maitriser le nombre de tâches (processus ou threads) potentiellement actives simultanément. Mécanisme très commun en programmation concurrente, on trouve des sémaphores dans bon nombre de composants intégrés à Windows : applications utilisateur, bibliothèques de support système, pilotes de périphérique, etc.

À propos générique concernant les sémaphores

Conceptuellement, un sémaphore joue le rôle d'un distributeur de tickets. Un processus du sémaphore demande un ticket en invoquant une opération particulière (sur laquelle nous reviendrons par la suite). Si au moins un ticket est disponible, le processus qui le réclame le prend et poursuit son exécution. Dans le cas contraire, le demandeur est enregistré dans une file d'attente et est bloqué dans l'attente de l'obtention d'un ticket. Par l'entremise et les effets d'une opération inverse, un processus dépose son ticket dans le distributeur, avec comme conséquence de rendre celui-ci à nouveau disponible. Si des processus sont en attente dans la file du sémaphore, le premier d'entre eux est débloqué et obtient son ticket.

Un sémaphore est dans sa forme la plus simple une variable qui tient lieu de compteur et à laquelle il est possible de n'accéder qu'à travers trois opérations : l'initialisation, qui consiste à donner au sémaphore une valeur initiale, le test, qui consiste à décroître la valeur du sémaphore d'une unité à condition que le résultat ne devienne pas négatif, et l'incrément, qui consiste simplement à accroître la valeur du sémaphore d'une unité. Ces deux dernières opérations sont généralement notées P et V, en référence aux abréviations des termes *tester* et *incrémenter* en hollandais, langue maternelle de l'inventeur des sémaphores, Edsger Dijkstra. En dehors de la phase d'initialisation, faite logiquement en une seule fois et qui n'est à ce titre pas touchée par les contraintes inhérentes aux systèmes multi programmés, les autres opérations prises en charge par les sémaphores sont atomiques, c'est-à-dire qu'elles ne peuvent être ni divisées ni interrompues. En particulier, l'activité du processus qui effectue une telle opération ne peut pas être commutée au cours de celle-ci. La conséquence directe de cet aspect est qu'un processus qui désire exécuter une opération déjà en cours d'exécution au niveau d'un autre processus doit attendre que celui-ci prenne fin. L'atomicité des opérations peut être assurée par une instruction processeur spéciale ou en ignorant les interruptions afin d'empêcher d'autres processus de devenir actifs.

Si un sémaphore ne peut posséder que les valeurs 0 et 1, il est appelé sémaphore binaire. Un tel sémaphore est couramment appelé sémaphore d'exclusion mutuelle (mutex). S'il peut prendre des valeurs positives quelconques, il s'agit d'un sémaphore général ou à compteur.

Les sémaphores sont employés aussi bien dans le cadre de l'exclusion mutuelle que dans le contexte dans la synchronisation conditionnelle. Pour réaliser une exclusion mutuelle, on associe un sémaphore à une section critique. (Notez que nous utilisons ici ce terme dans une acception très classique - à savoir un ensemble d'instructions qui ne peuvent être traitées que par une entité exécutable à la fois, sans pour autant renvoyer

au mécanisme Windows du même nom.) Pour réaliser une synchronisation conditionnelle, on associe un sémaphore à chaque condition.

Structure de données

Les sémaphores sont de nature assez basique pour ne pas avoir de représentation qui leur est associée au sein de la couche exécutive de Windows. La structure KSEMAPHORE sert au niveau système à agréger les données individuelles de chaque sémaphore.

```
lkd> dt nt!_KSEMAPHORE
   +0x000 Header           : _DISPATCHER_HEADER
   +0x018 Limit            : Int4B
```

La structure KSEMAPHORE, si elle contient un nombre effectivement réduit d'attributs, se montre typique de la façon dont Windows implémente la plupart des dispositifs servant à synchroniser l'exécution de multiples flux.

- **Objet dispatcher (Header)** Incorpore des capacités de synchronisation sur lesquelles reposent les processus et les threads quand ils emploient des sémaphores.
- **Compteur de sémaphore (Limit)** Compteur qui contrôle l'accès à une ressource en permettant à un nombre maximal de threads d'accéder à la ressource ainsi protégée.

L'en-tête dispatcher encapsule le type d'objet, l'état de signal et la liste des threads attendant le sémaphore.

Interfaces Windows de sémaphore

Le tableau énumère les fonctions Windows les plus courantes en matière de sémaphores.

Tableau 7.2. API Windows de sémaphore

Fonction	Description
CreateSemaphore	Créee et initialise un objet sémaphore ou sans nom, avec une valeur de compteur
CloseHandle	Ferme un handle de sémaphore et supprime la référence à l'objet sous-jacent
WaitForSingleObject	Attend que l'appartenance d'un seul objet sémaphore lui soit accordée
WaitForMultipleObjects	Attend que l'appartenance d'un ou plusieurs objets sémaphore lui soit accordée.
ReleaseSemaphore	Libère un objet sémaphore.

Tableau 7.3. SEMAPHORE_INFORMATION_CLASS

ID	Classe	Type de données associé
0x0000	SemaphoreBasicInformation	SEMAPHORE_BASIC_INFORMATION

typedef struct _SEMAPHORE_BASIC_INFORMATION { LONG CurrentCount; LONG MaximumCount; } SEMAPHORE_BASIC_INFORMATION, PSEMAPHORE_BASIC_INFORMATION;

Tableau 7.4. Valeurs initiales des champs de la structure OBJECT_BASIC_INFORMATION

Champ	Valeur initiale
CurrentCount	KSEMAPHORE Header.SignalState
MaximumCount	KSEMAPHORE Limit

Compteur et état de signal de sémaphores

Comme tous les objets qui obtiennent leurs capacités de synchronisation d'objets noyau dispatcher, un sémaphore est à tout moment dans l'un des états suivants : signalé ou non-signalé. Cette dualité, si elle bien

celle observable parmi toutes les fonctions et tous les services (internes ou non) relatifs aux sémaphores, reflète assez peu la façon dont les choses se passent en interne. Dans un sémaphore, plutôt qu'une sémantique de signal actif / non actif, c'est la valeur du compteur de sémaphore, lequel limite le nombre de threads pouvant accéder simultanément à l'objet de synchronisation, qui fait office d'état de signal.

Le rapport entre un sémaphore, le compteur et l'état de signal de l'objet est le suivant : le sémaphore est considéré signalé quand le compteur est positif, non signalé quand le compteur vaut 0. La libération d'un sémaphore augmente le compteur, tandis qu'une attente de l'objet le décrémente. Si le compteur décroît jusqu'à la valeur nulle, le sémaphore est mis à l'état non signalé, avec comme conséquence le blocage des appelants de fonctions d'attente d'objet dispatcher (KeWaitForSingleObject, KeWaitForMultipleObjects).

A ce moment, aucun autre thread ne peut accéder au sémaphore, donc a fortiori aux ressources protégées par ce biais, tant que le thread propriétaire n'aura pas appelé de fonction de libération de sémaphore, ce qui a pour effet d'incrémenter le compteur d'une valeur spécifiée et de remettre à nouveau l'objet sémaphore dans l'état signalé.

Un moyen efficace d'interroger l'état du signal ou le compteur associé à un sémaphore (vous venons de voir que ces notions se recouvraient) est d'abord constitué par la fonction mode noyau KeReadStateSemaphore. En interne, cette routine extrait l'état de signal de l'objet dispatcher sur lequel s'appuie le sémaphore passé en paramètre, et retourne la valeur à l'appelant. Attention toutefois, puisque KeReadStateSemaphore, au contraire de traitements comme KeReleaseSemaphore ou KeWaitForSingleObject, ne synchronise pas ses accès à l'objet. En mode utilisateur, si le sous-système Windows ne fournit pas en l'état de méthode pour interroger le compteur d'un sémaphore, on trouve dans Ntdll un appel avec lequel parvenir à cette fin : NtQuerySemaphore.

Partage d'objets sémaphore

Plusieurs processus peuvent obtenir une référence vers un même objet sémaphore en vue de se le partager. La fonction Windows CreateSemaphore, avec laquelle créer un objet sémaphore, est la première à rendre visible ce comportement, dans la mesure où elle permet la production d'un objet idoine qui soit pourvu d'un nom. (Des sémaphores construits sans nom ne sont pas disponibles pour la synchronisation entre processus.) Un processus peut créer, nommer un objet (paramètre lpName de la fonction CreateSemaphore) et un second processus ouvrir cet objet (OpenSemaphore) et obtenir un descripteur partagé avec le premier processus. Aussi, puisque un processus obtient un descripteur en recevant une copie d'un autre processus, un processus peut spécifier le descripteur d'un sémaphore lors d'un appel à la fonction DuplicateHandle, avec pour résultat la création d'un duplicata pouvant être utilisé par un autre processus. Finalement, autre approche concernant le partage d'objets de type sémaphore, un processus peut hériter d'un processus parent un descripteur du même sémaphore - ce seulement si les règles de succession dictés par le processus créateur du sémaphore l'y autorisent (paramètre lpSemaphoreAttributes de CreateSemaphore)

Limite

En plus de la valeur de compteur, chaque sémaphore se voit attribuer une limite, qui résume, en chiffre, les capacités maximales d'un objet à octroyer des accès à une ressource. La notion de limite, dans ce contexte, apparaît dès la fonction d'initialisation des objets (keInitializeSemaphore), où un paramètre d'appel nommé Limit indique la valeur de compte maximum qu'un sémaphore peut atteindre. Une fois enregistrée dans l'objet idoine (KSEMAPHORE Limit), la limite associée à tout sémaphore sert de barrière dans la routine KeReleaseSemaphore qui, si elle vient à ajuster le compteur de propriété d'un sémaphore au-delà la valeur de compteur maximal, soulève une exception (`STATUS_SEMAPHORE_LIMIT_EXCEEDED`). Une telle erreur survient, par exemple, quand un pilote essaie de libérer un sémaphore plus de fois que l'appartenance de l'objet sémaphore lui a été accordée.

%header

Fonction	Service	Routine
CreateSemaphore	NtCreateSemaphore	KeInitializeSemaphore
OpenSemaphore	NtOpenSemaphore	ObOpenObjectByName
ReleaseSemaphore	NtReleaseSemaphore	KeReleaseSemaphore

NtQuerySemaphore

KeReadStateSemaphore

Synchronisation sur le plan utilisateur

Sections critiques

Parmi d'autres mécanismes que fournit Windows afin de parvenir à de l'exclusion mutuelle, les sections critiques sont des objets qui synchronisent les threads et coordonnent les accès à des ressources partagées dans le contexte d'un processus donné. (Une section critique ne peut pas être partagée par plusieurs processus.) Le fait qu'un thread soit en possession d'une telle primitive (on parle en la matière d'entrer dans une section critique, et à l'inverse d'en sortir) lui confère l'exclusivité des accès à certaines ressources ou à certains traitements. En effet, si un autre thread tente d'entrer dans la même section critique, il est alors suspendu dans l'attente de la sortie de la section critique par le thread qui s'y trouve à ce moment-là.

Une majeure partie de la prise en charge des sections critiques est assurée par la bibliothèque Windows principale (Kernel32), épaulée en la matière par quelques routines générales de la bibliothèque d'exécution (fonctions Rtl dans Ntdll). L'autre partie, effectuée coté noyau, endosse la responsabilité des autres objets de synchronisation qu'utilisent les sections critiques pour les cas avec compétition, soit en l'occurence les événements (à réinitialisation automatique ou à clé).

Généralités

Le grand principe à partir duquel partent les sections critiques est de rendre atomique et de donner un caractère mutuellement exclusif à un ensemble d'instructions qui, s'il n'était pas assujetti à ces obligations, serait susceptible de produire des résultats imprévisibles. De ce fait, un seul thread à fois peut accéder à la ressource en toute sécurité à un moment donné.

En général, les processus qui exécutent des sections critiques sont structurés comme suit :

1. Section non critique
2. Demande d'entrée en section critique
3. Section critique
4. Accès aux ressources partagées
5. Demande de sortie de la section critique
6. Section non critique

La première chose à se produire lors d'une demande d'entrée en section critique (étape 1 et 2) est l'évaluation par le système de la disponibilité de ladite section. Un thread qui réclame une section critique déjà en possession d'un autre thread est mis en attente, et un basculement de contexte en faveur de ce thread aura lieu quand les circonstances se montreront propices. Lorsque celui-ci s'effectue, le thread est alors en mesure d'accéder aux ressources partagées (étape 3 et 4). Une fois terminé, le thread présente présente une demande de sortie de section critique (étape 5 et 6), de façon à ce que l'exécution d'autres threads puisse suivre le chemin qui lui-même vient d'emprunter.

Utilisation

La tableau qui suit énumère les principales fonctions de l'API Windows avec lesquelles les applications interagissent dès lors qu'elles emploient des sections critiques.

Tableau 7.5. API de section critique

Fonction	Description
InitializeCriticalSection	Crée et initialise un objet section critique
InitializeCriticalSectionAndSpinCount	Crée un objet section critique avec compteur de rotations
DeleteCriticalSection	Détruit un objet section critique.
EnterCriticalSection	Obtient un objet section critique.
TryEnterCriticalSection	Tente d'obtenir un objet section critique.
LeaveCriticalSection	Libère un objet section critique.
SetCriticalSectionSpinCount	Définit le compteur de rotations d'un objet section critique.

L'utilisation des sections critiques est relativement simple. Comme prélude, l'application doit déclarer une variable de type CRITICAL_SECTION, généralement globale. Sur le plan factuel, cela correspond à la création d'un objet section critique, dont il faut ensuite procéder à l'initialisation via InitializeCriticalSection. Avant toute toute utilisation d'une section critique donnée, il est obligatoire de l'initialiser.

Pour accéder à une ressource protégée par une section critique, un thread appelle la fonction EnterCriticalSection, dont le rôle est d'attendre que l'objet correspondant soit disponible, de sorte à pouvoir se l'approprier par la suite. Dans l'éventualité où une section critique est libre (au sens où aucun thread ne se trouve à ce moment en son sein), il en résulte que n'importe quel thread peut y entrer. Dans le cas contraire, Si un thread veut entrer avec un appel EnterCriticalSection dans une section qui n'est pas libre, le noyau le force à entrer dans l'état d'attente.

Dans certains cas, il n'est pas efficace suspendre l'exécution du thread, par exemple quand une ressource facultative a de fortes probabilités de se montrer indisponible pour de longs moments, voire ne jamais l'être. Le cas échéant, les traitements effectués pour l'attente de la ressource consomment les ressources du noyau sans même de réelle plus-value, tant que l'attente n'est pas satisfaite, rien ne peut motiver le thread à reprendre son exécution. Un recours spécifique à ce problème consiste à utiliser une section critique sans blocage, cela en appelant la fonction TryEnterCriticalSection. Cette fonction tente d'obtenir la section critique et retourne immédiatement si la section critique ne peut pas être utilisée (elle ne fait en l'occurence l'effet d'aucun blocage). Le thread peut alors poursuivre son exécution en suivant un chemin de code alternatif.

Après obtention de l'objet section critique à l'aide des fonctions EnterCriticalSection ou TryEnterCriticalSection, et tant que dure la possession dudit objet, le thread bénéficie d'un accès exclusif à la ressource. Entre autres, ce mécanisme explique pourquoi pourquoi il est imprudent et contre-indiqué de mettre fin à des threads en utilisant TerminateThread, laquelle fonction n'effectue pas de nettoyage. Quand un thread terminé de la sorte est propriétaire d'une section critique, la ressource ainsi protégée devient inutilisable jusqu'à ce que l'utilisateur redémarre l'application.

Comme jamais plus d'un seul thread ne peut être actif au coeur d'une section critique, le thread la détenant doit la libérer le plus vite possible pour éviter qu'elle ne devienne un goulot d'étranglement. De la même manière, du fait que les instructions exécutées dans une section critique ne peuvent interrompues, le chemin d'exécution emprunté doit être court (en temps d'exécution) et fini - ce n'est en l'occurence jamais un bon endroit pour effectuer des traitements complexes ou dont découlent des opérations hautement consommatrices de ressources. Libérer la section critique permet d'améliorer les performances en permettant aux threads en attente d'y accéder.

Extension de commande !cs

L'extension de commande !cs du débogueur permet de voir quelles sections critiques sont utilisés par un processus. Ce qui suit montre un exemple de sortie de la commande, ciblant une instance de la Calculatrice (Calc.exe).

```
0:003> !cs
-----------------------------------------
DebugInfo          = 0x000007ff5060ece0
Critical section   = 0x000007ff50608140 (ntdll!RtlpProcessHeapsListLock+0x0)
```

```
NOT LOCKED
LockSemaphore      = 0x0
SpinCount          = 0x0000000000000000
-----------------------------------------
DebugInfo          = 0x000007ff5060ed10
Critical section   = 0x000000259e3c0298 (+0x259E3C0298)
NOT LOCKED
LockSemaphore      = 0x0
SpinCount          = 0x0000000000000000
-----------------------------------------
.
.
.
-----------------------------------------
DebugInfo          = 0x000000259e40b040
Critical section   = 0x000007ff49c44300 (WINMM!TimerThreadCritSec+0x0)
NOT LOCKED
LockSemaphore      = 0x174
SpinCount          = 0x0000000000000000
```

Les informations présentées par la commande sont en réalité une version abrégée de la structure logique réelle sous-jacente aux sections critiques, que nous verrons plus loin. Elle permet par contre d'afficher de façon claire les renseignements les plus utiles, à savoir l'état dans lequel se trouve la section critique, le nombre de threads qui l'ont réclamé, ainsi que les informations d'identification de celui qui en a actuellement la possession. La commande !cs peut peut également être utilisé dans le but d'examiner une section critique individuelle, cela en spécifiant lors de l'appel l'adresse de la section, comme indiqué ici.

```
0:003> !cs 0x000007ff49c44300
-----------------------------------------
Critical section   = 0x000007ff49c44300 (WINMM!TimerThreadCritSec+0x0)
DebugInfo          = 0x000000d69374a9b0
NOT LOCKED
LockSemaphore      = 0x170
SpinCount          = 0x0000000000000000
```

Structures de données et mécanismes internes

Les données relatives aux sections critiques sont regroupées au sein du module ntdll et ont pour représentation sous-jacente la structure _RTL_CRITICAL_SECTION, que vous pouvez voir au moyen de la commande dt.

```
0:001> dt ntdll!_RTL_CRITICAL_SECTION
   +0x000 DebugInfo        : Ptr64 _RTL_CRITICAL_SECTION_DEBUG
   +0x008 LockCount        : Int4B
   +0x00c RecursionCount   : Int4B
   +0x010 OwningThread     : Ptr64 Void
   +0x018 LockSemaphore    : Ptr64 Void
   +0x020 SpinCount        : Uint8B
```

Les données individuelles qui font partie de la structure RTL_CRITICAL_SECTION sont montrées plus en détail dans ce qui suit.

- *Informations de débogage* (*DebugInfo*) Contient un assortiment d'informations additionnelles en ce concerne la section, par exemple s'il y a de la compétition pour l'objet.
- *Compteur de verrouillage* (*LockCount*) Masque binaire servant à déterminer l'état de la section critique (verrouillée/déverrouillée) et le nombre de threads en attente sur elle. Le bit de poids faible indique l'état du verrou : inarmé (0) pour un verrou actif, armé (1) pour un inactif. Modifier la valeur de cet attribut ne sollicite pas le noyau, profite d'opérations inter verrouillées, et permet au système de déterminer rapidement quand une section critique peut immédiatement satisfaire une attente.
- *Compteur de récursion* (*RecursionCount*) Comptabilise le nombre d'entrée dans la section critique de la part de son thread propriétaire. Les objets avec lesquels fonctionnent les sections critiques peuvent être acquis de

manière récursive, en conséquence de quoi un thread peut entrer à volonté dans une même section critique, et appeler plusieurs fois EnterCriticalSection.

- *Propriétaire* (*OwningThread*) ID client du thread entré dans la section critique. Notez que la notion de propriété s'applique dans ce contexte non aux objets internes alloués pour la prise en charge de la section, mais se réfère bel et bien aux instructions de la section.

- *Objet interne de synchronisation* (*LockSemaphore*) Handle d'un objet événement à réinitialisation automatique.

- Compteur de rotations (*SpinCount*) Permet de temporiser la main mise de l'ordonnanceur envers un thread réclamant une section critique momentanément indisponible (systèmes multiprocesseurs seulement).

Maintenant que nous avons vu quels éléments constituaient la structure _RTL_CRITICAL_SECTION, passons en revue ceux appartenant à la sous-structure RTL_CRITICAL_SECTION_DEBUG. Remarquez que si la liste qui vient vous parait courte, c'est pour la simple et bonne raison que la plupart des données qu'emporte l'enregistrement RTL_CRITICAL_SECTION_DEBUG sont ignorées dans les systèmes Windows modernes.

```
0:000> dt ntdll!_RTL_CRITICAL_SECTION_DEBUG
   +0x000 Type             : Uint2B
   +0x002 CreatorBackTraceIndex : Uint2B
   +0x008 CriticalSection  : Ptr64 _RTL_CRITICAL_SECTION
   +0x010 ProcessLocksList : _LIST_ENTRY
   +0x020 EntryCount       : Uint4B
   +0x024 ContentionCount  : Uint4B
   +0x028 Flags            : Uint4B
   +0x02c CreatorBackTraceIndexHigh : Uint2B
   +0x02e SpareUSHORT      : Uint2B
```

Section critique (*CriticalSection*) Pointeur vers la section critique associée à cet enregistrement ; permet essentiellement de partir de la structure de débogage d'une section pour arriver à la section proprement dite.

Liste des sections critique (*ProcessLocksList*) Noeud de liste parmi la liste des sections critiques d'un processus.

Compteur de compétition Enregistre le nombre de fois que des threads sont entrés en état d'attente suite à cause de l'indisponibilité de la section critique.

Les sections critiques constituent un mécanisme de synchronisation plus primitif que les mutex, mais aussi plus léger. S'il n'y a pas de compétition, les fonctions de section critique ne font pas de transition vers le mode noyau. (La plupart du code machine exécutable concernant ce dispositif est réparti entre deux bibliothèques fondamentales du sous-système Windows en mode utilisateur, à savoir Kernel32 et Ntdll.) S'il y a de la compétition, le système alloue dynamiquement un objet événement à réinitilisation automatique, qui sera mis à l'état signalé lorsque le thread propriétaire de la section critique s'en sera détaché (LeaveCriticalSection).

```
0:003> !cs 0x000007ff49c44300
-----------------------------------------
Critical section   = 0x000007ff49c44300 (WINMM!TimerThreadCritSec+0x0)
DebugInfo          = 0x000000259e40b040
NOT LOCKED
LockSemaphore      = 0x174
SpinCount          = 0x0000000000000000

0:003> !handle 174 f
Handle 174
  Type          Event
  Attributes    0
  GrantedAccess 0x100003:
         Synch
         QueryState,ModifyState
  HandleCount   2
  PointerCount  524287
```

```
  Name          <none>
  Object Specific Information
    Event Type Auto Reset
    Event is Waiting
```

Liste des sections critiques de processus

Les sections critiques déclarées à l'intérieur d'un même processus sont chainées entre elles de sorte à former une liste. (C'est par ailleurs ce sur quoi se base la commande !cs pour mettre en avant ses résultats.) Pour avoir un aperçu de cette liste, considérez les faits suivants : (1) tout enregistrement de débogage de section (RTL_CRITICAL_SECTION_DEBUG) comporte à la fois un noeud parmi cette liste (ProcessLocksList) et un pointeur vers la section critique en elle-même, et (2) la variable RtlCriticalSectionList donne l'adresse du premier de ce noeuds. Dans le débogueur, cela se présente comme suit.

```
0:003> !cs
-----------------------------------------
DebugInfo          = 0x000007ff5060ece0
Critical section   = 0x000007ff50608140 (ntdll!RtlpProcessHeapsListLock+0x0)
NOT LOCKED
LockSemaphore      = 0x0
SpinCount          = 0x0000000000000000
-----------------------------------------
DebugInfo          = 0x000007ff5060ed10
Critical section   = 0x000000d693700298 (+0xD693700298)
NOT LOCKED
LockSemaphore      = 0x0
SpinCount          = 0x0000000000000000

0:003> ? poi(ntdll!RtlCriticalSectionList)
Evaluate expression: 8793146584304 = 000007ff`5060ecf0

0:003> dt ntdll!_RTL_CRITICAL_SECTION_DEBUG ProcessLocksList
   +0x010 ProcessLocksList : _LIST_ENTRY

0:003> dt ntdll!_RTL_CRITICAL_SECTION_DEBUG 000007ff`5060ecf0-10
   +0x000 Type             : 0
   +0x002 CreatorBackTraceIndex : 0
   +0x008 CriticalSection  : 0x000007ff`50608140 _RTL_CRITICAL_SECTION
   +0x010 ProcessLocksList : _LIST_ENTRY [ 0x000007ff`5060ed20 - 0x000007ff`50608730 ]
   +0x020 EntryCount       : 0
   +0x024 ContentionCount  : 0
   +0x028 Flags            : 0
   +0x02c CreatorBackTraceIndexHigh : 0
   +0x02e SpareUSHORT      : 0

0:003> dt ntdll!_RTL_CRITICAL_SECTION 0x000007ff`50608140
   +0x000 DebugInfo        : 0x000007ff`5060ece0 _RTL_CRITICAL_SECTION_DEBUG
   +0x008 LockCount        : 0n-1
   +0x00c RecursionCount   : 0n0
   +0x010 OwningThread     : (null)
   +0x018 LockSemaphore    : (null)
   +0x020 SpinCount        : 0
```

Soulignons-le, la liste des sections critiques employées par un processus est uniquement locale, située dans l'espace d'adressage de ce processus. Il n'existe en l'occurence pas de recensement au niveau global (système) de ces dispositifs.

Sections critiques et événements à clé

Afin d'aider les processus à mieux affronter les situations de ressources faibles quand ils utilisent des sections critiques, Windows s'en remet à un objet événement à clé global du nom de CritSecOutOfMemoryEvent, lequel rend possible un chemin d'exécution alternatif suivi dès lors que les circonstances l'imposent.

Par défaut, le comportement habituel de la fonction Windows d'entrée en section critique (EnterCriticalSection) est de vérifier s'il y a de la compétition pour l'objet. S'il y en a, elle sollicite alors le noyau en vue de créer un objet événement pour la section critique. Si elle au premier abord triviale, une telle demande se décompose en diverses opérations toutes susceptibles de ne pas avoir d'issue favorable. À titres d'exemples, le système peut manquer de mémoire, se trouver à court de handles, voire être dans l'incapacité d'en insérer un dans la table des objets du processus. (Notez que les deux dernières options sont quand même rarissimes.) Même en tenant compte du fait que des exceptions sont soulevées à chacun de ces cas, il n'en reste pas moins que la structure de section critique est pour l'occasion laissée à un état indéterminé, ce qui en exclut absolument l'utilisation.

Pour toutes les raisons que nous venons de voir, quand l'allocation d'un événement dédié à une section critique échoue, Windows oblige le thread appelant à utiliser CritSecOutOfMemoryEvent au lieu d'un événement standard. Cet événement à clé - comme par ailleurs tous les autres événements de ce type - permet à un thread de spécifier une clé qu'il attend, le thread se réveillant quand un autre thread du même processus signale l'événement avec la même clé. Un thread attendant la section critique emploie de ce fait comme clé l'adresse de la section critique. Bien qu'il soit de nature globale, partagé en l'occurrence par tous les processus, CritSecOutOfMemoryEvent (encore une fois comme tous les autres objets assimilés) ne peut impliquer le réveil que d'un seul thread à la fois. Cela permet aux fonctions de section critiques de disposer d'un nombre arbitraire d'événements, sans les couts entrainés par l'allocation de tels objets.

CritSecOutOfMemoryEvent se situe dans l'espace dans le répertoire \Kernel de l'espace de noms du gestionnaire d'objets. L'objet est créé dès le démarrage du système, et tous les processus possèdent une référence sur lui.

```
lkd> !object \KernelObjects\CritSecOutOfMemoryEvent
Object: fffff8a00000e060  Type: (fffffa8044b23080) KeyedEvent
    ObjectHeader: fffff8a00000e030 (new version)
    HandleCount: 0  PointerCount: 2
    Directory Object: fffff8a000009a30  Name: CritSecOutOfMemoryEvent
```

Limites

Les principaux facteurs limitatifs en ce qui concerne les sections critiques sont attribuables au fait, d'une part, que ces entités ne soient pas par nature des objets, et d'autre part, qu'elles ne permettent pas la distinction entre accès en lecture et accès en écriture.

- *Absence de représentation objet* Ne faisant pas partie du modèle objet incorporé à Windows (il n'existe pas parmi les types déclarés au gestionnaire d'objets de représentation commune à toutes les sections critiques), les sections critiques sont ainsi privées de tous les avantages inhérents à ce dispositif : sécurité, nom, partage, etc. De ce fait, deux processus ne peuvent pas employer la même section critique pour coordonner leurs opérations, ni solliciter le duplicata d'un handle ou en hériter.

- *Absence de mode d'accès* Une section critique fait peu de cas qu'un thread accède à une ressource pour l'interroger (lecture) ou pour la manipuler (écriture). Si l'ensemble des accès à une variable partagée sont des accès en lecture seule, il n'y a en réalité pas besoin de les sérialiser. Dans ce cas de figure, les accès en lecture des différents threads sont inutilement exclusifs les uns par rapport aux autres, vu que ce sont uniquement les accès en écriture qui requièrent un accès exclusif. Les sections critiques ne mettent pas en oeuvre cette démarcation.

Sections critiques et systèmes multi processeurs

Sur un système à plusieurs processeurs (ou à un seul processeur muni de plusieurs coeurs), la suspension d'un thread désireux d'entrer dans une section critique devient si coûteuse qu'une alternative est privilégiée. Dans une telle configuration, plutôt que le système vienne modifier sans détour l'état d'ordonnancement du thread concerné (ce qui implique une transition vers le mode noyau), ce dernier boucle sur lui-même un certain nombre de fois en essayant d'obtenir l'accès à la section critique - ce que l'on pourrait en somme considérer comme des tours d'essais.

Le nombre d'itérations à réaliser avant suspension du thread réclamant une section critique est donné par un attribut nommée compteur de rotations (Spin Count). Sur un système à un seul processeur, une valeur de zéro pour cet aspect (mise en attente automatique si la section critique n'est pas immédiatement disponible) est parfaitement acceptable. Sur un système multi processeurs, on voudra habituellement un compteur de rotations plus élevé.

Deux fonctions de l'API Windows laissent transparaitre un comportement en lien avec le compteur de rotation d'une section critique. La première, SetCriticalSectionSpinCount, se limite à affecter audit attribut la valeur de la variable reçue en paramètre ; la seconde, InitializeCriticalSectionAndSpinCount, combine en une seule étape l'initialisation d'une section critique et du compteur lui étant propre.

Délai de sections critiques

Pour empêcher toute privation en termes de ressources, Windows limite la période maximale pendant laquelle des sections critiques peuvent être attendues. Lorsque ce délai expire, le système soulève une exception de type EXCEPTION_POSSIBLE_DEADLOCK, laquelle permet alors de pister d'éventuels inter blocages. Compte tenu des dysfonctionnements à quoi pourrait mener cette erreur, il est conseillé de gérer l'exception non au niveau logiciel mais au niveau source (autrement dit remonter à la source du problème en utilisant un débogueur.

Le paramètre de délai d'attente de sections critiques est déterminé en fonction de la valeur CriticalSectionTimeout sous la clé HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager.

Variables conditionnelles

Les variables conditionnelles sont des objets permettant aux threads de synchroniser leur exécution en fonction de développements ultérieurs, par exemple des circonstances nouvelles ou qui ont été modifiés depuis la dernière configuration. Elles offrent de la sorte un mécanisme d'attente flexible, adaptable par conséquent à de nombreux cas de figure, par exemple mettre en pause un thread jusqu'à ce qu'un traitement ait lieu, qu'une donnée atteigne une valeur particulière ou qu'un événement se produise.

Les variables conditionnelles sont à l'origine une primitive de synchronisation venues du monde Unix, définies dans la norme POSIX pthreads. Une particularité importante des variables conditionnelles est de ne pas constituer un outil de synchronisation complet (comprendre auto suffisant). De tels objets dépendent en effet par nature de primitives tierce, à savoir sections critique ou verrous en lecture/écriture. Chaque variable conditionnelle est ainsi liée à verrou externe qui la protège.

Un thread établit l'état initial auquel est subordonné une variable conditionnelle en sollicitant la fonction InitializeConditionVariable. Les threads souhaitant attendre sur une variable conditionnelle peuvent le faire par le biais soit des fonctions SleepConditionVariableCS, qui s'appuie sur une section critique (dans laquelle le thread appelant doit être entré), et SleepConditionVariableSRW, qui repose sur un verrou en lecture/écriture (dont le thread appelant doit être détenteur). Les fonctions de la gamme SleepConditionVariable ont deux effets majeurs : (1) libérer le verrou externe et (2) mettre le thread appelant en sommeil. Par la suite, lorsqu'un thread se réveille (soit parce que la variable a été notifiée, soit à cause de la fin d'une période d'attente), il se réapproprie le verrou externe, de sorte à garantir qu'en retour de ces fonctions, le verrou soit toujours aux mains du thread appelant.

Contrairement à d'autres environnements de synchronisation, par exemple les événements, il n'y a pas dans le contexte des variables conditionnelles de mémorisation de l'état interne. De ce fait, si une primitive de déblocage (WakeConditionVariable ou WakeAllConditionVariable) est appelée tandis qu'aucune tâche n'est en attente sur une variable conditionnelle, l'opération qui en résulte est en quelque sorte perdue (comprendre effectuée en pure perte, ne débouchant sur aucune action significative). Le prochain thread qui appellera SleepConditionVariable * sera bloqué jusqu'à un autre appel à WakeConditionVariable ou WakeAllConditionVariable.

Toutes les opérations en lien avec des sections critiques ou des verrous en lecture/écriture à l'intérieur des primitives de manipulation des variables conditionnelles sont atomiques, c'est à dire non interruptibles.

Tableau 7.6. Fonctions Windows concernant les variables conditionnelles

Fonction	Description
InitializeConditionVariable	Initialise une variable conditionnelle.
SleepConditionVariableCS	Attend sur une variable conditionnelle et libère la section critique sur laquelle elle s'appuie.
SleepConditionVariableSRW	Attend sur une variable conditionnelle et libère le verrou de lecture/écriture sur laquelle elle s'appuie.
WakeAllConditionVariable	Eveille tous les threads en attente sur une variable conditionnelle.
WakeConditionVariable	Eveille un thread en attente sur la variable de condition spécifiée.

Verrous lecture-écriture

La gestion de l'accès concurrent parmi les mécanismes de synchronisation offerts par Windows s'appuie généralement sur des stratégies de verrouillage dont la stricte application peut, pour certains types de transactions, s'avérer inutilement restrictive. Il n'est par exemple pas impératif quand toutes les interactions sur une variable partagée sont des accès en lecture d'exiger un fonctionnement transactionnel. Comme il n'y a aucun danger d'interférence entre plusieurs accès en lecture concurrents, il n'est pas nécessaire d'assurer l'exclusion mutuelle pour eux ; il n'y a autrement dit dans ce contexte pas de problème particulier à ce que différents threads puissent lire en même temps le même emplacement de mémoire. Les verrous en lecture/écriture (SRW, Slim Reader/Writer) constituent dans ce cas de figure une alternative pratique, de tels primitives pouvant être posés pour des accès en lecture (qui n'ont pas besoin d'être exclusif) ou bien en écriture (qui eux ont besoin de l'être).

La stratégie de verrouillage implémentée par les verrous de lecture-écriture autorise plusieurs lecteurs simultanés mais un seul écrivain. Sur le plan programmatique, cela signifie que plusieurs threads peuvent en même temps détenir un verrou en lecture, mais qu'un seul peut le modifier. Acquérir un accès en écriture est exclusif, dont il découle que pour acquérir un SRW en écriture, aucun autre thread ne doit posséder le verrou, ni en lecture, ni en écriture.

Les verrous type lecture-écriture constituent une optimisation des performances et permettent une plus grande concurrence dans certaines situations. En pratique, ils améliorent les performances pour les structures de données auxquels on accède le plus souvent en lecture sur des systèmes multiprocesseurs. Ils sont en outre extrêmement légers en termes d'occupation mémoire (chacun fait la même taille qu'un pointeur) et de performances (s'il n'y a pas de contention, les appels n'impliquent pas de passage en mode noyau). Autres avantages : l'utilisation, en interne, d'opérations inter verrouillées (atomiques), le réarrangement et l'optimisation des listes d'attente, ainsi que diverses mesures pour éviter les convois de verrouillage.

En contrepartie de ce qu'ils apportent, les verrous lecture-écriture imposent imposent quelques limitations, notamment le fait de de ne pas supporter la récursion. De ce fait, en cas d'un influx ininterrompu d'accès en lecture, un écrivain risque de se trouver dans une attente infinie, ce qu'on appelle une situation de famine. Pour les cas simples d'utilisation comme la protection d'une variable contre les accès concurrents, ils proposent cependant une alternative intéressante aux sections critiques- au point la plupart du temps de pouvoir se substituer complètement à ce genre de mécanisme.

A titre informatif, notez que s'il est sur le plan logique parfaitement compréhensible de penser à un verrou lecture-écriture comme à deux verrous séparés (un pour lire et un autre pour écrire), il n'existe en interne qu'un seul objet. Chaque verrou lecture-écriture est autrement dit unique, et c'est seule la mise en oeuvre de ce type de primitives qui permet de distinguer deux états.

Les données que Windows manipule par l'intermédiaire des fonctions de verrous lecture-écriture sont régis par le type RTL_SRWLOCK, lequel se trouve être en réalité un pointeur encapsulant l'adresse et les propriétés (au titre de verrou) d'un objet. Utilisez la commande dt des débogueurs Windows standards pour voir concrètement ces informations.

```
lkd> dt ntdll!_RTL_SRWLOCK
   +0x000 Locked           : Pos 0, 1 Bit
```

```
   +0x000 Waiting          : Pos 1, 1 Bit
   +0x000 Waking           : Pos 2, 1 Bit
   +0x000 MultipleShared   : Pos 3, 1 Bit
   +0x000 Shared           : Pos 4, 60 Bits
   +0x000 Value            : Uint8B
   +0x000 Ptr              : Ptr64 Void
```

Les applications Windows interagissent avec des verrous en lecture/écriture à l'aide des interfaces que voici.

- *AcquireSRWLockExclusive* Acquiert un verrou en mode écriture (mode exclusif).
- *AcquireSRWLockShared* Acquiert un verrou en mode lecture (accès partagé)
- *InitializeSRWLock* Initialise un verrou en lecture/écriture.
- *ReleaseSRWLockExclusive* Relâche un verrou qui a été acquis par AcquireSRWLockExclusive.
- *ReleaseSRWLockShared* Relâche un verrou qui a été acquis par AcquireSRWLockShared.
- *TryAcquireSRWLockExclusive* Tente l'acquisition d'un verrou en mode écriture (accès exclusif).
- *TryAcquireSRWLockShared* Tente l'acquisition d'un verrou en mode lecture (accès partagé).

Contrairement à d'autres mécanismes de synchronisation dans Windows, qui peuvent pour certains avoir tendance à donner la priorité aux threads lecteurs ou rédacteurs (l'un ou l'autre), les verrous lecture-écriture ne favorise aucune de ces catégories. Les performances sont par conséquent identiques dans les deux configurations.

Mutex

Entre autres figures du contrôle de l'accès concurrent à des ressources partagées, le système d'exclusion mutuelle nommé mutex est sans aucun doute le plus évident à appréhender, et sur le plan pratique le plus facile à mettre en oeuvre. Un mutex (de l'anglais mutual exclusion) est une primitive (au sens Windows un objet) utilisée de sorte à éviter que les ressources partagées d'un système ne soient utilisées en même temps, facilitant de la sorte la synchronisation entre processus.

La structure même d'un mutex, sa nature intrinsèque, lui donne deux états : libre ou verrouillé. Lorsqu'un mutex est verrouillé par un thread, on dit que ce dernier *tient* le mutex. Un mutex ne peut être tenu que par un seul thread à la fois, en conséquence de quoi lorsqu'un thread demande à verrouiller un mutex déjà tenu par un autre thread, le premier est bloqué jusqu'à ce que le mutex soit libéré.

Un mutex est toujours et de façon inconditionnelle associé à une ressource - l'objet en lui même n'ayant autrement dit pas grand intérêt. Il peut en l'occurence s'agir d'une variable partagée, mais également de structures plus complexes, comme par exemple une base de données ou un fichier.

Comme tant d'autres primitives orientées synchronisation, chaque mutex existe dans la perspective Windows sous la forme d'un objet à état, signalé ou non, qui se rapporte ici essentiellement au fait que le mutex ait un possesseur en titre. L'état d'un objet mutex est signalé tandis que le ledit objet n'appartient à aucun thread, et non signalé autrement - un mutex entre en d'autres termes dans l'état non signalé qu'à la condition d'être possédé par un thread.

Le tableau suivant liste les principales fonctions utilisables avec des mutex.

Tableau 7.7. Fonctions Windows relatives aux mutex

Fonction	Description
CreateMutex	Crée et initialise un objet mutex avec ou sans nom.
CloseHandle	Ferme un handle de mutex.

WaitForSingleObject	Attend que l'appartenance d'un seul objet mutex lui soit accordée.
WaitForMultipleObjects	Attend que l'appartenance d'un ou plusieurs objets mutex lui soit accordée.
ReleaseMutex	Libère un objet mutex.

La propriété (comprendre l'appartenance) d'un mutex à tel ou tel thread peut être concédée immédiatement après la création, soit a posteriori par l'intermédiaire des fonctions Windows d'attente (WaitForSingleObject et consorts), lesquelles mettront en sommeil le thread appelant jusqu'à ce que les conditions propices soient réunies. Un mutex ne peut être utilisé que par un seul thread à la fois, les autres devant attendre que le système leur permette de se l'approprier.

Appelez la fonction CreateMutex afin de donner lieu à un mutex. Le thread de création peut spécifier un nom pour l'objet lors de sa création, ou bien au contraire le laisser dans l'anonymat. Il est impératif pour protéger les ressources partagées entre d'utiliser des objets mutex nommés.

Les mutex supportent l'acquisition récursive ; en conséquence de quoi un thread peut appeler une fonction d'attente plusieurs fois sans bloquer sa propre exécution. Le système ne bloque pas le thread propriétaire afin d'éviter tout dead lock, mais le thread doit toujours se détacher de la propriété d'un mutex autant de fois que celui-ci a su satisfaire une attente.

Chaque objet de type mutex est représenté dans les couches inférieures du système par le biais d'une structure KMUTANT.

```
lkd> dt nt!_KMUTANT
   +0x000 Header           : _DISPATCHER_HEADER
   +0x018 MutantListEntry  : _LIST_ENTRY
   +0x028 OwnerThread      : Ptr64 _KTHREAD
   +0x030 Abandoned        : UChar
   +0x031 ApcDisable       : UChar
```

- *État abandonné (Abandoned)* Valeur booléenne qui, du moment qu'elle est armée, indique que le thread détenteur du mutex a pris fin sans avoir eu l'occasion de le délivrer. Voir service NtQueryMutant (MutantInformationClass = MutantBasicInformation) ; routine KeReleaseMutant.

- *Entête (Header)* Entête d'objet dispatcher commun duquel le mutex tire ses capacités de synchronisation.

- *Propriétaire (OwnerThread)* Pointeur vers le bloc KTHREAD du thread propriétaire de l'objet. Voir routine KeInitializeMutant (paramètre InitialOwner).

Un thread peut lors de la création d'un mutex se présenter comme étant son propriétaire initial, et auquel cas s'en emparer immédiatement. Le mutex ne sera dans cette éventualité pas signalé à un autre thread tant que le thread ayant engendré le mutex ne l'aura pas libéré. Autrement, l'objet est créé à l'état signalé, signe que celui-ci est disponible pour qui le souhaite. L'attribut OwnerThread de la structure KMUTANT fait référence au bloc KTHREAD du propriétaire d'un mutex. Cette information est notamment utile du fait qu'un mutex doit être libéré par le même thread l'ayant obtenu. Si tel n'est pas le cas, le système génère une exception.

Windows implémente la primitive mutex de synchronisation à même le système d'exploitation, ce qui en fait alors un outil puissant mais lourd (chaque accès est un appel système). Pour synchroniser des accès entre threads à l'intérieur d'un même processus, il existe des variantes plus légères, comme les sections critiques.

Tableau 7.8. MUTANT_INFORMATION_CLASS

ID	Classe	Type de données associé
0x0000	MutantBasicInformation	MUTANT_BASIC_INFO

```
typedef struct _MUTANT_BASIC_INFORMATION {
    LONG CurrentCount;
    BOOLEAN OwnedByCaller;
    BOOLEAN AbandonedState;
```

```
} MUTANT_BASIC_INFORMATION, *PMUTANT_BASIC_INFORMATION;
```

Tableau 7.9. Valeurs initiales des champs de la structure MUTANT_BASIC_INFO

Champ	Valeur initiale
AbandonedState	KMUTANT Abandoned
CurrentCount	KMUTANT Header.SignalState
OwnedByCaller	KMUTANT OwnerThread

Mutex noyau

Comme leurs homologues mode utilisateur en ce qui concerne les applications, les mutex noyau offrent aux composants de l'exécutif et aux pilotes de périphérique une méthode simple (du reste pas nécessairement la meilleure) afin de sérialiser les accès à une ressource partagée.

Quand un thread prend le contrôle d'un objet mutex, le système d'exploitation interrompt automatiquement la livraison des APC mode noyau normaux en augmentant l'IRQL du processeur à APC_LEVEL. Dans la pratique, cela signifie que tout thread en possession d'un mutex ne peut se voir préempté que via l'émission d'une APC noyau spéciale.

La liste qui suit énumère les routines prévues pour une utilisation avec des objets mutex.

- *KeInitializeMutex* Initialise un objet mutex.
- *KeReadStateMutex* Retourne l'état de signal d'un mutex.
- *KeReleaseMutex* Libère un objet mutex.
- *Get/SetKernelObjectSecurity* Gère la sécurité des mutex.

Compte tenu des restrictions les accompagnant, les mutex restent parmi les composants de Windows exécutés en mode noyau d'une utilisation relativement confidentielle. On trouve ce genre de mécanisme ainsi surtout dans le contexte de pilotes de haut niveau, par exemple pilotes de système de fichiers, qui emploient des threads auxiliaires exécutif.

Mutex abandonné

Quand un thread prend fin tandis qu'il se trouvait en possession d'un mutex, on dit de ce dernier qu'il est *abandonné*. A l'échelle du système, un mutex abandonné peut indiquer l'arrêt soudain - en tout cas imprévu - d'une application, ou un défaut parmi la programmation de celle-ci. Cela a dans tous les cas pour effet de laisser en mémoire des données a la cohérence suspecte, qu'il faut par conséquent examiner avec circonspection. Lorsqu'une application s'arrête de façon inopinée avec entre les mains un mutex, l'objet est automatiquement relâché par le système d'exploitation, ce sous le compte du thread fautif. Le prochain thread réclamant le mutex est alors informé de la situation - à quoi correspond le code de retour WAIT_ABANDONED dans les fonctions d'attente - et peut continuer, idéalement en prenant soin de vérifier l'intégrité des structures de données protégées par le mutex.

Le fait qu'un mutex ait ou n'ait pas été abandonné est rendu visible par l'attribut Abandoned du la structure correspondante (bloc KMUTANT) pour représenter l'objet.

```
lkd> dt nt!_KMUTANT Abandoned fffffa80037792c0
   +0x030 Abandoned : 0x1 ''
```

Les thread noyau ont la possibilité d'abandonner un mutex par le biais du paramètre Abandoned de la routine KeReleaseMutant. Relacher un mutex en l'abandonnant permet de s'assurer que l'objet est retiré de la liste des objets mutant que le thread possède (KTHREAD MutantListHead).

Tableau 7.10. Éléments auxiliaires sous-jacents à l'abandon d'un mutex

Elément	Type	Lieu	Autres références
AbandonedState	BOOLEAN	MUTANT_BASIC_INFORMATION	ZwQueryMutant
Abandoned	BOOLEAN	KMUTANT	KeReleaseMutant
Abandoned	BOOLEAN	KeReleaseMutant	KMUTANT Abandoned

Mutex rapides

Les mutex rapides, appelés aussi mutex exécutifs, offrent une alternative sérieuse aux objets mutex, dont ils se différencient à plusieurs niveaux :

- Les mutex rapides n'honorent pas l'acquisition récursive.
- Ils protègent du code s'exécutant à un IRQL inférieur ou égal au niveau APC (APC_LEVEL).
- Les fonctions définies pour leur acquisition donnent lieu à une configuration IRQL du processeur qui désactive à la fois les APC mode noyau normales et les APC mode noyau spéciales.

Les mutex rapides tirent leur efficacité du fait que les étapes d'acquisition et de libération se rapportant à de telles primitives sont optimisées pour les cas où il n'y a pas de compétition. Ainsi, lorsqu'un thread réclame un mutex, le noyau commence par tester et initialiser un compteur qui indique combien de threads possèdent ou attendent le mutex. Si aucun autre thread ne possède le mutex, aucune action supplémentaire n'est requise. A l'inverse, si un autre thread possède déjà le mutex, le noyau force le thread appelant à attendre sur un événement de synchronisation apparenté audit mutex.

L'exécutif définit trois fonctions pour l'acquisition de mutex rapides : ExAcquireFastMutex, ExTryToAcquireFastMutex et ExAcquireFastMutexUnsafe. Les deux premières bloquent entièrement la livraison des APC, en augmentant l'IRQL du processeur à `APC_LEVEL` ; la troisième exige en tant que condition contractuelle la désactivation préalable de la livraison des APC mode noyau normaux, chose qui se concrétise soit en appelant KeRaiseIrql pour configurer l'IRQL au niveau APC, soit en appelant KeEnterCriticalRegion.

Le tableau qui suit dresse l'inventaire des routines utilisables conjointement aux mutex rapides.

Routine	Description
ExAcquireFastMutex	Revendique la propriété d'un mutex rapide
ExAcquireFastMutexUnsafe	Revendique la propriété d'un mutex rapide, après désactivation de la livraison des APC mode noyau normaux
ExInitializeFastMutex	Initialise un mutex rapide
ExReleaseFastMutex	Libère un mutex rapide
ExReleaseFastMutexUnsafe	Libère un mutex sans réactiver la livraison des APC
ExAcquireFastMutexUnsafe	Revendique la propriété d'un mutex s'il est possible de le faire sans attendre

Chaque mutex rapide est en interne représenté par une structure `FAST_MUTEX`, constituée des attributs que voici.

```
lkd> dt nt!_FAST_MUTEX
   +0x000 Count            : Int4B
   +0x008 Owner            : Ptr64 Void
   +0x010 Contention       : Uint4B
   +0x018 Event            : _KEVENT
   +0x030 OldIrql          : Uint4B
```

- **Compteur de compétition (Contention)** Indique le nombre nombre de fois où des threads ont dû patienter avant de pouvoir acquérir un mutex. Valeur essentiellement statistique, sans influence fonctionnelle directe.
- **Compteur (Count)** Reflète l'état d'un mutex : libre ou détenu. Sa valeur initiale (établie au cours de la routine `ExInitializeFastMutex`) est 1, ce qui correspond à un mutex disponible. Lorsqu'un thread en réclame la

possession, s'ensuit une opération de décrémentation (`InterlockedDecrement`). Si le résultat est nul, le thread obtient le mutex. Une valeur négative (-1, -2, etc.) signifie au contraire que le mutex est déjà détenu, et que d'autres threads attendent sa libération. Chaque acquisition manquée décrémente à nouveau le compteur pour refléter le nombre de threads en attente. Lorsqu'un thread libère le mutex, le compteur est revu à la hausse (il est incrémenté) ; s'il demeure négatif, cela indique que d'autres threads doivent encore être réveillés.

- **Événement (Event)** Objet événement dispatcher sur lequel se mettent en attente les threads n'ayant pu acquérir immédiatement un mutex rapide.

- **Ancien IRQL (OldIrql)** Les procédures d'acquisition ayant trait aux mutex rapides donnent généralement lieu à une modification de la configuration IRQL du processeur (seules les routines XxxUnsafe ne sont pas concernées). Aussi, le noyau enregistre ce niveau lors de l'acquisition (ExAcquireFastMutex, ExTryToAcquireFastMutex) et le restaure lors la libération (ExReleaseFastMutex).

- **Propriétaire (Owner)** Pointeur vers le bloc `KTHREAD` du thread actuellement détenteur d'un mutex. En pratique, un tel attribut permet de concrétiser une relation de possession stricte : seul le thread ayant acquis un mutex est autorisé à le libérer. Voir routine ExTryToAcquireFastMutex.

Le tableau suivant fournit un comparatif des différentes formes de mutex, mettant par la même occasion en lumière les forces et les faiblesses de chacune d'entre elles.

Tableau 7.11. Comparatif sur les mutex

	Mutex	**Mutex rapides**	**Mutex gardés**
Structure de données	`KMUTEX`	`FAST_MUTEX`	`KGUARDED_MUTEX`
Caractéristique générale	Objet noyau dispatcher	Structure noyau légère	Structure noyau optimisée SMP
Usage typique	Synchronisation interprocessus ou interthreads générique	Sections critiques internes au noyau	Verrous hautes performances
Performance sur SMP	Faible	Moyenne	Excellente
Récursivité	Oui	Non	Non
Désactivation APC normales	Oui	Oui	Oui
Désactivation APC spéciales	Oui	Oui/Non	Oui
IRQL maximal d'utilisation	⇐ `APC_LEVEL`	⇐ `APC_LEVEL`	⇐ `SYNCH_LEVEL`

Mutex gardés

La liste qui suit passe en revue les interfaces utilisables avec des mutex gardés :

Routine	Description
`KeAcquireGuardedMutex`	Acquiert un mutex gardé et entre dans la section critique correspondante ; bloque le thread appelant jusqu'à ce que le mutex soit disponible
`KeAcquireGuardedMutexUnsafe`	Acquiert un mutex gardé sans désactiver la préemption de threads ; à utiliser uniquement lorsque l'exécution est déjà protégée
`KeInitializeGuardedMutex`	Configure une structure mutex gardé (`KGUARDED_MUTEX`) en vue d'un usage ultérieur
`KeReleaseGuardedMutex`	Libère un mutex gardé et quitte la section critique correspondante ; réactive, le cas échéant, la préemption des threads
`KeReleaseGuardedMutexUnsafe`	Libère un mutex gardé mais n'interfère pas avec les mécanismes de préemption de threads ; doit être utilisé de manière symétrique à `KeAcquireGuardedMutexUnsafe`
`KeTryToAcquireGuardedMutex`	Tente d'acquérir un mutex gardé sans bloquer l'exécution ; l'acquisition ne réussit que si le verrou est disponible au moment de l'appel

Spinlocks

Un mécanisme naïf de synchronisation employé par le noyau pour réaliser de l'exclusion mutuelle multi processeur est le verrou rotatif (spinlock). Offrant des accès mutuellement exclusifs sur l'ensemble des processeurs, des spinlocks contraignent les threads à se coordonner autour la modification de structures de données globales, et permettent au système de garantir qu'un processeur et un seul exécute une section critique du noyau.

A l'instar de nombreuses autres primitives de synchronisation, un spinlock se trouve à tout moment dans l'un ou l'autre des états suivants : disponible ou pris (verrouillé). Un spinlock pris oblige le noyau à le réclamer encore et encore jusqu'à se l'être approprié. Le terme spinlock vient du fait que le noyau (et par extension le processeur) « boucle » (spin) jusqu'à ce que le verrou soit libéré. (On parle auquel cas d'attente active.) De ce fait, un seul processeur à la fois peut être en possession d'un spinlock.

Le noyau associe à chaque spinlock un IRQL qui est toujours supérieur ou égal au niveau DPC/dispatch. Par conséquent, quand un essaie d'acquérir un spinlock, toutes les autres activités faites à un niveau inférieur à l'IRQL du spinlock sont momentanément stoppées. L'ordonnancement des threads s'effectuant au niveau DPC/dispatch, un thread en possession d'un spinlock ne peut jamais être suspendu (l'IRQL ôtant dans cette configuration au noyau toute possibilité de le faire cesser).

Les portions de code s'exécutant tandis qu'un spinlock est verrouillé ne sont pas préemptibles. Elles doivent par conséquent être les plus brèves possibles de façon à libérer rapidement le verrou et, compte tenu de la configuration IRQL à laquelle elles sont rattachées, ne pas être la cause d'exceptions et de défauts de page. Tout manquement à ces deux dernières règles serait immédiatement sanctionné par un effondrement du système.

L'implémentation Windows des spinlocks, comme celle d'autres systèmes d'exploitation, s'appuie sur les dispositifs offerts à cet égard par l'architecture processeur sous-jacente. Dans nombre d'architectures, les spinlocks sont mis en oeuvre à l'aide d'une opération matérielle de test-et-initialisation, lequelle teste la valeur d'une variable verrou et acquiert le verrou en une instruction atomique. Procéder ainsi empêche un second thread de s'emparer du verrou entre le moment où le premier teste la variable et le moment où il acquiert le verrou.

Les jeux d'instruction des architectures x86 et x64 intègrent une commande spéciale en entier dévolue aux spinlocks : PAUSE, qui indique au processeur que la séquence d'instructions en cours d'exécution l'est dans le cadre d'une attente active. Cette information est notamment utile pour (1) diminuer les pénalités induites, et (2), réduire la consommation d'énergie.

De nombreux composants parmi la sphère noyau Windows exigent des spinlocks, dont les pilotes. Par exemple, la plupart des pilotes qui interagissant avec des équipements emploie des spinlocks pour garantir que les registres de périphérique et autres structures de données globales ne sont accessibles qu'à une seule partie d'un pilote (et depuis un seul processeur) à la fois.

Récursion. Tenter d'acquérir un spinlock de manière récursive provoque une situation d'inter blocage (deadlock). Dans un tel scénario, un thread qui possède le verrou boucle dans l'attente que celui-ci satisfasse une nouvelle fois son attente, ce qui est ici chose impossible vu que le thread qui réclame le verrou est en même temps le seul capable de le libérer, et ainsi mettre fin à cette attente.

Tableau 7.12. Routines concernant les spinlocks

KeAcquireSpinLock	S'approprie un spinlock.
KeAcquireSpinLockAtDpcLevel	S'approprie un spinlock au niveau DPC/dispatch.
KeInitializeSpinLock	Initialise un spinlock.
KeReleaseSpinLock	Libère un spinlock.
KeTestSpinLock	Vérifie la disponibilité d'un spinlock

Accessibilité des spinlocks Comme les structures qu'ils protègent, les spinlocks résident dans la mémoire globale du système, les rendant ainsi visibles et utilisables par tous les processeurs. En pratique, une telle disposition permet aux routines de synchronisation du noyau d'y accéder indépendamment du contexte

d'exécution, y compris à des niveaux d'interruption élevés, garantissant ainsi une coordination rapide entre les différentes unités de calcul.

Spinlocks en file

Un type spécial de spinlock, appelé spinlock en file (*queued spinlock*) ou éventuellement spinlock numéroté (*numbered spinlock*), offre une meilleur évolutivité (scalability) qu'un spinlock standard sur les systèmes multi processeur. En règle générale, Windows emploie l'un ou l'autre de ces mécanismes suivant le degré estimé de compétition pour le verrou : spinlock standard s'il est faible, spinlock en file autrement.

Par contraste avec leurs homologues standard, qui résident dans la mémoire globale et sont alloués à la demande, les spinlocks en file s'appuient sur des portions préétablies de l'espace d'adressage et interagissent de prime abord avec des informations locales au processeur. Quand un processeur réclame un spinlock en file déjà retenu, sa demande est enregistré dans une file associée au spinlock. Un processeur en attente d'un spinlock occupé vérifie le statut non du spinlock lui-même mais d'un flag de niveau processeur que le processeur détenteur du spinlock arme quand il le libère. Suite à quoi le processeur en attente peut désormais aller de l'avant et s'emparer du spinlock.

Un avantage majeur offert par les spinlocks en file est qu'ils appliquent une stratégie de type FIFO parmi les processeurs qui concurrent pour un même verrou (tandis que les verrous spinlocks ne le font pas).

Windows définit un certain de spinlocks en file globaux, en stockant des pointeurs vers ceux-ci dans un tableau contenu dans le PCR de chaque processeur. Voir champ LockQueue de la structure KPRCB.

Le tableau suivant fait le lien entre numéros de file d'attente et spinlocks réels.

Tableau 7.13. Spinlocks numérotés

Index	Constante	Verrou
00	LockQueueDispatcherLock	KiDispatcherLock
01	LockQueueExpansionLock	-
02	LockQueuePfnLock	MmPfnLock
03	LockQueueSystemSpaceLock	MmSystemSpaceLock
04	LockQueueVacbLock	CcVacbSpinLock
05	LockQueueMasterLock	CcMasterSpinLock
06	LockQueueNonPagedPoolLock	NonPagedPoolLock
07	LockQueueIoCancelLock	IopCancelSpinLock
08	LockQueueWorkQueueLock	CcWorkQueueSpinLock
09	LockQueueIoVpbLock	IopVpbSpinLock
10	LockQueueIoDatabaseLock	IopDatabaseLock
11	LockQueueIoCompletionLock	IopCompletionLock
12	LockQueueNtfsStructLock	NtfsStructLock
13	LockQueueAfdWorkQueueLock	AfdWorkQueueSpinLock
14	LockQueueBcbLock	CcBcbSpinLock
15	LockQueueMmNonPagedPoolLock	MmNonPagedPoolLock
16	LockQueueUnusedSpare16	-
17	LockQueueTimerTableLock	-

Chaque spinlock en file est au niveau noyau représenté par une structure KSPIN_LOCK_QUEUE.

Tableau 7.14. Routines concernant les spinlocks en file

KeAcquireQueuedSpinLock	Acquiert un spinlock en file
KeReleaseQueuedSpinLock	Libère un spinlock en file
KeTryToAcquireQueuedSpinLock	-

Tableau 7.15. Routines concernant les spinlocks en file

KIRQL	KeAcquireQueuedSpinLock (KSPIN_LOCK_QUEUE_NUMBER Number)
VOID	KeReleaseQueuedSpinLock (KSPIN_LOCK_QUEUE_NUMBER Number, KIRQL OldIrql)

Spinlocks en file dynamiques

En plus des spinlocks en file statiques qui sont définis globalement, Windows gère les spinlocks en file alloués dynamiquement.

Tableau 7.16. Routines concernant les spinlocks en file dynamiques

Routine	Description
KeAcquireInStackQueuedSpinLock(AtDpcLevel)	Acquiert un spinlock en file dynamique
KeReleaseInStackQueuedSpinLock(FromDpcLevel)	Libère un spinlock en file dynamique

Spinlocks lecture/écriture

Lorsque les opérations ciblant des données partagées tendent à interroger (lire) plus qu'à manipuler (écrire) des données, une version spécialisée du spin lock, appelée spin lock lecture/écriture, peut être avantageuse, dans la mesure où elle fournit des accès exclusifs (en écriture) et des accès partagés (en lecture), offrant ainsi la possibilité à de multiples threads lecteurs d'entrer dans une même section critique du noyau.

Par nature, les spinlocks lecture/écriture se prêtent particulièrement bien aux scénarios dont le déroulement implique un nombre important de lecteurs, mais peu d'écrivains. Chaque représentant d'une telle primitive (comprendre chaque spinlock lecture/écriture) peut avoir plusieurs lecteurs en même temps mais un seul écrivain, et il ne peut pas y avoir de lecteurs quand il y a un écrivain.

Si un thread lecteur a besoin d'acquérir un spin lock pour l'accès partagé, et que le dit verrou est déjà détenu exclusivement par un autre thread, le lecteur doit d'abord attendre que cet autre libère le verrou. Dans la même veine, un thread qui souhaite avoir le contrôle exclusif d'un verrou, si ce verrou est déjà possédé en accès partagé par un ou plusieurs autres threads, doit attendre que tous aient libéré le verrou. Néanmoins, l'attente d'un thread écrivain prive d'autres threads lecteurs de voir leur attente satisfaite. Au lieu de cela, un lecteur qui a besoin d'acquérir le verrou que l'écrivain attend doit d'abord attendre que l'écrivain ait obtenu puis libéré le verrou.

Au contraire des spinlocks standard ou en file (statiques ou non), qui dépendent du noyau, la prise en charge des spinlocks de type lecture/écriture est assurée au niveau de l'exécutif.

Spinlocks d'interruption

Quand un périphérique génère une interruption et rompt la séquence d'exécution en cours sur un processeur, le gestionnaire d'interruption du noyau transfère le contrôle à une routine spécialisée (ISR, *Interrupt Service Routine*), alimentée par des données volatiles, et dont l'exécution se fait à l'IRQL supérieur au niveau DPC/dispatch. Dans le même temps, les mécanismes de gestion standard des spinlocks ne permettent d'élever l'IRQL qu'à DPC/dispatch, ce qui n'est pas suffisant pour se protéger des interruptions. (On peut par exemple penser au cas où une donnée d'un pilote de périphérique sur un processeur devient manœuvré par une routine de service d'interruption du même pilote en cours d'exécution sur un autre processeur). Pour prendre en compte cette dimension du problème, Windows intègre parmi chaque objet interruption un verrou spécial, dit spinlock d'interruption.

Les composants mode noyau emploient pilotes emploient un spinlock d'interruption pour synchroniser leurs accès à des ressources matérielles ou des données globales partagés avec un service d'interruption. Tout code de section critique qui s'exécute en étant détenteur d'un verrou le fait à l'IRQL égal à DIRQL, ce qui empêche l'entrée dans la routine de traitement de l'interruption. L'IRQL masque les mécanismes d'interception par le processeur courant, le verrou empêche l'interruption par un autre processeur.

Le noyau rend visible deux approches concernant le code d'acquisition et libération de verrous associés aux objets interruption. Il exporte pour la première la routine KeAcquireInterruptSpinLock, permettant de synchroniser une section critique et de s'emparer explicitement du verrou. En sus, deuxième approche, le système fournit une routine de synchronisation spéciale appelée KeSynchronizeExecution, capable de synchroniser une fonction entière.

Pushlocks

Les pushlocks sont un dispositif de verrouillage introduit dans une version serveur de Windows (Windows Server 2003), principalement utilisé en place des spinlocks pour protéger les structures de données principales du noyau. Le mécanisme s'appuie sur l'objet interne de synchronisation KGATE, une version optimisée de l'objet événement de base qui permet aux threads de n'attendre que quand il y a compétition pour le verrou. L'avantage des pushlocks par rapport aux spinlocks - ces derniers devant être acquis exclusivement pour toutes les opérations - est la possibilité pour un pushlock d'être partagé par plusieurs lecteurs, et de n'être acquis de façon exclusive que si un thread a besoin de modifier une donnée.

Il existe deux types de pushlocks : normal et compatible cache. Lorsqu'un thread acquiert un pushlock normal, le pushlock est marqué comme étant possédé s'il ne l'est pas déjà. Si le pushlock est possédé de façon exclusive ou si le thread veut acquérir exclusivement le verrou alors qu'un autre thread le possède en mode partagé, le thread alloue un bloc d'attente sur sa pile, initialise un objet événement dans le bloc d'attente et ajoute le bloc à la liste d'attente associée au pushlock. Lorsqu'un thread libère le pushlock, il réveille le prochain attendeur en signalant l'événement du bloc d'attente de l'attendeur.

L'utilisation de push locks revêt plusieurs avantages. Primo, dans le scénario exclusif, de tels vecteurs emploient une politique de file d'attente FIFO synchronisée, ce qui élimine les situations de compétition et améliore les accès au cache (*cache trashing*). Secondo, chaque push lock a un taux d'occupation mémoire directement lié à la taille du mot machine (4 octets sur un système 32 bits, 8 octets sur un système 64 bits). Les primitives d'acquisition et de libération concernant les push locks peuvent de la sorte prendre appui sur de simples opérations atomiques (InterlockedCompareExchangePointer, InterlockedBitTestAndReset, et consorts). (Notez que cela ne concerne que les cas non litigieux.) En plus de cela, dernier avantage, Windows optimise automatiquement l'ordre de la liste des threads candidats sur une base partagée à un même push lock.

Un pushlock se traduit en interne par une structure de données de type EX_PUSH_LOCK, laquelle se présente comme suit :

```
lkd> dt nt!_EX_PUSH_LOCK
   +0x000 Locked           : Pos 0, 1 Bit
   +0x000 Waiting          : Pos 1, 1 Bit
   +0x000 Waking           : Pos 2, 1 Bit
   +0x000 MultipleShared   : Pos 3, 1 Bit
   +0x000 Shared           : Pos 4, 60 Bits
   +0x000 Value            : Uint8B
   +0x000 Ptr              : Ptr64 Void
```

Le bit Locked est armé dès lors qu'un thread est entré en possession du verrou, autant en mode exclusif que partagé. Le bit Waiting indique qu'il existe au moins un thread en attente du verrou. Le champ Shared garde trace du nombre d'acquisitions partagées en attente sur le verrou. Si le bit Locked est positionné et que le masque binaire exprimée par l'attribut Shared vaut 0 (aucun attendeur), le verrou est détenu exclusivement par un thread.

Événements

Contrairement aux mécanismes de synchronisation vus jusqu'ici, qui induisent une notion explicite d'appartenance pour protéger l'accès à des données partagées, les événements synchronisent le flux d'exécution.

Les événements sont des objets du noyau sur lesquels les threads ont la possibilité de se synchroniser dans l'attente de conditions spécifiques, par exemple une tâche accomplie ou la disponibilité de données pour les

lecteurs potentiels. Chaque événement comporte des informations d'état qui, selon que les circonstances attendues soient réunies ou non, contrôlent (à travers le système d'exploitation) les threads enregistrés à son compte, qui peuvent ainsi soit continuer de se tenir prêt, soit reprendre leurs traitements là où ils l'avaient suspendu. Windows utilise des événements, entre autres, pour certains cas d'E/S fichier, certaines situations d'achèvements, ou encore pour notifier d'une situation particulière dans le système.

Concrètement, les événements se basent sur une logique à deux états mutuellement exclusifs : signalé et non signalé. Ces états permettent aux threads d'une application d'attendre qu'un événement soit signalé.

Le support des événements fourni par Windows se décompose en deux types a priori bien distincts : événement de notification (appelé événement à réinitialisation manuelle dans le sous-système Windows) et événement de notification (événement à réinitialisation automatique). La principale différence entre les instances des deux catégories tient au fait que les événements à réinitialisation automatique se réinitialisent automatiquement (autrement dit, deviennent non signalées) immédiatement après avoir satisfait les attentes les concernant. Par contraste, les événements à réinitialisation manuelle restent signalés jusqu'à ce qu'une primitive les remettent à leur état initial. Notez que la nuance entre repositionnement manuel et repositionnement automatique est plus importante qu'il n'y parait. Ainsi, si plusieurs threads attendent sur un même événement événement à repositionnement automatique, il est auquel cas nécessaire de signaler l'événement une fois pour chaque thread. Dans le cas d'un événement à repositionnement manuel, il suffit de signaler une seule fois l'événement pour libérer tous les threads en attente.

Événements nommés et anonymes . À l'instar d'autres objets du noyau, les événements peuvent être dotés d'un nom leur conférant une visibilité au-delà des frontières d'un domaine d'application et de processus. La prise en charge d'événements nommés se fait au niveau des fonctions CreateEvent et CreateEventEx.

Structures de données. Compte tenu de la relative simplicité des exigences auxquelles sont censés répondre les événements, c'est le noyau, et non les sphères exécutives qui gravitent autour, qui en assume la formulation et la quasi-totalité de la prise en charge. Chaque événement se présente ainsi sous la forme d'une structure KEVENT, laquelle sert du reste uniquement d'armature à un entête dispatcheur (structure DISPATCHER_HEADER). Une parfaite illustration de ceci est montrée par la commande dt du débogueur noyau.

Les informations établies à l'échelle d'objets dispatcher n'ont pas toutes un rapport direct avec les techniques de synchronisation par événements. Le texte suivant met en relief quels attributs retenir de la structure DISPATCHER_HEADER pour ce qui concerne la perspective de cette étude.

```
lkd> dt -r nt!_KEVENT
   +0x000 Header           : _DISPATCHER_HEADER
      +0x000 Lock             : Int4B
      +0x000 LockNV           : Int4B
      +0x000 Type             : UChar
      +0x001 Signalling       : UChar
      +0x002 Size             : UChar
      +0x003 Reserved1        : UChar
      +0x000 TimerType        : UChar
      +0x001 TimerControlFlags : UChar
      +0x001 Absolute         : Pos 0, 1 Bit
      +0x001 Wake             : Pos 1, 1 Bit
      +0x001 EncodedTolerableDelay : Pos 2, 6 Bits
      +0x002 Hand             : UChar
      +0x003 TimerMiscFlags   : UChar
      +0x003 Index            : Pos 0, 6 Bits
      +0x003 Inserted         : Pos 6, 1 Bit
      +0x003 Expired          : Pos 7, 1 Bit
      +0x000 Timer2Type       : UChar
      +0x001 Timer2Flags      : UChar
      +0x001 Timer2Inserted   : Pos 0, 1 Bit
      +0x001 Timer2Expiring   : Pos 1, 1 Bit
      +0x001 Timer2CancelPending : Pos 2, 1 Bit
      +0x001 Timer2SetPending : Pos 3, 1 Bit
      +0x001 Timer2Running    : Pos 4, 1 Bit
```

```
+0x001 Timer2Disabled   : Pos 5, 1 Bit
+0x001 Timer2ReservedFlags : Pos 6, 2 Bits
+0x002 Timer2Reserved1  : UChar
+0x003 Timer2Reserved2  : UChar
+0x000 QueueType        : UChar
+0x001 QueueControlFlags : UChar
+0x001 Abandoned        : Pos 0, 1 Bit
+0x001 DisableIncrement : Pos 1, 1 Bit
+0x001 QueueReservedControlFlags : Pos 2, 6 Bits
+0x002 QueueSize        : UChar
+0x003 QueueReserved    : UChar
+0x000 ThreadType       : UChar
+0x001 ThreadReserved   : UChar
+0x002 ThreadControlFlags : UChar
+0x002 CycleProfiling   : Pos 0, 1 Bit
+0x002 CounterProfiling : Pos 1, 1 Bit
+0x002 GroupScheduling  : Pos 2, 1 Bit
+0x002 AffinitySet      : Pos 3, 1 Bit
+0x002 Tagged           : Pos 4, 1 Bit
+0x002 EnergyProfiling  : Pos 5, 1 Bit
+0x002 ThreadReservedControlFlags : Pos 6, 2 Bits
+0x003 DebugActive      : UChar
+0x003 ActiveDR7        : Pos 0, 1 Bit
+0x003 Instrumented     : Pos 1, 1 Bit
+0x003 Minimal          : Pos 2, 1 Bit
+0x003 Reserved4        : Pos 3, 3 Bits
+0x003 UmsScheduled     : Pos 6, 1 Bit
+0x003 UmsPrimary       : Pos 7, 1 Bit
+0x000 MutantType       : UChar
+0x001 MutantSize       : UChar
+0x002 DpcActive        : UChar
+0x003 MutantReserved   : UChar
+0x004 SignalState      : Int4B
+0x008 WaitListHead     : _LIST_ENTRY
   +0x000 Flink            : Ptr64 _LIST_ENTRY
   +0x008 Blink            : Ptr64 _LIST_ENTRY
```

La liste qui vient présente les quelques-uns des attributs de l'entête dispatcher qui font sens dans le contexte des objets événement.

- *État de signal (SignalState)* État de l'objet en tant que primitive de synchronisation. Voir fonctions CreateEvent (paramètre bInitialState), SetEvent et ResetEvent ; routine KeInitializeEvent (paramètre State), KeClearEvent et KeReadStateEvent.

- *Taille (Size)* Taille de la structure représentant un objet événement divisée par 4. Voir routine KeInitializeEvent.

- *Type* Indique si l'événement est de type notification (réinitialisation manuelle) ou de type synchronisation (réinitialisation automatique). Le comportement de l'API d'événements diffère selon le type d'objet événements impliqué. Un objet à réinitialisation manuelle réveille tous les threads en attente et doit être explicitement réinitialisé à l'état non signalé, tandis qu'un objet à réinitialisation automatique réveille uniquement un des threads en attente. Voir routine KeInitializeEvent (paramètre Type).

- *Liste d'attente (WaitListHead)* Liste des threads en attente sur l'événement. Voir routine KeInitializeEvent.

Le tableau suivant répertorie les les principales fonctions utilisables avec des objets événement.

Tableau 7.17. Opérations concernant les évènements

Opération	Fonction	Service
Ouvrir un objet événement	OpenEvent	NtOpenEvent
Créer un objet événement	CreateEvent	NtCreateEvent
Réinitialiser un événement signalé	ResetEvent	NtResetEvent

Opération	Fonction	Service
Signaler un objet événement.	SetEvent	NtSetEvent
Attendre qu'un objet événement passe à l'état signalé	WaitForSingleObject	NtWaitForSingleObject
Attendre que plusieurs objets événement passent à l'état signalé	WaitForMultipleObjects	NtWaitForMultipleObjects
Refermer un handle ouvert sur un objet événement	CloseHandle	NtCloseHandle
		NtSetEventBoostPriority

Tableau 7.18. Mappage générique des droits d'accès aux événements

Droit générique	Signification
GENERIC_ALL	STANDARD_RIGHTS_REQUIRED
	SYNCHRONIZE
	EVENT_QUERY_STATE
	EVENT_MODIFY_STATE
GENERIC_EXECUTE	STANDARD_RIGHTS_EXECUTE
	SYNCHRONIZE
GENERIC_READ	STANDARD_RIGHTS_READ
	EVENT_QUERY_STATE
GENERIC_WRITE	STANDARD_RIGHTS_WRITE
	EVENT_MODIFY_STATE

Tableau 7.19. Autorisations spécifiques aux événements

Constante	Valeur	Description
EVENT_QUERY_STATE	0x0001	Requis pour demander des informations relatives à un objet événement. Controlé par NtQueryEvent.
EVENT_MODIFY_STATE	0x0002	Requis pour modifier l'état de signal d'un objet événement. Controlé par NtClearEvent, NtPulseEvent, NtResetEvent, NtSetEvent et NtSetEventBoostPriority.

Tableau 7.20. EVENT_INFORMATION_CLASS

ID	Classe	Type de données associé
0x0000	EventBasicInformation	EVENT_BASIC_INFORMATION

EVENT_BASIC_INFORMATION. typedef struct EVENTBASICINFORMATION { EVENTTYPE EventType; LONG EventState; } EVENTBASICINFORMATION, PEVENTBASICINFORMATION;

Tableau 7.21. Valeurs initiales des champs de la structure EVENT_BASIC_INFORMATION

Champ	Valeur initiale
EventState	KEVENT Header.SignalState
EventType	KEVENT Header.Type

Evénements à clé

Outre être au coeur des fonctions Windows de section critique, les objets événement à clé jouent un rôle important dans diverses primitives de synchronisation en mode utilisateur, dont les variables de condition (*condition variables*) et les verrous en lecture/écriture légers (Slim Reader Writer Locks).

Tableau 7.22. Opérations concernant les évènements à clé

Opération	Service	Routine
Créer un objet événement à clé	NtCreateKeyedEvent	ObCreateObject

Opération	Service	Routine
Ouvrir un objet événement à clé	NtOpenKeyedEvent	ObOpenObjectByName
Libérer (et par la même occasion signaler) un objet événement à clé	NtReleaseKeyedEvent	
Attendre qu'un objet événement passe à l'état signalé	NtWaitForKeyedEvent	

Tableau 7.23. Autorisations spécifiques aux événements à clé

Constante	Valeur	Description
KEYEDEVENT_WAIT	0x0001	Requis pour attendre un objet événement à clé. Controlé par NtWaitForKeyedEvent.
KEYEDEVENT_WAKE	0x0002	Requis pour signaler un objet événement à clé. Controlé par NtReleaseKeyedEvent.

Tableau 7.24. Mappage générique des droits d'accès à des événements à clé

Droit générique	Signification
GENERIC_ALL	STANDARD_RIGHTS_REQUIRED
	KEYEDEVENT_WAIT
	KEYEDEVENT_WAKE
GENERIC_EXECUTE	STANDARD_RIGHTS_EXECUTE
GENERIC_READ	STANDARD_RIGHTS_READ
	KEYEDEVENT_WAIT
GENERIC_WRITE	STANDARD_RIGHTS_WRITE
	KEYEDEVENT_WAKE

Evénements appariés

Tableau 7.25. Opérations concernant les évènements appariés

Opération	Service	Routine
Créer un objet événement apparié	NtCreateEventPair	ObCreateObject
Ouvrir un objet événement apparié	NtOpenEventPair	ObOpenObjectByName
Signaler la partie basse d'un objet événement apparié	NtSetLowEventPair	
Signaler la partie haute d'un objet événement apparié	NtSetHighEventPair	
Attendre que la partie haute d'un objet événement apparié soit signalé	NtWaitHighEventPair	

Tableau 7.26. Mappage générique des droits d'accès à des événements appariés

Droit générique	Signification
GENERIC_ALL	STANDARD_RIGHTS_REQUIRED
	SYNCHRONIZE
GENERIC_EXECUTE	STANDARD_RIGHTS_EXECUTE
	SYNCHRONIZE
GENERIC_READ	STANDARD_RIGHTS_READ
	SYNCHRONIZE
GENERIC_WRITE	STANDARD_RIGHTS_WRITE
	SYNCHRONIZE

Ressources exécutif

Les ressources exécutif constituent un mécanisme de synchronisation flexible qui fournit des accès exclusifs et des accès en lecture partagés (de multiples lecteurs se partagent des accès en lecture à une même structure).

Comme quelques autres mécanisme noyau de synchronisation, les ressources exécutif exigent la désactivation préalable de la livraison des APC mode noyau normales.

Les ressources ne sont accessibles qu'au sein de la sphère noyau, où ils s'utilisent du reste surtout dans les pilotes de système de fichiers. L'API Windows exporte toutefois des solutions basées sur des principes similaires, par exemple les verrous SRW (Slim Reader/Writer).

Les threads qui attendent d'acquérir une ressource pour un accès partagé le font en interne par le biais d'un sémaphore associé à la ressource (ERESOURCE SharedWaiters), alors que les threads qui attendent une ressource pour un accès exclusif attendent un événement (ERSOURCES ExclusiveWaiters). On utilise un sémaphore à compteur illimité pour les attendeurs partagés dans la mesure où tous peuvent être réveillés et se voir accorder l'accès à la ressource dès disponibilité de celle-ci. À l'inverse, quand un thread réclame l'accès exclusif à une ressource momentanément prise, on se réfère alors à un objet événement de synchronisation (comprendre un objet événement dont le type renvoie sans ambiguïté à de la synchronisation, et non à de la notification), dont la mise à l'état signalé aura pour effet de réveiller un seul des attendeurs.

Par contraste avec de nombreux autres dispositifs de synchronisation en oeuvre au sein de la sphère exécutive, les ressources exécutif n'ont pas de lien direct avec des objets dispatcher, mais se présentent en l'occurence sous la forme de structures de données, allouées depuis le pool non paginé, disposant en l'état de leurs propres services spécialisés de configuration (initialisation), verrouillage, libération, consultation et attente.

La tableau qui suit énumère les routines de support établies en adéquation avec des ressources exécutif.

Tableau 7.27. Routine noyau associées aux ressources exécutif

Routine	Description
ExInitializeResourceLite	Initialise une ressource exécutif
ExAcquireResourceExclusiveLite	Acquiert une ressources exécutif en mode accès exclusif
ExAcquireResourceSharedLite	Acquiert une ressources exécutif en mode accès partagé
ExAcquireSharedStarveExclusive	
ExAcquireSharedWaitForExclusive	
ExConvertExclusiveToSharedLite	
ExDeleteResourceLite	
ExGetExclusiveWaiterCount	Retourne le nombre de threads actuellement en attente sur une ressource pour un accès exclusif
ExGetSharedWaiterCount	Retourne le nombre de threads actuellement en attente sur une ressource pour un accès partagé
ExIsResourceAcquiredExclusiveLite	
ExIsResourceAcquiredSharedLite	
ExReinitializeResourceLite	
ExReleaseResourceForThreadLite	

La liste qui suit décrit les éléments clé de la structure ERESOURCE.

```
lkd>  dt nt!_ERESOURCE
   +0x000 SystemResourcesList : _LIST_ENTRY
   +0x010 OwnerTable       : Ptr64 _OWNER_ENTRY
   +0x018 ActiveCount      : Int2B
   +0x01a Flag             : Uint2B
   +0x01a ReservedLowFlags : UChar
   +0x01b WaiterPriority   : UChar
   +0x020 SharedWaiters    : Ptr64 _KSEMAPHORE
   +0x028 ExclusiveWaiters : Ptr64 _KEVENT
   +0x030 OwnerEntry       : _OWNER_ENTRY
   +0x040 ActiveEntries    : Uint4B
   +0x044 ContentionCount  : Uint4B
   +0x048 NumberOfSharedWaiters : Uint4B
   +0x04c NumberOfExclusiveWaiters : Uint4B
```

```
+0x050 Reserved2        : Ptr64 Void
+0x058 Address          : Ptr64 Void
+0x058 CreatorBackTraceIndex : Uint8B
+0x060 SpinLock         : Uint8B
```

- *Compteur d'actifs (ActiveCount)* Nombre de détenteurs de la ressource. Si le nombre de threads actifs sur la ressource à égal à zéro, cela signifie qu'elle est sans propriétaire ; l'accès à la ressource peut alors être accordée immédiatement. Comme il ne peut y avoir qu'un seul propriétaire exclusif d'une ressource, le compteur d'actif d'une ressource dans ce mode vaut toujours 1. Par contre, le compteur d'actifs est incrémenté d'une unité chaque fois que des threads qui attendent d'acquérir une ressource pour un accès partagé voient leur attente satisfaite.

- *Compteur de compétition (ContentionCount)* Enregistre le nombre de fois que des threads ont essayé d'entrer en possession de la ressource mais ont dû attendre parce qu'elle était déjà prise.

- *Verrouillage exclusif (ExclusiveWaiters)* Événement qu'attendent les threads ayant sollicité l'accès exclusif à une ressource.

- *Nombre d'attendeurs exclusifs (NumberOfExclusiveWaiters)* Nombre de threads qui attendent d'acquérir la ressource pour un accès en mode exclusif.

- *Nombre d'attendeurs partagés (NumberOfSharedWaiters)* Nombre de threads qui attendent d'acquérir la ressource pour un accès en mode partagé.

- *Verrouillage partagé (SharedWaiters)* Sémaphore qu'attendent les threads ayant sollicité un accès partagé à une ressource.

- *Liste des ressources (SystemResourcesList)* Entrée d'une liste chainée qui permet d'itérer rapidement sur les différentes structures sous-jacentes aux ressources exécutif et de voir où chacune a individuellement été allouée.

Liste des ressources exécutif. Les ressources exécutif sont pour des raisons pratiques organisées sous forme d'une liste chainée. De ce fait, chaque fois qu'une ressource exécutif dépasse la phase de configuration initiale (ExInitializeResourceLite), la structure résultante (ERESOURCE) se voit reliée à une autre par le biais de l'attribut SystemResourcesList. La variable noyau ExpSystemResourcesList spécifie dans ce contexte la tête de liste.

Acquisition récursive. Contrairement à d'autres objets noyau de synchronisation, les ressources de l'exécutif peuvent être acquises récursivement, une règle importante en la matière étant de conserver la parité entre les différentes opérations d'acquisition et de libération (comprendre qu'un thread qui a obtenu plusieurs fois une ressource doit la relâcher autant de fois que le système lui a confié sa propriété.

Visualisation des ressources exécutif

La commande !locks du débogeur noyau affiche des informations concernant les ressources exécutif.

```
lkd> !locks
**** DUMP OF ALL RESOURCE OBJECTS ****
KD: Scanning for held locks....

Resource @ 0xffffa7049dc2c468    Exclusively owned
     Threads: ffffa704a06af040-01<*>
KD: Scanning for held locks.............
48061 total locks, 1 locks currently held
```

Par défaut, la commande énumère seulement les ressources exécutif actuellement possédées ; l'option -d permet d'énumérer l'ensemble des ressources exécutif.

Pour examiner les détails d'un objet ressource en particulier, y compris le thread qui possède la ressource, spécifiez le commutateur -v suivi de l'adresse de la ressource.

```
lkd> !locks -v 0xffffa7049dc2c468

Resource @ 0xffffa7049dc2c468    Exclusively owned
     Threads: ffffa704a06af040-01<*>

     THREAD ffffa704a06af040  Cid 0004.23b8  Teb: 0000000000000000 Win32Thread: 0000000000000000 WAIT: (Executive)
 KernelMode Non-Alertable
         ffffec0ace1cf598  SynchronizationEvent
     Not impersonating
     DeviceMap                 ffffb9894c8134e0
     Owning Process            ffffa7049be65380       Image:         System
     Attached Process          N/A            Image:         N/A
     Wait Start TickCount      1927047        Ticks: 6636595 (1:04:48:16.796)
     Context Switch Count      24701          IdealProcessor: 0
     UserTime                  00:00:00.000
     KernelTime                00:00:01.390
     Win32 Start Address nt!ExpWorkerThread (0xfffff801640ae730)
     Stack Init ffffec0ace1cfc10 Current ffffec0ace1cf1c0
     Base ffffec0ace1d0000 Limit ffffec0ace1c9000 Call 0000000000000000
     Priority 13 BasePriority 12 PriorityDecrement 0 IoPriority 2 PagePriority 5
GetContextState failed, 0x80004001
Unable to get current machine context, HRESULT 0x80004001
     Child-SP          RetAddr           Call Site
     ffffec0a`ce1cf200 fffff801`640318c2 nt!KiSwapContext+0x76
     ffffec0a`ce1cf340 fffff801`64030f54 nt!KiSwapThread+0x3f2
     ffffec0a`ce1cf3e0 fffff801`640306f5 nt!KiCommitThreadWait+0x144
     ffffec0a`ce1cf480 fffff801`6411820b nt!KeWaitForSingleObject+0x255
     ffffec0a`ce1cf560 fffff801`64054573 nt!ExfWaitForRundownProtectionRelease+0xfb
     ffffec0a`ce1cf5d0 fffff801`670cd01c nt!ExWaitForRundownProtectionRelease+0x23
     ffffec0a`ce1cf600 00000000`00000000 0xfffff801`670cd01c


1 total locks
```

Objets noyau dispatcher

Une large majorité des primitives de synchronisation implémentées au niveau de l'exécutif Windows le sont par l'intermédiaire de structures noyau regroupées sous l'appellation d'*objets dispatcher*, lesquels renferment l'ensemble des moyens et des stratégies internes appropriés en la matière. Tout objet visible de l'utilisateur qui gère de la synchronisation encapsule au moins un en-tête dispatcher, combiné sur le plan pratique à une interface unifiée simple et cohérente. Un thread d'une application Windows peut de la sorte de synchroniser avec toutes sortes d'objects : processus, thread, événement, sémaphore, fichier, et bien d'autres.

Les objets dispatcher peuvent être soit dans un état signalé soit dans un état non signalé. Un état signalé indique qu'un objet est disponible et peut dès lors satisfaire toute demande en ce qui concerne sa propriété. Un état non signalé indique qu'un objet n'est pas disponible et qu'un thread bloquera en attendant d'acquérir l'objet.

Un thread peut se synchroniser avec un objet dispatcher en attendant le handle de l'objet. Se trouve ainsi dessinée en creux une relation ténue entre l'état d'un objet dispatcher et l'état d'ordonnancement d'un thread. Lorsque thread se voit confronté à un objet dispatcher non signalé (dont on ne peut donc pas disposer dans l'immédiat), le noyau le suspend et actualise son état d'ordonnancement pour le faire passer de « en cours » (running) à « en attente » (waiting), l'excluant de facto de l'ensemble des threads à considérer pour l'exécution. Le thread est suite à cela ajouté à une liste de threads attendant l'objet, threads dont l'attente ne pourra en définitive prendre fin qu'à la condition que l'objet sur lequel ils veulent se synchroniser subisse un changement d'état, passant de l'état non signalé à l'état signalé.

Le nombre de threads que le noyau sélectionne depuis la file d'attente dépend de la nature de l'objet attendu. Par exemple, dans le cas d'un mutex, un thread seulement verra son attente comblée, étant donné qu'un objet mutex ne peut être verrouillé que par un seul thread à la fois. Pour un objet événement, le noyau sélectionnera tous les threads qui sont en train de l'attendre. L'exemple suivant montre, en partant du principe que la primitive attendue est un objet événement, les interactions entre synchronisation et ordonnancement de thread :

- Un thread mode utilisateur attend le handle d'un objet événement.
- Un autre thread initialise l'événement.
- Le noyau parcourt la liste des threads en attente de l'événement. Si les conditions d'attente d'un thread sont remplis (elles peuvent encore ne pas l'être à ce stade, vu qu'un thread peut attendre plusieurs objets), le noyau modifie l'état du thread pour le faire passer de "en attente" à "prêt".
- Étant donné qu'un nouveau thread est devenu prêt pour l'exécution, le dispatcher réordonnance. S'il trouve un thread en cours d'exécution qui a une priorité inférieure à celle du thread dont l'état vient de basculer à "prêt", il préempte le thread de priorité inférieure et émet une interruption logicielle pour initier un basculement de contexte en faveur du thread plus prioritaire.
- Si aucun thread ne peut être préempté, le dispatcher place le thread prêt dans la liste "prêt" du dispatcher pour ordonnancement ultérieur.

Basé sur un modèle commun, tous les objets noyau dispatcher s'appuient sur la même structure de données, appelée dans ce contexte un entête dispatcheur (dispatcher header), de type DISPATCHER_HEADER.

```
lkd> dt nt!_DISPATCHER_HEADER @$Thread
   +0x000 Lock             : 0n2097158
   +0x000 LockNV           : 0n2097158
   +0x000 Type             : 0x6 ''
   +0x001 Signalling       : 0 ''
   +0x002 Size             : 0x20 ' '
   +0x003 Reserved1        : 0 ''
   +0x000 TimerType        : 0x6 ''
   +0x001 TimerControlFlags : 0 ''
   +0x001 Absolute         : 0y0
   +0x001 Wake             : 0y0
   +0x001 EncodedTolerableDelay : 0y000000 (0)
   +0x002 Hand             : 0x20 ' '
   +0x003 TimerMiscFlags   : 0 ''
   +0x003 Index            : 0y000000 (0)
   +0x003 Inserted         : 0y0
   +0x003 Expired          : 0y0
   +0x000 Timer2Type       : 0x6 ''
   +0x001 Timer2Flags      : 0 ''
   +0x001 Timer2Inserted   : 0y0
   +0x001 Timer2Expiring   : 0y0
   +0x001 Timer2CancelPending : 0y0
   +0x001 Timer2SetPending : 0y0
   +0x001 Timer2Running    : 0y0
   +0x001 Timer2Disabled   : 0y0
   +0x001 Timer2ReservedFlags : 0y00
   +0x002 Timer2Reserved1  : 0x20 ' '
   +0x003 Timer2Reserved2  : 0 ''
   +0x000 QueueType        : 0x6 ''
   +0x001 QueueControlFlags : 0 ''
   +0x001 Abandoned        : 0y0
   +0x001 DisableIncrement : 0y0
   +0x001 QueueReservedControlFlags : 0y000000 (0)
   +0x002 QueueSize        : 0x20 ' '
   +0x003 QueueReserved    : 0 ''
   +0x000 ThreadType       : 0x6 ''
   +0x001 ThreadReserved   : 0 ''
   +0x002 ThreadControlFlags : 0x20 ' '
   +0x002 CycleProfiling   : 0y0
   +0x002 CounterProfiling : 0y0
   +0x002 GroupScheduling  : 0y0
   +0x002 AffinitySet      : 0y0
   +0x002 Tagged           : 0y0
   +0x002 EnergyProfiling  : 0y1
```

```
   +0x002 ThreadReservedControlFlags : 0y00
   +0x003 DebugActive      : 0 ''
   +0x003 ActiveDR7        : 0y0
   +0x003 Instrumented     : 0y0
   +0x003 Minimal          : 0y0
   +0x003 Reserved4        : 0y000
   +0x003 UmsScheduled     : 0y0
   +0x003 UmsPrimary       : 0y0
   +0x000 MutantType       : 0x6 ''
   +0x001 MutantSize       : 0 ''
   +0x002 DpcActive        : 0x20 ' '
   +0x003 MutantReserved   : 0 ''
   +0x004 SignalState      : 0n0
   +0x008 WaitListHead     : _LIST_ENTRY [ 0xfffff800`f043e948 - 0xfffff800`f043e948 ]
```

- *Représentation absolue ou relative (Absolute)* Signifie qu'un délai d'expiration est absolu (par opposition à un autre exprimé de façon relative).
- *État abandonné (Abandoned)* Signifie que le thread détenteur d'un mutex a pris fin sans avoir eu l'occasion de le libérer. S'applique uniquement aux mutex.
- *Points d'arrêt actifs (ActiveDR7)* Indique que des points d'arrêt matériels sont programmés de manière à contrôler l'exécution du thread.
- *Lock* Voir routine KiAcquireKobjectLock.
- *Minimal* Indique un thread minimal. Les threads créés à l'intérieur d'un processus minimal (ou pico processus) sont appelés threads minimaux.
- *Taille (Size)* Taille de l'objet auquel se rapporte l'entête dispatcher divisée par 4 ; s'applique à l'ensemble des objets.
- *Type de l'objet (Type)* Donne le type le type auquel appartient l'objet, par exemple processus, mutex, événement à réinitialisation manuelle, événement à réinitialisation automatique, et bien d'autres possibilités. Voir structure de données KOBJECTS.

KOBJECTS.

```
lkd> dt nt!_KOBJECTS
   EventNotificationObject = 0n0
   EventSynchronizationObject = 0n1
   MutantObject = 0n2
   ProcessObject = 0n3
   QueueObject = 0n4
   SemaphoreObject = 0n5
   ThreadObject = 0n6
   GateObject = 0n7
   TimerNotificationObject = 0n8
   TimerSynchronizationObject = 0n9
   Spare2Object = 0n10
   Spare3Object = 0n11
   Spare4Object = 0n12
   Spare5Object = 0n13
   Spare6Object = 0n14
   Spare7Object = 0n15
   Spare8Object = 0n16
   ProfileCallbackObject = 0n17
   ApcObject = 0n18
   DpcObject = 0n19
   DeviceQueueObject = 0n20
   PriQueueObject = 0n21
   InterruptObject = 0n22
   ProfileObject = 0n23
```

```
Timer2NotificationObject = 0n24
Timer2SynchronizationObject = 0n25
ThreadedDpcObject = 0n26
MaximumKernelObject = 0n27
```

Tableau 7.28. Définitions de l'état signalé

Type d'objet	Configuré à l'état signalé quand	Effet sur les threads en attente
Événement (type notification)	thread initialise l'événement	Tous libérés
Événement (type synchronisation)	thread initialise l'événement	Un thread libéré ; objet événement réinitialisé
Événement à clé	thread initialise l'événement	Libération du thread qui attend la clé
Fichier	opération d'E/S se termine	Tous libérés
File d'attente	élément est placé dans la file	Un thread libéré
Minuterie (type notification)	heure d'initialisation arrive ou délai expire	Tous libérés
Minuterie (type synchronisation)	heure d'initialisation arrive ou délai expire	Un thread libéré
Mutex	thread libère le mutex	Un thread libéré
Processus	dernier thread se termine	Tous libérés
Sémaphore	compteur du sémaphore tombe à 1	Un thread libéré
Thread	thread se termine	Tous libérés

Rundown

Un système multi programmé tel Windows gère l'exclusion mutuelle essentiellement à l'aide de primitives de verrouillage, et assure par ce biais des accès synchronisés aux structures communes entre différents processus. Quand la stratégie basée sur des verrous n'est pas optimale, le système s'appuie alors sur une solution plus fine, appelée protection rundown, qui offre un moyen commode de se prémunir contre la suppression prématurée de ressources globales partagées. (Scénario qui, s'il n'était pris en compte, se traduirait inévitablement par, au mieux une corruption des données, au pire un comportement imprévisible - voire un effondrement - de l'ordinateur.)

Tableau 7.29. Routines se rapportant à la protection rundown

BOOLEAN	ExAcquireRundownProtection (PEX_RUNDOWN_REF RunRef)
VOID	ExInitializeRundownProtection (PEX_RUNDOWN_REF RunRef)
VOID	ExReleaseRundownProtection (PEX_RUNDOWN_REF RunRef)

Horloges et minuteries (timers)

Les ordinateurs modernes intègrent pour la plupart des horloges et des minuteries dont résultent un certain nombre de fonctionnalités, notamment : (1) donner l'heure actuelle ; (2) prendre la mesure du temps écoulé ; (3) déclencher une opération en fonction d'une heure définie.

Les minuteries (timers) sont des objets utilisés pour enregistrer le passage du temps, le plus souvent en vue de reporter l'exécution d'une fonction donnée à un moment ultérieur.

Le tableau suivant répertorie quelles fonctions parmi les interfaces de programmation de Windows ont un lien avec ses objets minuterie.

Tableau 7.30. Opérations sur les minuteries

Service	Description
NtCreateTimer	Crée, ou éventuellement ouvre, un objet minuterie.
NtOpenTimer	Ouvre un objet minuterie existant.
NtCancelTimer	Suspend le délai associé à une minuterie
NtQueryTimer	Retourne des informations relatives à une minuterie.

NtSetTimer	Active une minuterie

Tableau 7.31. Routines concernant les timers

Routine	Description
KeReadStateTimer	Retourne l'état de signal associé à une minuterie donnée.
KeSetTimer	Modifie les paramètres d'une minuterie donnée.

L'arrêt d'une minuterie donnée avant échéance se fait par un appel à CancelWaitableTimer, laquelle à pour effet la suspension du compte à rebours et l'annulation des APC en suspens associés. La fonction n'a par contre aucune incidence en ce qui concerne l'état de signal de l'objet. De ce fait, les threads bloqués dans l'attente d'une minuterie le restent jusqu'à l'issue de leur délai d'attente interne ou, autre configuration, que la minuterie soit ranimée (fonction SetWaitableTimer) et alors éventuellement signalée.

Le tableau suivant présente quels attributs sont impliquées lors du contrôle d'accès à des objets minuterie.

Tableau 7.32. Droits d'accès concernant les timers

Constante	Valeur	Description
TIMER_QUERY_STATE	0x0001	Requis pour demander des informations relatives à une minuterie. Controlé par NtQueryTimer.
TIMER_MODIFY_STATE	0x0002	Requis pour modifier les paramètres qui régissent le comportement d'une minuterie. Controlé par NtCancelTimer.

La liste qui suit décrit quelques-uns des attributs de la structure ETIMER :

```
lkd> dt nt!_ETIMER
   +0x000 KeTimer          : _KTIMER
   +0x040 Lock             : Uint8B
   +0x048 TimerApc         : _KAPC
   +0x0a0 TimerDpc         : _KDPC
   +0x0e0 ActiveTimerListEntry : _LIST_ENTRY
   +0x0f0 Period           : Uint4B
   +0x0f4 TimerFlags       : Char
   +0x0f4 ApcAssociated    : Pos 0, 1 Bit
   +0x0f4 FlushDpcs        : Pos 1, 1 Bit
   +0x0f4 Paused           : Pos 2, 1 Bit
   +0x0f4 Spare1           : Pos 3, 5 Bits
   +0x0f5 DueTimeType      : UChar
   +0x0f6 Spare2           : Uint2B
   +0x0f8 WakeReason       : Ptr64 _DIAGNOSTIC_CONTEXT
   +0x100 WakeTimerListEntry : _LIST_ENTRY
   +0x110 VirtualizedTimerCookie : Ptr64 Void
   +0x118 VirtualizedTimerLinks : _LIST_ENTRY
   +0x128 DueTime          : Uint8B
   +0x130 CoalescingWindow : Uint4B

lkd> dt nt!_KTIMER
   +0x000 Header           : _DISPATCHER_HEADER
   +0x018 DueTime          : _ULARGE_INTEGER
   +0x020 TimerListEntry   : _LIST_ENTRY
   +0x030 Dpc              : Ptr64 _KDPC
   +0x038 Processor        : Uint4B
   +0x03c Period           : Uint4B
```

- *APC associé (ApcAssociated)* Variable booléenne dont l'état valide, ou au contraire, infirme le fait qu'un APC est associé à la minuterie. Voir service NtSetTimer (argument TimerApcRoutine).

- *Compte à rebours (DueTime)* Le compte à rebours avant que le noyau ne fasse passer un objet minuterie à l'état signalé. Cette valeur peut revêtir 2 formes, et signifier soit une heure spécifique soit un délai. Quand un thread enregistre les conditions d'activation d'un timer (KeSetTimer), il mentionne une période de temps

fixe, exprimée en unités de 100 nanosecondes, dont c'est la valeur qui, positive ou négative, conditionne l'interprétation : une valeur positive indique un temps absolu (par exemple, le 19 juillet 2018 à 17:35), une valeur négative indique un intervalle par rapport à l'actuelle horloge du système (par exemple, 5 minutes à partir de maintenant). (Si l'heure système subit un réglage, celle associée aux timers déjà en place est ajustée en conséquence.) Voir service NtSetTimer (paramètre DueTime).

- *Bloc Timer du noyau (KeTimer)* Regroupe les différents attributs dont le noyau a besoin pour gérer l'objet.
- *Verrou (Lock)* Verrou tournant employé lors de modifications ou de manipulations internes de la structure.
- *Période (Period)* La période de la minuterie, en millisecondes. Si cette propriété a la valeur zéro, l'objet se comporte comme une minuterie à déclenchement unique. Autrement, la minuterie est périodique. Voir service NtSetTimer (paramètre Period) ; routine système KeSetTimerEx (paramètre Period).

Tableau 7.33. Opérations sur les minuteries

Opération	Fonction	Service	Routine
Créer un objet minuterie	CreateWaitableTimer(Ex)	NtCreateTimer	ObCreateObject
Ouvrir un objet minuterie	OpenWaitableTimer	NtOpenTimer	ObOpenObjectByName
Suspendre le délai associé à une minuterie	CancelWaitableTimer	NtCancelTimer	KeCancelTimer
Obtenir des informations relatives à une minuterie.		NtQueryTimer	KeReadStateTimer
Modifier les paramètres d'une minuterie donnée	SetWaitableTimer	NtSetTimer	KeSetTimer(Ex)
Obtenir l'état de signal associé à une minuterie donnée		NtQueryTimer	KeReadStateTimer

Tableau 7.34. TIMER_INFORMATION_CLASS

ID	Classe	Type de données associé
0x00	TimerBasicInformation	TIMER_BASIC_INFORMATION

TIMER_BASIC_INFORMATION.

```
typedef struct _TIMER_BASIC_INFORMATION {
    LARGE_INTEGER RemainingTime;
    BOOLEAN TimerState;
} TIMER_BASIC_INFORMATION, *PTIMER_BASIC_INFORMATION;
```

Tableau 7.35. Valeurs initiales des champs de la structure SYSTEM_BASIC_INFORMATION

Champ	Valeur initiale
RemainingTime	ETIMER DueTime
TimerState	ETIMER KeTimer.Header.SignalState

Tableau 7.36. Eléments auxiliaires sous-jacents au caractère périodique ou non d'une minuterie

Composant	Élément	Attribut
Kernel32	SetWaitableTimer	lPeriod
Ntdll.dll	NtSetTimer	Period
Ntoskrnl.exe	KeSetTimerEx	Period
	ETIMER	Period

Tableau 7.37. Eléments auxiliaires sous-jacents à l'état de signal d'une minuterie

Composant	Élément	Attribut
Ntdll.dll	SYSTEM_BASIC_INFORMATION	TimerState
	NtSetTimer	PreviousState
Ntoskrnl.exe	ETIMER	KeTimer.Header.SignalState

Tableau 7.38. Droits d'accès spécifiques aux sémaphores

Constante	Description	Description
SEMAPHORE_QUERY_STATE	0x0001	Requis pour demander des informations relatives à un sémaphore. Controlé par NtQuerySemaphore.
SEMAPHORE_MODIFY_STATE	0x0002	Requis pour modifier les paramètres qui régissent le comportement d'un sémaphore. Controlé par NtReleaseSemaphore.

Opérations inter verrouillées

Un moyen simple de synchroniser les accès à une ressource partagée entre plusieurs threads et éviter les situations de compétition est de s'assurer qu'une opération est atomique, c'est-à-dire considérée comme indivisible, exécutée en une seule instruction. De rudimentaires opérations atomiques sont fournies par les architectures modernes de processeur, et sont souvent la base d'autres mécanismes gérés par le système d'exploitation. Qu'une opération soit ou non atomique (au sens qu'elle le soit au niveau du matériel) est fonction des critères suivants :

- Les instructions qui font zéro ou un accès mémoire sont exécutées de façon atomique.
- Les opérations pour lesquelles la micro architecture est de type lire / modifier / écrire, tels que les instructions inc ou dec qui lisent une donnée à partir de la mémoire, la mettent à jour, et transvase la valeur modifiée dans la mémoire, sont atomiques si aucun autre processeur n'interfère avec le bus mémoire pendant l'opération. (Naturellement, cette situation ne se produit jamais dans un système uni processeur, vu que tous les accès mémoire sont fabriqués par le même processeur.)
- Les instructions dont la sémantique est associée au préfixe d'instruction lock (comme dans lock xadd) sont atomiques même dans un système multi processeur. Lorsqu'une unité de traitement détecte ce préfixe, elle verrouille le bus multi processeur jusqu'à ce que l'instruction soit terminée, de façon qu'aucun autre processeur ne puisse accéder à l'emplacement de mémoire tandis que l'opération est en cours.
- Les instructions dont la sémantique est associée au préfixe d'instruction rep (comme dans rep movsb), obligeant l'unité de traitement à répéter plusieurs fois la même instruction, ne sont pas atomiques.

Dans Windows, on retrouve un certain nombre d'interfaces directement conscientes de multiplicité des processeurs et d'exclusion mutuelle, des fonctions comme InterlockedIncrement, InterlockedDecrement, ou InterlockedExchange, qui emploient le préfixe d'instruction lock pour être réalisée sous forme d'opération atomique. Ce genre de synchronisation élémentaire est utilisé tant dans les sous-systèmes d'environnement que le noyau et les pilotes.

Tableau 7.39. Fonctions d'arithmétique

Fonction	Description
InterlockedCompareExchange	Compare deux valeurs et, en cas d'égalité, en affecte une troisième à l'une d'entre elles
InterlockedDecrement	Soustrait 1 d'une valeur
InterlockedExchange	Affecte une valeur donnée à une variable

Région critique et région gardée

Région critique et région gardée sont deux primitives de synchronisation du noyau conçus pour empêcher l'exécution de certaines routines asynchrones (APC), évitant de la sorte qu'un thread soit interrompu tandis qu'il manipule une ressource partagée. De tels mécanismes sont souvent utilisées en conjonction avec des verrous (par exemple ressources exécutif, ou mutex rapides/gardés), mais n'assurent pas à eux seuls l'exclusion mutuelle.

Une région critique empêche temporairement l'exécution des APC noyau normaux pour le thread actuel. Elle n'interdit cependant ni les interruptions système vitales (APC spéciaux), ni les APC en mode utilisateur - lesquels ne peuvent de toute manière pas s'exécuter tant que le thread reste en mode noyau. Un thread entre dans une région critique au moyen de la fonction `KeEnterCriticalRegion`, et en sort explicitement avec `KeLeaveCriticalRegion`. Quelques-unes des routines du noyau exigent que le thread se trouve déjà dans une section critique quand elles sont appelées, par exemple celles impliquant des manipulations de ressources exécutif.

Plus stricte qu'une région critique, une région gardée empêche temporairement l'exécution de tous les types d'APC : utilisateur, noyau normaux et noyau spéciaux, agissant ainsi comme une barrière totalement hermétique aux interruptions. L'entrée dans cette région s'effectue via `KeEnterGuardedRegion`, et la sortie via `KeLeaveGuardedRegion`.

Chapitre 8. Processus et threads

Dans ce chapitre, nous allons voir les abstractions fondamentales dont se servent les systèmes d'exploitation pour exécuter du code, les *threads*, et fournir aux programmes de quoi s'exécuter, les *processus*. Nous décrirons les structures de données, les mécanismes internes et les procédés algorithmiques en œuvre dans la gestion de ces entités dans Microsoft Windows, et là où existent des variables noyau ou des compteurs de performance, nous les mentionnerons. La première section se concentre sur les coulisses des processus, la seconde sur celles des threads et de leur ordonnancement. Le chapitre se terminera par une description de l'objet job.

Processus et threads sont en lien avec nombre d'autres sujets, et font contact avec pléthore de composants dans l'architecture Windows. Conséquemment, une large partie des notions vues dans ce chapitre sont expliquées en détail ailleurs dans l'ouvrage. Pour tirer les meilleurs enseignements de ce chapitre, vous devrez connaître les notions théoriques encadrant l'exécution des logiciels (vues au chapitre 1) ; par exemple la différence entre processus et programme, entre processus et thread, entre mode utilisateur et mode noyau. Vous devez également être au fait de l'utilisation dans Windows d'objets. Ceux sur lesquels s'appuie la gestion des processus et des threads sont mis en lumière à l'intérieur de ce chapitre.

Dans les coulisses des processus

Dans Windows, un processus est un conteneur pour un ensemble de ressources employées lors de l'exécution d'une instance de programme. Chaque processus réunit donc l'orchestration de toutes les activités motrices nécessaires au fonctionnement d'un logiciel. Cela inclut, sur une perspective (très) globale un espace d'adressage virtuel, un programme exécutable, des capacités à manipuler diverses ressources système, et au moins un thread d'exécution.

Les sections qui suivent couvrent les mécanismes et procédures internes implémentés en tant que parties prenantes de la gestion des processus.

Structure de données

Cette section décrit les structures de données Windows fondamentales sur lesquelles s'appuie la gestion des processus. Notez dès à présent que, compte tenu de la nature du système, qui comme nous l'avons rapporté ailleurs fait référence à un empilement de couches successives, c'est à la fois le noyau (Ntoskrnl.exe), le gestionnaire de processus (Ntoskrnl.exe encore), la portion mode noyau du sous-système Windows (Win32k.sys) et le sous-système environnemental (Csrss), qui tous sont impliqués à l'agencement des données en ce qui concerne les processus (et, partant, les threads). En termes concret, cela se traduit par l'existence des canevas suivants :

- EPROCESS (*Executive Process*)
- KPROCESS (*Kernel Process*)
- W32PROCESS (*Win32 Process*)
- PEB (*Process Environment Block*)

Objet processus de l'exécutif (EPROCESS)

Chaque processus sous Windows est représenté au niveau de la sphère supérieure du système par un bloc EPROCESS (*Executive Process*), lequel héberge toutes les informations de contrôle requis pour la maintenance d'un espace d'adressage virtuel et l'exécution d'un ensemble d'objets thread. Cela inclut, entres autres, l'ID client du processus, les informations de mémoire et d'ensemble de travail (working set), la description du profil de sécurité.

Outre les nombreux attributs relatifs à un processeur, un bloc EPROCESS sert de passerelle (comprendre qu'il contient ou pointe) vers plusieurs autres structures de données associés. Ainsi, chaque processus se voit confier ses capacités de synchronisation par le biais d'un entête d'objet dispatcher commun, qui fait partie du bloc EPROCESS, et est le réceptacle d'un ou plusieurs threads représentés par des blocs ETHREAD (Executive Thread), pointés par le bloc EPROCESS.

Etant donné le lien entre la structure EPROCESS et la gestion opérée au niveau de l'environnement local, il existe autant d'occurence de ladite structure que de processus s'exécutant la station de travail, y compris ceux ne l'étant pas à part entière, à savoir les entités baptisées Inactif et System.

Visualisation du format de la structure EPROCESS

Pour voir une liste détaillée des éléments qui constituent tout objet processus de l'exécutif, saisissez dans la fenêtre du débogueur noyau la commande dt nt!_EPROCESS. Le résultat (tronqué de sorte à économiser de l'espace) ressemble à ce qui suit :

```
lkd> *dt nt!_EPROCESS*
   +0x000 Pcb              : _KPROCESS
   +0x2c8 ProcessLock      : _EX_PUSH_LOCK
   +0x2d0 CreateTime       : _LARGE_INTEGER
   +0x2d8 RundownProtect   : _EX_RUNDOWN_REF
   +0x2e0 UniqueProcessId  : Ptr64 Void
   +0x2e8 ActiveProcessLinks : _LIST_ENTRY
   .
   .
   .
```

La commande dt montre le format du bloc EPROCESS, pas son contenu. Pour afficher une instance d'un processus concret, vous pouvez spécifier l'adresse d'une structure EPROCESS comme argument de dt. Pour connaitre les adresses de tous les blocs EPROCESS du système, employez la commande !process 0 0. Pour connaitre l'adresse du bloc EPROCESS d'un processus en particulier, employez la commande !process 0 0 suivie du nom de l'image exécutée dans ce processus. Voici ce que donne la commande dt quand elle est conjointe à une adresse. Notez qu'au contraire de l'exemple qui précède, nous laissons cette fois se développer entièrement l'affichage résultant de l'opération. Vous pourrez de cette façon y revenir au besoin lorsque nous ferons référence à la nature programmatique de tel ou tel champ de la structure processus.

```
lkd> dtx nt!_EPROCESS ffffda8e1bb5a080
(*((nt!_EPROCESS *)0xffffda8e1bb5a080))                [Type: _EPROCESS]
    [+0x000] Pcb              [Type: _KPROCESS]
    [+0x2e0] ProcessLock      [Type: _EX_PUSH_LOCK]
    [+0x2e8] UniqueProcessId  : 0x193c [Type: void *]
    [+0x2f0] ActiveProcessLinks [Type: _LIST_ENTRY]
    [+0x300] RundownProtect   [Type: _EX_RUNDOWN_REF]
    [+0x308] Flags2           : 0x200d000 [Type: unsigned long]
    [+0x308 ( 0: 0)] JobNotReallyActive : 0x0 [Type: unsigned long]
    [+0x308 ( 1: 1)] AccountingFolded : 0x0 [Type: unsigned long]
    [+0x308 ( 2: 2)] NewProcessReported : 0x0 [Type: unsigned long]
    [+0x308 ( 3: 3)] ExitProcessReported : 0x0 [Type: unsigned long]
    [+0x308 ( 4: 4)] ReportCommitChanges : 0x0 [Type: unsigned long]
    [+0x308 ( 5: 5)] LastReportMemory : 0x0 [Type: unsigned long]
    [+0x308 ( 6: 6)] ForceWakeCharge  : 0x0 [Type: unsigned long]
    [+0x308 ( 7: 7)] CrossSessionCreate : 0x0 [Type: unsigned long]
    [+0x308 ( 8: 8)] NeedsHandleRundown : 0x0 [Type: unsigned long]
    [+0x308 ( 9: 9)] RefTraceEnabled  : 0x0 [Type: unsigned long]
    [+0x308 (10:10)] PicoCreated      : 0x0 [Type: unsigned long]
    [+0x308 (11:11)] EmptyJobEvaluated : 0x0 [Type: unsigned long]
    [+0x308 (14:12)] DefaultPagePriority : 0x5 [Type: unsigned long]
    [+0x308 (15:15)] PrimaryTokenFrozen : 0x1 [Type: unsigned long]
    [+0x308 (16:16)] ProcessVerifierTarget : 0x0 [Type: unsigned long]
    [+0x308 (17:17)] RestrictSetThreadContext : 0x0 [Type: unsigned long]
    [+0x308 (18:18)] AffinityPermanent : 0x0 [Type: unsigned long]
    [+0x308 (19:19)] AffinityUpdateEnable : 0x0 [Type: unsigned long]
    [+0x308 (20:20)] PropagateNode    : 0x0 [Type: unsigned long]
    [+0x308 (21:21)] ExplicitAffinity : 0x0 [Type: unsigned long]
    [+0x308 (23:22)] ProcessExecutionState : 0x0 [Type: unsigned long]
    [+0x308 (24:24)] EnableReadVmLogging : 0x0 [Type: unsigned long]
    [+0x308 (25:25)] EnableWriteVmLogging : 0x1 [Type: unsigned long]
    [+0x308 (26:26)] FatalAccessTerminationRequested : 0x0 [Type: unsigned long]
    [+0x308 (27:27)] DisableSystemAllowedCpuSet : 0x0 [Type: unsigned long]
```

```
[+0x308 (29:28)] ProcessStateChangeRequest : 0x0 [Type: unsigned long]
[+0x308 (30:30)] ProcessStateChangeInProgress : 0x0 [Type: unsigned long]
[+0x308 (31:31)] InPrivate        : 0x0 [Type: unsigned long]
[+0x30c] Flags            : 0x144d0c01 [Type: unsigned long]
[+0x30c ( 0: 0)] CreateReported   : 0x1 [Type: unsigned long]
[+0x30c ( 1: 1)] NoDebugInherit   : 0x0 [Type: unsigned long]
[+0x30c ( 2: 2)] ProcessExiting   : 0x0 [Type: unsigned long]
[+0x30c ( 3: 3)] ProcessDelete    : 0x0 [Type: unsigned long]
[+0x30c ( 4: 4)] ManageExecutableMemoryWrites : 0x0 [Type: unsigned long]
[+0x30c ( 5: 5)] VmDeleted        : 0x0 [Type: unsigned long]
[+0x30c ( 6: 6)] OutswapEnabled   : 0x0 [Type: unsigned long]
[+0x30c ( 7: 7)] Outswapped       : 0x0 [Type: unsigned long]
[+0x30c ( 8: 8)] FailFastOnCommitFail : 0x0 [Type: unsigned long]
[+0x30c ( 9: 9)] Wow64VaSpace4Gb  : 0x0 [Type: unsigned long]
[+0x30c (11:10)] AddressSpaceInitialized : 0x3 [Type: unsigned long]
[+0x30c (12:12)] SetTimerResolution : 0x0 [Type: unsigned long]
[+0x30c (13:13)] BreakOnTermination : 0x0 [Type: unsigned long]
[+0x30c (14:14)] DeprioritizeViews : 0x0 [Type: unsigned long]
[+0x30c (15:15)] WriteWatch       : 0x0 [Type: unsigned long]
[+0x30c (16:16)] ProcessInSession : 0x1 [Type: unsigned long]
[+0x30c (17:17)] OverrideAddressSpace : 0x0 [Type: unsigned long]
[+0x30c (18:18)] HasAddressSpace  : 0x1 [Type: unsigned long]
[+0x30c (19:19)] LaunchPrefetched : 0x1 [Type: unsigned long]
[+0x30c (20:20)] Background       : 0x0 [Type: unsigned long]
[+0x30c (21:21)] VmTopDown        : 0x0 [Type: unsigned long]
[+0x30c (22:22)] ImageNotifyDone  : 0x1 [Type: unsigned long]
[+0x30c (23:23)] PdeUpdateNeeded  : 0x0 [Type: unsigned long]
[+0x30c (24:24)] VdmAllowed       : 0x0 [Type: unsigned long]
[+0x30c (25:25)] ProcessRundown   : 0x0 [Type: unsigned long]
[+0x30c (26:26)] ProcessInserted  : 0x1 [Type: unsigned long]
[+0x30c (29:27)] DefaultIoPriority : 0x2 [Type: unsigned long]
[+0x30c (30:30)] ProcessSelfDelete : 0x0 [Type: unsigned long]
[+0x30c (31:31)] SetTimerResolutionLink : 0x0 [Type: unsigned long]
[+0x310] CreateTime       : {132343910626410887} [Type: _LARGE_INTEGER]
[+0x318] ProcessQuotaUsage [Type: unsigned __int64 [2]]
[+0x328] ProcessQuotaPeak [Type: unsigned __int64 [2]]
[+0x338] PeakVirtualSize  : 0x2010a5de000 [Type: unsigned __int64]
[+0x340] VirtualSize      : 0x2010a35e000 [Type: unsigned __int64]
[+0x348] SessionProcessLinks [Type: _LIST_ENTRY]
[+0x358] ExceptionPortData : 0xffffda8e1a35edf0 [Type: void *]
[+0x358] ExceptionPortValue : 0xffffda8e1a35edf0 [Type: unsigned __int64]
[+0x358 ( 2: 0)] ExceptionPortState : 0x0 [Type: unsigned __int64]
[+0x360] Token            [Type: _EX_FAST_REF]
[+0x368] MmReserved       : 0x0 [Type: unsigned __int64]
[+0x370] AddressCreationLock [Type: _EX_PUSH_LOCK]
[+0x378] PageTableCommitmentLock [Type: _EX_PUSH_LOCK]
[+0x380] RotateInProgress : 0x0 [Type: _ETHREAD *]
[+0x388] ForkInProgress   : 0x0 [Type: _ETHREAD *]
[+0x390] CommitChargeJob  : 0x0 [Type: _EJOB *]
[+0x398] CloneRoot        [Type: _RTL_AVL_TREE]
[+0x3a0] NumberOfPrivatePages : 0x204 [Type: unsigned __int64]
[+0x3a8] NumberOfLockedPages : 0x0 [Type: unsigned __int64]
[+0x3b0] Win32Process     : 0xffffc390c3049660 [Type: void *]
[+0x3b8] Job              : 0x0 [Type: _EJOB *]
[+0x3c0] SectionObject    : 0xffffc4841f25d970 [Type: void *]
[+0x3c8] SectionBaseAddress : 0x7ff7bf050000 [Type: void *]
[+0x3d0] Cookie           : 0x68bd1366 [Type: unsigned long]
[+0x3d8] WorkingSetWatch  : 0x0 [Type: _PAGEFAULT_HISTORY *]
[+0x3e0] Win32WindowStation : 0x98 [Type: void *]
[+0x3e8] InheritedFromUniqueProcessId : 0xdb4 [Type: void *]
[+0x3f0] OwnerProcessId   : 0xdb6 [Type: unsigned __int64]
[+0x3f8] Peb              : 0xde76abe000 [Type: _PEB *]
[+0x400] Session          : 0xffff8701b1853000 [Type: _MM_SESSION_SPACE *]
[+0x408] Spare1           : 0x0 [Type: void *]
[+0x410] QuotaBlock       : 0xffffda8e1bde5d40 [Type: _EPROCESS_QUOTA_BLOCK *]
```

```
[+0x418] ObjectTable      : 0xffffc4841724d480 [Type: _HANDLE_TABLE *]
[+0x420] DebugPort        : 0x0 [Type: void *]
[+0x428] WoW64Process     : 0x0 [Type: _EWOW64PROCESS *]
[+0x430] DeviceMap        : 0xffffc48415379b70 [Type: void *]
[+0x438] EtwDataSource    : 0xffffda8e1d2035d0 [Type: void *]
[+0x440] PageDirectoryPte : 0x0 [Type: unsigned __int64]
[+0x448] ImageFilePointer : 0xffffda8e1f402e60 [Type: _FILE_OBJECT *]
[+0x450] ImageFileName    [Type: unsigned char [15]]
[+0x45f] PriorityClass    : 0x2 [Type: unsigned char]
[+0x460] SecurityPort     : 0x0 [Type: void *]
[+0x468] SeAuditProcessCreationInfo [Type: _SE_AUDIT_PROCESS_CREATION_INFO]
[+0x470] JobLinks         [Type: _LIST_ENTRY]
[+0x480] HighestUserAddress : 0x7fffffff0000 [Type: void *]
[+0x488] ThreadListHead   [Type: _LIST_ENTRY]
[+0x498] ActiveThreads    : 0x1 [Type: unsigned long]
[+0x49c] ImagePathHash    : 0xb28cc291 [Type: unsigned long]
[+0x4a0] DefaultHardErrorProcessing : 0x1 [Type: unsigned long]
[+0x4a4] LastThreadExitStatus : 0 [Type: long]
[+0x4a8] PrefetchTrace    [Type: _EX_FAST_REF]
[+0x4b0] LockedPagesList  : 0x0 [Type: void *]
[+0x4b8] ReadOperationCount : {0} [Type: _LARGE_INTEGER]
[+0x4c0] WriteOperationCount : {0} [Type: _LARGE_INTEGER]
[+0x4c8] OtherOperationCount : {4} [Type: _LARGE_INTEGER]
[+0x4d0] ReadTransferCount : {0} [Type: _LARGE_INTEGER]
[+0x4d8] WriteTransferCount : {0} [Type: _LARGE_INTEGER]
[+0x4e0] OtherTransferCount : {0} [Type: _LARGE_INTEGER]
[+0x4e8] CommitChargeLimit : 0x0 [Type: unsigned __int64]
[+0x4f0] CommitCharge     : 0x28e [Type: unsigned __int64]
[+0x4f8] CommitChargePeak : 0x2fd [Type: unsigned __int64]
[+0x500] Vm               [Type: _MMSUPPORT_FULL]
[+0x640] MmProcessLinks   [Type: _LIST_ENTRY]
[+0x650] ModifiedPageCount : 0x1 [Type: unsigned long]
[+0x654] ExitStatus       : 259 [Type: long]
[+0x658] VadRoot          [Type: _RTL_AVL_TREE]
[+0x660] VadHint          : 0xffffda8e1cb951a0 [Type: void *]
[+0x668] VadCount         : 0x5a [Type: unsigned __int64]
[+0x670] VadPhysicalPages : 0x0 [Type: unsigned __int64]
[+0x678] VadPhysicalPagesLimit : 0x0 [Type: unsigned __int64]
[+0x680] AlpcContext      [Type: _ALPC_PROCESS_CONTEXT]
[+0x6a0] TimerResolutionLink [Type: _LIST_ENTRY]
[+0x6b0] TimerResolutionStackRecord : 0x0 [Type: _PO_DIAG_STACK_RECORD *]
[+0x6b8] RequestedTimerResolution : 0x0 [Type: unsigned long]
[+0x6bc] SmallestTimerResolution : 0x0 [Type: unsigned long]
[+0x6c0] ExitTime         : {0} [Type: _LARGE_INTEGER]
[+0x6c8] InvertedFunctionTable : 0x0 [Type: _INVERTED_FUNCTION_TABLE *]
[+0x6d0] InvertedFunctionTableLock [Type: _EX_PUSH_LOCK]
[+0x6d8] ActiveThreadsHighWatermark : 0x6 [Type: unsigned long]
[+0x6dc] LargePrivateVadCount : 0x0 [Type: unsigned long]
[+0x6e0] ThreadListLock   [Type: _EX_PUSH_LOCK]
[+0x6e8] WnfContext       : 0xffffc4841f263b40 [Type: void *]
[+0x6f0] ServerSilo       : 0x0 [Type: _EJOB *]
[+0x6f8] SignatureLevel   : 0x0 [Type: unsigned char]
[+0x6f9] SectionSignatureLevel : 0x0 [Type: unsigned char]
[+0x6fa] Protection       [Type: _PS_PROTECTION]
[+0x6fb ( 2: 0)] HangCount        : 0x0 [Type: unsigned char]
[+0x6fb ( 5: 3)] GhostCount       : 0x0 [Type: unsigned char]
[+0x6fb ( 6: 6)] PrefilterException : 0x0 [Type: unsigned char]
[+0x6fc] Flags3           : 0x40c000 [Type: unsigned long]
[+0x6fc ( 0: 0)] Minimal          : 0x0 [Type: unsigned long]
[+0x6fc ( 1: 1)] ReplacingPageRoot : 0x0 [Type: unsigned long]
[+0x6fc ( 2: 2)] Crashed          : 0x0 [Type: unsigned long]
[+0x6fc ( 3: 3)] JobVadsAreTracked : 0x0 [Type: unsigned long]
[+0x6fc ( 4: 4)] VadTrackingDisabled : 0x0 [Type: unsigned long]
[+0x6fc ( 5: 5)] AuxiliaryProcess : 0x0 [Type: unsigned long]
[+0x6fc ( 6: 6)] SubsystemProcess : 0x0 [Type: unsigned long]
```

```
[+0x6fc ( 7: 7)] IndirectCpuSets  : 0x0 [Type: unsigned long]
[+0x6fc ( 8: 8)] RelinquishedCommit : 0x0 [Type: unsigned long]
[+0x6fc ( 9: 9)] HighGraphicsPriority : 0x0 [Type: unsigned long]
[+0x6fc (10:10)] CommitFailLogged : 0x0 [Type: unsigned long]
[+0x6fc (11:11)] ReserveFailLogged : 0x0 [Type: unsigned long]
[+0x6fc (12:12)] SystemProcess    : 0x0 [Type: unsigned long]
[+0x6fc (13:13)] HideImageBaseAddresses : 0x0 [Type: unsigned long]
[+0x6fc (14:14)] AddressPolicyFrozen : 0x1 [Type: unsigned long]
[+0x6fc (15:15)] ProcessFirstResume : 0x1 [Type: unsigned long]
[+0x6fc (16:16)] ForegroundExternal : 0x0 [Type: unsigned long]
[+0x6fc (17:17)] ForegroundSystem : 0x0 [Type: unsigned long]
[+0x6fc (18:18)] HighMemoryPriority : 0x0 [Type: unsigned long]
[+0x6fc (19:19)] EnableProcessSuspendResumeLogging : 0x0 [Type: unsigned long]
[+0x6fc (20:20)] EnableThreadSuspendResumeLogging : 0x0 [Type: unsigned long]
[+0x6fc (21:21)] SecurityDomainChanged : 0x0 [Type: unsigned long]
[+0x6fc (22:22)] SecurityFreezeComplete : 0x1 [Type: unsigned long]
[+0x6fc (23:23)] VmProcessorHost  : 0x0 [Type: unsigned long]
[+0x700] DeviceAsid       : 0 [Type: long]
[+0x708] SvmData          : 0x0 [Type: void *]
[+0x710] SvmProcessLock   [Type: _EX_PUSH_LOCK]
[+0x718] SvmLock          : 0x0 [Type: unsigned __int64]
[+0x720] SvmProcessDeviceListHead [Type: _LIST_ENTRY]
[+0x730] LastFreezeInterruptTime : 0x0 [Type: unsigned __int64]
[+0x738] DiskCounters     : 0xffffda8e1bb5a900 [Type: _PROCESS_DISK_COUNTERS *]
[+0x740] PicoContext      : 0x0 [Type: void *]
[+0x748] EnclaveTable     : 0x0 [Type: void *]
[+0x750] EnclaveNumber    : 0x0 [Type: unsigned __int64]
[+0x758] EnclaveLock      [Type: _EX_PUSH_LOCK]
[+0x760] HighPriorityFaultsAllowed : 0x0 [Type: unsigned long]
[+0x768] EnergyContext    : 0xffffda8e1bb5a928 [Type: _PO_PROCESS_ENERGY_CONTEXT *]
[+0x770] VmContext        : 0x0 [Type: void *]
[+0x778] SequenceNumber   : 0x84e [Type: unsigned __int64]
[+0x780] CreateInterruptTime : 0x301ece0a14 [Type: unsigned __int64]
[+0x788] CreateUnbiasedInterruptTime : 0x301ece0a14 [Type: unsigned __int64]
[+0x790] TotalUnbiasedFrozenTime : 0x0 [Type: unsigned __int64]
[+0x798] LastAppStateUpdateTime : 0x301ece0a14 [Type: unsigned __int64]
[+0x7a0 (60: 0)] LastAppStateUptime : 0x0 [Type: unsigned __int64]
[+0x7a0 (63:61)] LastAppState     : 0x0 [Type: unsigned __int64]
[+0x7a8] SharedCommitCharge : 0x1359 [Type: unsigned __int64]
[+0x7b0] SharedCommitLock [Type: _EX_PUSH_LOCK]
[+0x7b8] SharedCommitLinks [Type: _LIST_ENTRY]
[+0x7c8] AllowedCpuSets   : 0x0 [Type: unsigned __int64]
[+0x7d0] DefaultCpuSets   : 0x0 [Type: unsigned __int64]
[+0x7c8] AllowedCpuSetsIndirect : 0x0 [Type: unsigned __int64 *]
[+0x7d0] DefaultCpuSetsIndirect : 0x0 [Type: unsigned __int64 *]
[+0x7d8] DiskIoAttribution : 0x0 [Type: void *]
[+0x7e0] DxgProcess       : 0xffffc484234f61f0 [Type: void *]
[+0x7e8] Win32KFilterSet  : 0x0 [Type: unsigned long]
[+0x7f0] ProcessTimerDelay [Type: _PS_INTERLOCKED_TIMER_DELAY_VALUES]
[+0x7f8] KTimerSets       : 0x0 [Type: unsigned long]
[+0x7fc] KTimer2Sets      : 0x0 [Type: unsigned long]
[+0x800] ThreadTimerSets  : 0x5 [Type: unsigned long]
[+0x808] VirtualTimerListLock : 0x0 [Type: unsigned __int64]
[+0x810] VirtualTimerListHead [Type: _LIST_ENTRY]
[+0x820] WakeChannel      [Type: _WNF_STATE_NAME]
[+0x820] WakeInfo         [Type: _PS_PROCESS_WAKE_INFORMATION]
[+0x850] MitigationFlags  : 0x21 [Type: unsigned long]
[+0x850] MitigationFlagsValues [Type: <anonymous-tag>]
[+0x854] MitigationFlags2 : 0x0 [Type: unsigned long]
[+0x854] MitigationFlags2Values [Type: <anonymous-tag>]
[+0x858] PartitionObject  : 0xffffda8e17498ea0 [Type: void *]
[+0x860] SecurityDomain   : 0x10000002a [Type: unsigned __int64]
[+0x868] ParentSecurityDomain : 0x10000002a [Type: unsigned __int64]
[+0x870] CoverageSamplerContext : 0x0 [Type: void *]
[+0x878] MmHotPatchContext : 0x0 [Type: void *]
```

- **Lien de processus actif (ActiveProcessLinks)** Entrée parmi une liste doublement chainée qui permet de relier entre eux les différents processus du système. Cette liste permet un accès rapide et efficace aux processus en cours d'exécution. Voir routine PspCreateProcess.

- **Compteur de threads (ActiveThreads)** Nombre de threads s'exécutant dans le contexte (dans l'espace d'adressage) du processus. Voir service NtQueryInformationThread (ThreadInformationClass = ThreadAmILastThread) ; routines PspCreateThread et PspExitThread.

- **Espace d'adressage initialisé (AddressSpaceInitialize)** Indique si l'espace d'adressage virtuel d'un processus est allé au-delà de sa phase de configuration initiale, assurant que ce dernier est en mesure d'interagir avec sa mémoire allouée et d'exécuter son code en toute sécurité (sans risque de violations d'accès).

- **Actualisation dynamique de l'affinité processeur (AffinityUpdateEnable)** Indique si le sous-ensemble des coeurs ou processeurs sur lesquels un processus a le droit de s'exécuter peut être réduit ou étendu après que ce dernier ait été créé. Dès lors que cet attribut est actif, l'affinité reste immuable, figée telle que définie initialement, y compris en cas de tentatives de modification par le système ou d'autres outils. Voir fonctions QueryProcessAffinityMode et SetProcessAffinityUpdateMode.

- **Arrêt sur terminaison (BreakOnTermination)** Détermine si le système doit déclencher une exception d'arrêt (break) lorsqu'un processus est sur le point de se terminer, offrant ainsi à un logiciel tiers - par exemple un débogueur - la possibilité d'enregistrer l'événement et d'examiner les conditions ayant favorisé cette terminaison. Voir services NtQueryInformationProcess (ProcessInformationClass = ProcessBreakOnTermination), NtSetInformationProcess (ProcessInformationClass = ProcessBreakOnTermination), NtTerminateProcess et RtlSetProcessIsCritical.

- **Témoin (Cookie)** Valeur unique associée à un processus, utilisée pour encoder ou décoder des pointeurs de manière sécurisée. Voir fonctions DecodePointer et EncodePointer, services NtQueryInformationProcess (ProcessInformationClass = ProcessCookie), RtlDecodePointer et RtlEncodePointer.

- **Charge mémoire engagée (CommitCharge)** Quantité totale de mémoire virtuelle qu'un processus a réservée et engagée auprès du système, offrant au noyau la possibilité d'imposer certaines limites vis-à-vis de l'utilisation des ressources.

- **Horodatage de création, temps d'interruption (CreateInterruptTime)** Enregistre le moment précis où un processus a été créé, exprimé en intervalles de 100 nanosecondes écoulés depuis le dernier démarrage du système. Cette mesure garantit un suivi fiable de la durée de vie d'un processus, indépendamment des modifications de l'heure système ou des ajustements de l'horloge.

- **Horodatage de création, temps système (CreateTime)** Date et heure à laquelle les services Windows internes de création de processus ont terminé dé générer le processus. Voir fonction GetProcessTimes ; service NtQueryInformationProcess (ProcessInformationClass = ProcessTimes) ; routines PsGetProcessCreateTimeQuadPart, *KeQuerySystemTime* et *PspCreateProcess*.

- **Création intersession (CrossSessionCreate)** Indique que le processus ayant créé celui-ci réside dans une session utilisateur différente. Le cas le plus majoritaire concerne les processus UWP, hébergés par un processus SvcHost standard qui s'exécute dans la session zéro.

- **Port de débogage (DebugPort)** Canal de communication inter processus (port LPC) utilisé lors d'événements de débogage. Voir services NtCreateProcess (DebugPort) et NtQueryInformationProcess (ProcessInformationClass = ProcessDebugPort) ; routines PsGetProcessDebugPort et PsIsProcessBeingDebugged.

- **Processus graphique (DxgProcess)** État du processus tel que défini dans la perspective du sous-système DirectX (`dxgkrnl.sys`), y compris ses contextes GPU, ses ressources graphiques allouées (textures, surfaces,

tampons, files de commandes, etc.) et, plus largement, ses interactions avec le modèle de pilote d'affichage (WDDM, Windows Display Driver Model).

- **Mémoire engagée (CommitCharge)** Nombre de pages de mémoire virtuelle réservées pour le compte d'un processus et qui sont garanties d'avoir un espace de sauvegarde dans le système, soit en mémoire physique, soit dans le fichier de pagination.

- **Limite de mémoire engagée (CommitChargeLimit)** Nombre maximal de pages que le processus est autorisé à réserver ou engager auprès du système de gestion de la mémoire virtuelle.

- **Pic de mémoire engagée (CommitChargePeak)** Nombre le plus élevé de pages qu'un processus a engagé depuis son lancement.

- **Priorité de pages par défaut (DefaultPagePriority)** Valeur, généralement comprise entre 0 (priorité faible) et 7 (priorité élevée) sur laquelle s'appuie le système pour déterminer quelles pages mémoire associées à un processus doivent être conservées en mémoire physique, et lesquelles peuvent être remplacées. Des processus système critiques, pour réduire les risques que leurs pages ne soient évincées de la mémoire centrale, peuvent se voir attribuer une priorité plus élevée. À l'inverse, des processus en arrière-plan, peuvent avoir une valeur plus faible, rendant leurs pages plus susceptibles d'être paginées sur le disque. Pour plus d'informations, voir section Priorité de pages au chapitre Gestion de la mémoire.

- **Traitement par défaut des erreurs critiques (DefaultHardErrorProcessing)** Entérine la stratégie générale à adopter dans le cas d'un dysfonctionnement sévère , permettant de décider si l'erreur exige une interaction utilisateur (affichage, le plus souvent, d'une boite de dialogue), être consignée dans un journal, ou précipiter la fin du processus concerné, garantissant ainsi un traitement maitrisé des situations critiques. Voir fonctions GetErrorMode et SetErrorMode ; services NtQueryInformationProcess et NtSetInformationProcess (ProcessInformationClass = ProcessDefaultHardErrorMode).

- **Mappage de périphérique (DeviceMap)** Gère les correspondances entre des lettres de lecteur (par exemple, `C:`, `D:`) et les chemins d'accès qui en résultent dans la perspective du noyau (par exemple, `\Device\HarddiskVolumeX`. Ce mécanisme permet ainsi aux processus de résoudre les chemins de type DOS vers leurs équivalents internes, utilisés par le système pour accéder aux périphériques. Voir service NtQueryInformationProcess (ProcessInformationClass = ProcessDeviceMap) ; routine ObSetCurrentProcessDeviceMap.

- **Interdiction du repositionement aléatoire (DisallowStrippedImages)** Indique que l'image exécutable sous-jacente au processus ne peut être redéfinie de manière aléatoire au moment du chargement via ASLR (option /DYNAMICBASE de l'Éditeur de liens). Voir fonctions Get/SetProcessMitigationPolicy (MitigationPolicy = ProcessASLRPolicy).

- **Restriction d'appels système (DisallowWin32kSystemCalls)** Indique qu'un processus n'est pas autorisé à interagir avec le sous-système graphique (`Win32k.sys`), limitant ainsi sa potentielle surface d'attaque et renforçant son isolation. Voir fonctions Get/SetProcessMitigationPolicy (MitigationPolicy = ProcessSystemCallDisablePolicy).

- **Statistiques d'E/S disque (DiskCounters)** Compteurs relatifs aux activités disque générés par un processus, par exemple le nombre total d'octets lus ou écrits depuis/vers un support de stockage.

- **Verrou de niveau enclave (EnclaveLock)** Primitive d'exclusion mutuelle utilisée pour synchroniser les accès à la table des enclaves rattachées à un processus, garantissant que les opérations la concernant s'effectuent de manière sûre et atomique.

- **Nombre d'enclaves (EnclaveNumber)** Nombre d'enclaves mémoire sécurisées actuellement actives pour un processus.

- **Table des enclaves (EnclaveTable)** Table des enclaves mémoire sécurisées associées à un processus, employées pour l'exécution de code et le stockage de données hautement sensibles.

- **Contexte énergétique (EnergyContext)** Suivi de la consommation énergétique et de l'impact d'un processus sur les ressources système (CPU, GPU, mémoire, E/S), afin d'alimenter les compteurs de performances et les stratégies établies en la matière (réduction de fréquence, mise en veille, optimisation de l'autonomie).

- **File d'attente ETW (EtwDataSource)** Pointeur vers une structure interne qu'emploie le noyau pour associer un processus à l'Infrastructure de traces et de journalisation (ETW, Event Tracing for Windows), jouant le rôle de file d'attente via conservation des divers événements ETW destinés au processus. Voir service `NtTraceControl`.

- **État de sortie (ExitStatus)** Code d'erreur servant à indiquer la raison pour laquelle un processus a pris fin, qu'il s'agisse en la circonstance d'une issue normale (terminaison du processus) ou d'une interruption liée à un dysfonctionnement ou à une action externe. Voir fonction GetExitCodeProcess ; service NtQueryInformationProcess (ProcessInformationClass = ProcessBasicInformation) ; routine PsGetProcessExitStatus ; structure PROCESS_BASIC_INFORMATION, attribut ExitStatus.

- **Moment de fin (ExitTime)** Date et heure (valeur horodatée) auxquelles un processus a achevé son exécution. Voir fonction GetProcessTimes ; service NtQueryInformationProcess (ProcessInformationClass = ProcessTimes) ; routine PsGetProcessExitTime.

- **Échange vers disque autorisé (Flags.OutswapEnabled)** Détermine si un processus est éligible au déplacement de ses pages depuis la mémoire physique vers le fichier d'échange. Obsolète dans les versions modernes de Windows, un tel attribut n'en demeure pas moins présent pour compatibilité interne.

- **Échange vers disque effectif (Flags.Outswapped)** Indique qu'un processus s'est vu retiré de la mémoire physique, et ses pages remisées dans le fichier d'échange.

- *Flags2* Voir routine MiArbitraryCodeBlocked.

- *ForceRelocateImages* Voir fonctions Get/SetProcessMitigationPolicy (MitigationPolicy = ProcessASLRPolicy).

- **ASLR à entropie élevée (*HighEntropyASLREnabled*)** Détermine si oui ou non le processus prend en charge la distribution aléatoire de l'espace d'adressage (ASLR) à une entropie élevée. Voir fonctions Get/SetProcessMitigationPolicy (MitigationPolicy = ProcessASLRPolicy).

- **Image exécutée (ImageFileName)** Nom du fichier exécutable à l'origine d'un processus, stocké sous forme d'une chaîne de caractères ASCII de longueur fixe. Ce champ ne contient que le nom de fichier sans chemin, tel que `notepad.exe`, et peut être tronqué si l'exécutable a un nom trop long. Voir service NtQueryInformationProcess (classe d'information ProcessImageFileName), routine PsGetProcessImageFileName.

- **Objet fichier (ImageFilePointer)** Pointeur vers l'objet fichier représentant le module exécutable à partir duquel le processus a été créé. Ce champ permet au système de référencer le fichier original sur disque, avec toutes ses métadonnées (chemin d'accès complet, attributs, opérations d'E/S, etc.). système Voir routine PsReferenceProcessFilePointer.

- **Empreinte de chemin (ImagePathHash)** Valeur de hachage calculée à partir du chemin d'accès complet au module dont émane un processus, accélérant les opérations de recherche/comparaison qui exigent qu'une telle information soit impliquée.

- **Table des fonctions inversée (InvertedFunctionTable)** Voir PsInvertedFunctionTable et LdrpInvertedFunctionTable.

- **ID processus parent (InheritedFromUniqueProcessId)** Identifiant du processus ayant donné lieu à celui-ci. Cette valeur est enregistrée une seule fois à la création du processus, et n'est jamais actualisée par la suite. Ainsi, comme les identifiants de processus sont des ressources réutilisables, il est possible, passé un certain temps, que l'identifiant concerné fasse référence à un autre processus que le parent d'origine. Voir service NtQueryInformationProcess (ProcessInformationClass = ProcessBasicInformation) ; routine PsGetProcessInheritedFromUniqueProcessId.

- **Groupe de processus (Job)** Pointeur vers le groupe (objet job) auquel appartient éventuellement le processus. Voir fonction AssignProcessToJobObject ; service NtAssignProcessToJobObject ; routine système PsGetProcessJob.
- *KeepAliveCounter* Voir service NtQueryInformationProcess (ProcessInformationClass = ProcessKeepAliveCount) ; routine PspAdjustKeepAliveCountProcess.
- **État de sortie final (LastThreadExitStatus)** Code de fin du dernier thread à s'être exécuté dans le contexte du processus. Ce code reflète l'état de sortie du thread une fois que celui-ci a pris fin. Voir routine PspExitThread.
- **Chargement anticipé (*LaunchPrefetched*)** Indique si le chargement du processus a été anticipé via le prefetcher logique. (Quand le noyau exécute le premier thread mode utilisateur d'un processus, les pages susceptibles d'être utilisés lors de l'initialisation de l'application sont préchargées dans le cache du système.)
- **Stratégies de protection (MitigationFlags)** Contrôle l'activation/inhibition de différentes stratégies d'atténuation des risques, telles que la distribution aléatoire de l'espace d'adressage (ASLR), la protection contre l'exécution (DEP), la surveillance du flux de contrôle (CFG), et d'autres mécanismes de durcissement susceptibles de mettre à l'abri un processus contre les attaques.
- *Informations LDT (LdtInformation)* Voir structure LDTINFORMATION.
- **Liste des pages verrouillées (LockedPagesList)** Liste des pages que le processus a verrouillé en mémoire physique, empêchant le système de remiser celles-ci dans le fichier d'échange.
- **Compteur de pages modifiées (ModifiedPageCount)** Nombre de pages mémoire d'un processus qui ont fait l'objet de modification depuis leur dernière actualisation à partir du disque.
- *NoDebugInherit* Indique si le port de débogage auquel se rapporte le processus est ou partagé avec des processus enfants. Voir constante DEBUG_PROCESS.
- **Nombre de pages privées (NumberOfPrivatePages)** Nombre de pages mémoire allouées pour le compte d'un processus donné, et qui ne sont pas partagées avec d'autres processus.
- **Nombre de pages verrouillées (NumberOfLockedPages)** Nombre de pages que les threads du processus ont verrouillées. Voir routine MmProbeAndLockPages.
- **Table des handles (*ObjectTable*)** Adresse de la table des handles du processus, laquelle contient des informations sur les objets associés au processus, tels que les handles et les descripteurs de fichiers.
- *Compteur d'E/S autre (OtherOperationCount)* Nombre d'opérations d'E/S qui ne sont ni des lectures ni des écritures (fonctions de contrôle, par exemple) générées par le processus depuis son démarrage. Voir fonction GetProcessIoCounters.
- *OtherTransferCount* Voir fonction GetProcessIoCounters.
- **Domaine de sécurité parent (ParentSecurityDomain)** Domaine de sécurité du processus parent, mémorisé au moment de la création d'un processus, offrant la possibilité au système de comparer les appartenances et de gérer l'héritage des règles de sécurité entre processus de même filiation.
- **Partition mémoire (PartitionObject)** Partition mémoire à laquelle se rattache un processus, laquelle régit le contexte et les stratégies de gestion mémoire qui s'appliquent à ce dernier, notamment en termes de statistiques, quotas et limites de consommation. Pertinent en premier lieu dans les scénarios de virtualisation (Hyper-V) et de gestion optimisée des ressources.
- **Bloc PEB (Peb)** Pointeur vers le bloc d'environnement du processus (PEB, Process Environment Block), où résident diverses informations sur sa configuration d'exécution : variables spécifiques, modules chargés, paramètres de démarrage, etc. Voir routine PsGetProcessPeb.
- **Bloc KPROCESS (*Pcb*)** Encapsule les caractéristiques de base d'un processus auprès du noyau : informations de planification, liste des threads, suivi des ressources consommées, etc.

- **Processus Pico (PicoCreated)** Indique si le processus a été créé sous l'égide du sous-système Pico, employé notamment par des environnements d'exécution non-natifs comme WSL.
- **Gel du jeton d'accès principal (PrimaryTokenFrozen)** Indique si le jeton d'accès principal d'un processus a été figé, empêchant toute substitution ultérieure. Ce gel verrouille de manière irrémédiable l'identité et les privilèges associés au processus, afin de le protéger contre les changements non souhaités.
- **Classe de priorité (PriorityClass)** Valeur qu'emploie les services de planification pour déterminer comment les threads d'un processus sont exécutés par rapport aux autres processus du système. Les threads héritent de cette classe lors de leur création, ce qui influence leur priorité effective et, partant, le temps processeur qui leur est alloué. Voir routine PsGetProcessPriorityClass, PsSetProcessPriorityClass. Voir fonctions GetPriorityClass et SetPriorityClass ; services NtQueryInformationProcess et NtSetInformationProcess (ProcessInformationClass = ProcessPriorityClass) ; routines PsGetProcessPriorityClass et PsSetProcessPriorityClass ; variable noyau PspPriorityTable.
- **Marqueur de terminaison (ProcessExiting)** Indique quand l'un des threads du processus est entré dans un service système de terminaison de processus (PspProcessDelete, PspExitProcess, etc.). Voir routine PsGetProcessExitProcessCalled.
- **Verrou de niveau processus (ProcessLock)** Primitive de verrouillage (ici en l'occurrence un push lock) employée de telle sorte à protéger le bloc EPROCESS lors des opérations qui le nécessite, cela dans l'optique de gérer les accès concurrents, éviter tout risque d'inter-blocage, et ainsi de suite.
- **Niveau de protection (Protection)** Détermine dans quelle mesure un processus est protégé des opérations émanant d'autres processus : accès, suspension, injection, terminaison. Certains processus systèmes ou critiques (par exemple, `csrss.exe` ou `wininit.exe`) bénéficient d'une protection renforcée, les mettant ainsi à l'abri d'agissements potentiellement frauduleux. Voir service NtQueryInformationProcess (ProcessProtectionInformation) ; routine PsIsProtectedProcess. Pour plus d'informations, voir section Processus protégés du chapitre Processus et threads.
- **Bloc quota (QuotaBlock)** Pointeur vers une structure chargée de mesurer l'utilisation de certaines ressources par un processus : pool paginé, pool non paginé, entrées dans la table des handles noyau, etc., ce qui permet en définitive au syst de limiter la consommation qu'effectue un processus vis-à-vis de certaines ressources. Les allocations effectuées depuis les pools de mémoire système (paginé et non paginé), ou encore la table des handles noyau, y sont par exemple comptabilisés. Un tel suivi permet concrètement au système d'empêcher un dépassement de ressources (erreur `STATUS_QUOTA_EXCEEDED`) et d'appliquer les restrictions configurées au niveau d'un processus ou d'un groupe de processus (job). Voir routines PspQueryQuotaLimits et PsReturnProcessQuota ; structure EPROCESS_QUOTA_BLOCK.
- *Compteur d'E/S lecture (ReadOperationCount)* Nombre d'opérations d'E/S lecture générées par le processus depuis son démarrage. Voir fonction GetProcessIoCounters ; mesure Lectures d'E/S dans le Gestionnaire des taches.
- **Compteur d'octets en lecture (*ReadTransferCount*)** Nombre d'octets lus par le processus depuis le début de son exécution jusqu'à la période actuelle. Voir fonction GetProcessIoCounters.
- **Informations d'audit (SeAuditProcessCreationInfo)** Pointeur vers une structure sur laquelle s'appuie le sous-système de sécurité de sorte à conserver des informations sur un processus au moment de sa création, permettant ensuite la génération d'événements d'audit.
- **Objet section (SectionObject)** Pointeur vers la région de mémoire en lien avec le module sur disque dont est issu un processus. Quand plusieurs processus partagent une même image, par exemple plusieurs instances du Bloc-Notes (`notepad.exe`), tous auront un objet section qui fait référence à la même section sous-jacente. Quand un processus prend fin, l'objet section épaule le système quant au fait de diminuer les références à la section concernée, cela de manière à pouvoir libérer le fichier exécutable sitôt qu'il n'est plus employé. Voir routine PsReferenceProcessFilePointer.
- *Addresse de base de section (SectionBaseAddress)* Voir routine PsGetProcessSectionBaseAddress.

- **Domaine de sécurité (SecurityDomain)** Domaine de sécurité auquel se rattache un processus, ce qui sert à contrôler les interactions inter-processus en déterminant si deux processus relèvent du même domaine, condition requise pour certaines opérations sensibles, telles que la duplication de handles.
- **Port de sécurité (SecurityPort)** Voir routines PsGetProcessSecurityPort et PsSetProcessSecurityPort.
- *Session* Pointeur vers une structure MM_SESSION_SPACE qui contient des informations sur la session utilisateur et les objets graphiques (GUI). Voir routine PsGetCurrentProcessSessionId et MmGetSessionId.
- *SessionProcessLinks* Entrée parmi une liste doublement chainée qui permet de relier entre eux les différents processus d'une même session.
- *Flag résolution de la minuterie (SetTimerResolution)* Enregistre l'appel du processus à une fonction de manipulation de la résolution du timer système. Voir fonctions timeBeginPeriod et timeEndPeriod ; service NtSetTimerResolution ; routine ExSetTimerResolution.
- **Résolution minimale de minuterie (SmallestTimerResolution)** Voir service NtSetTimerResolution.
- *StackRandomizationDisabled* Spécifie la la désactivation des stratégies de placement aléatoire des zones de données de la mémoire virtuelle. Voir section Distribution aléatoire de l'espace d'adressage.
- **SwapListEntry** Voir variables noyau KiProcessInSwapListHead et KiProcessOutSwapListHead.
- **Liste des threads (ThreadListHead)** Entête de la liste des threads s'exécutant dans le contexte du processus (chaque entrée de la liste correspond à un bloc ETHREAD). Voir routine PsGetNextProcessThread.
- **Révision de l'horloge système (TimerResolutionLink)** Élément de liste servant à regrouper tous les processus qui ont sollicité une révision de l'horloge système.
- **Jeton d'accès (Token)** Informations liées à la sécurité et aux autorisations du processus, notamment les droits qui lui sont accordés. Voir routine PsReferencePrimaryToken.
- **Temps de fonctionnement effectif (TotalUnbiasedFrozenTime)** Temps total pendant lequel un processus a été gelé ou suspendu, mesuré sans tenir compte des biais liées à l'horloge système ou aux ajustements de temps.
- **ID unique (UniqueProcessID)** Code unique attribué à tout processus afin de l'identifier auprès des commandes et interfaces s'appliquant à un processus donné. Voir routines ExCreateHandle, PspCreateProcess, PsLookupProcessByProcessId, PsGetProcessId.
- **Temps utilisateur (UserTime)** Voir fonction GetProcessTimes ; service NtQueryInformationProcess (ProcessInformationClass = ProcessTimes).
- **Plages d'adresses (VadCount)** Nombre de plages d'adresses virtuelles actuellement rattachées à un processus.
- **Arbre VAD (VadRoot)** Contient des informations détaillées en ce qui concerne les plages d'adresses virtuelles allouées à un processus. Chaque VAD représente une zone mémoire contiguë dans l'espace d'adressage associé au processus, par exemple une zone contenant du code machine exécutable, une zone hébergeant des données, ou une section mappée depuis un fichier.
- **Exécution 16 bits (VdmAllowed)** Indique si le processus est autorisé à exécuter du code 16 bits via la machine DOS virtuelle (VDM, Virtual DOS machine), témoignant de la capacité du système à offrir un environnement compatible aux applications héritées. Voir service NtQueryInfomationProcess (ProcessInformationClass = ProcessWx86Information).
- *Mémoire virtuelle (Vm)* Voir routine MmSetMemoryPriorityProcess.
- *Adresses virtuelles hautes (VmTopDown)* Force les allocations mémoire à prendre leur essor à partir d'adresses virtuelles hautes. Voir fonctions VirtualAlloc, VirtualAllocEx et VirtualAllocExNuma (flAllocationType

= MEM_TOP_DOWN) ; services NtAllocateVirtualMemory et NtMapViewOfSection (AllocationType = MEM_TOP_DOWN)

- **Interface Windows (Win32Process)** Représentation d'un processus au niveau de la portion graphique du sous-système Windows (`win32k.sys`), incluant fenêtres, objets GDI et autres. Dans l'éventualité où cet attribut n'est pas défini, cela signifie que le processus n'a pas de composant Win32 associé, par exemple un pilote exécuté entièrement en mode noyau, ou un service dépourvu d'interface utilisateur. Voir routines PsGetCurrentProcessWin32Process, PsGetProcessWin32Process et PsSetProcessWin32Process.

- **Station fenêtre (*Win32WindowStation*)** Station fenêtre où est exécuté le processus. Voir routines PsGetProcessWin32WindowStation et PsSetProcessWindowStation.

- **Suivi de l'ensemble de travail (WorkingSetWatch)** Historique des défauts de page. Voir routine PspQueryWorkingSetWatch.

- **Compteur d'E/S écriture (WriteOperationCount)** Nombre d'opérations d'E/S écriture générées par le processus depuis son démarrage. Voir fonction GetProcessIoCounters ; mesure Écritures d'E/S dans le Gestionnaire des taches.

- **Compteur d'octets en écriture (WriteTransferCount)** Nombre d'octets écrits par le processus lors de ses opérations d'écriture depuis le début de son exécution jusqu'à la période actuelle. Voir fonction GetProcessIoCounters.

- **Espace d'adressage démantelé (VmDeleted)** Indique que l'espace d'adressage virtuel auquel se rapporte un processus n'est plus valide, empêchant de la sorte le noyau d'entreprendre toute opération ultérieure le concernant.

- **Ensemble de travail minimum (Vm.MinimumWorkingSetSize)** Nombre minimum de pages dont le processus est théoriquement certain de disposer en mémoire. Voir fonctions GetProcessWorkingSetSize et SetProcessWorkingSetSize ; service NtQueryInformationProcess (ProcessInformationClass = ProcessQuotaLimits).

- **Ensemble de travail maximum (Vm.MaximumWorkingSetSize)** Voir fonctions GetProcessWorkingSetSize et SetProcessWorkingSetSize ; service NtQueryInformationProcess (ProcessInformationClass = ProcessQuotaLimits).

- **Processus minimal (Minimal)** Indique que le processus est seulement doté d'un espace d'adressage en mode utilisateur.

Tableau 8.1. EPROCESS ExitStatus

Constante	Valeur
STATUS_ACCESS_VIOLATION	0xC00000E8
STATUS_ARRAY_BOUNDS_EXCEEDED	0xC000008C
STATUS_BREAKPOINT	0x80000003
STATUS_CONTROL_C_EXIT	0xC000013A
STATUS_DATATYPE_MISALIGNMENT	0x80000002
STATUS_FLOAT_DENORMAL_OPERAND	0xC000008D
STATUS_FLOAT_DIVIDE_BY_ZERO	0xC000008E
STATUS_FLOAT_INEXACT_RESULT	0xC000008F
STATUS_FLOAT_INVALID_OPERATION	0xC0000090
STATUS_FLOAT_MULTIPLE_FAULTS	0xC00002B4
STATUS_FLOAT_MULTIPLE_TRAPS	0xC00002B5
STATUS_FLOAT_OVERFLOW	0xC0000091
STATUS_FLOAT_STACK_CHECK	0xC0000092
STATUS_FLOAT_UNDERFLOW	0xC0000093
STATUS_GUARD_PAGE_VIOLATION	0x80000001

STATUS_ILLEGAL_INSTRUCTION	0xC000001D
STATUS_INTEGER_DIVIDE_BY_ZERO	0xC0000094
STATUS_INTEGER_OVERFLOW	0xC0000095
STATUS_INVALID_DISPOSITION	0xC0000026
STATUS_IN_PAGE_ERROR	0xC0000006
STATUS_NONCONTINUABLE_EXCEPTION	0xC0000025
STATUS_PRIVILEGED_INSTRUCTION	0xC0000096
STATUS_REG_NAT_CONSUMPTION	0xC00002C9
STATUS_SINGLE_STEP	0x80000004
STATUS_STACK_OVERFLOW	0xC00000FD

Objet processus du noyau (KPROCESS)

Le bloc KPROCESS fait partie du bloc EPROCESS - c'en est du reste une sous-structure, et même le premier constituant. La position en mémoire d'un bloc KPROCESS est par conséquent la même que celle du bloc EPROCESS qui l'héberge. (Pour connaître les adresses de tous les blocs EPROCESS du système, employez la commande *!process 0 0*.)

```
lkd> dt nt!_KPROCESS
   +0x000 Header           : _DISPATCHER_HEADER
   +0x018 ProfileListHead  : _LIST_ENTRY
   +0x028 DirectoryTableBase : Uint8B
   +0x030 ThreadListHead   : _LIST_ENTRY
   +0x040 ProcessLock      : Uint4B
   +0x044 Spare0           : Uint4B
   +0x048 DeepFreezeStartTime : Uint8B
   +0x050 Affinity         : _KAFFINITY_EX
   +0x0f8 ReadyListHead    : _LIST_ENTRY
   +0x108 SwapListEntry    : _SINGLE_LIST_ENTRY
   +0x110 ActiveProcessors : _KAFFINITY_EX
   +0x1b8 AutoAlignment    : Pos 0, 1 Bit
   +0x1b8 DisableBoost     : Pos 1, 1 Bit
   +0x1b8 DisableQuantum   : Pos 2, 1 Bit
   +0x1b8 DeepFreeze       : Pos 3, 1 Bit
   +0x1b8 TimerVirtualization : Pos 4, 1 Bit
   +0x1b8 CheckStackExtents : Pos 5, 1 Bit
   +0x1b8 SpareFlags0      : Pos 6, 2 Bits
   +0x1b8 ActiveGroupsMask : Pos 8, 20 Bits
   +0x1b8 ReservedFlags    : Pos 28, 4 Bits
   +0x1b8 ProcessFlags     : Int4B
   +0x1bc BasePriority     : Char
   +0x1bd QuantumReset     : Char
   +0x1be Visited          : UChar
   +0x1bf Flags            : _KEXECUTE_OPTIONS
   +0x1c0 ThreadSeed       : [20] Uint4B
   +0x210 IdealNode        : [20] Uint2B
   +0x238 IdealGlobalNode  : Uint2B
   +0x23a Spare1           : Uint2B
   +0x23c StackCount       : _KSTACK_COUNT
   +0x240 ProcessListEntry : _LIST_ENTRY
   +0x250 CycleTime        : Uint8B
   +0x258 ContextSwitches  : Uint8B
   +0x260 SchedulingGroup  : Ptr64 _KSCHEDULING_GROUP
   +0x268 FreezeCount      : Uint4B
   +0x26c KernelTime       : Uint4B
   +0x270 UserTime         : Uint4B
   +0x274 LdtFreeSelectorHint : Uint2B
   +0x276 LdtTableLength   : Uint2B
   +0x278 LdtSystemDescriptor : _KGDTENTRY64
   +0x288 LdtBaseAddress   : Ptr64 Void
   +0x290 LdtProcessLock   : _FAST_MUTEX
```

```
+0x2c8 InstrumentationCallback : Ptr64 Void
+0x2d0 SecurePid        : Uint8B
```

- *Masque d'affinité (Affinity)* Masque binaire des processeurs sur lesquels les threads du processus sont autorisés à fonctionner. Voir services KeQueryAffinityProcess et KeSetAffinityProcess.
- *AutoAlignment* Définit le mode de gestion de l'alignement des données pour le processus. Voir routine KeSetAutoAlignmentProcess.
- **Mode d'alignement (AutoAlignment)** Indique si les erreurs d'alignement mémoire d'un processus sont corrigées automatiquement. Sur certaines architectures, notamment x86 et x64, accéder à des données qui ne sont pas alignées correctement (par exemple, lire un entier de 4 octets à une adresse qui n'est pas un multiple de 4) peut entraîner une exception matérielle. Le mode automatique court-circuite ce schéma, le système corrigeant alors les anomalies sans générer une exception, ce qui permet en définitive au processus d'aller de l'avant. Voir PROCESS_HARDERROR_ALIGNMENT_BIT, se rapportant à EPROCESS DefaultHardErrorProcessing. Voir ProcessEnableAlignmentFaultFixup dans PROCESS_INFORMATION_CLASS. Voir SetErrorMode(SEM_NOALIGNMENTFAULTEXCEPT).
- *Priorité de base (BasePriority)* Priorité d'ordonnancement de base par défaut du processus. Définit la priorité de départ pour l'ensemble des threads créés dans ce processus. Voir routine KeSetPriorityAndQuantumProcess.
- *Nombre de basculements de contexte (ContextSwitches)* Nombre total de basculements de contexte qui se sont produits pour un processus et l'ensemble de ses threads.
- **Table des pages (DirectoryTableBase)** Adresse de base de la table des pages associée au processus, utilisée pour gérer la traduction des adresses virtuelles en adresses physiques et définir l'espace d'adressage virtuel dédié au processus.
- **Contrôle d'accélération de priorité (DisableBoost)** Détermine si oui ou non le processus est éligible en tant que bénéficiaire d'une accélération de la priorité de ses threads. Pour plus d'informations, voir section la section intitulée « Accélérations de priorité ».
- *Quantum actif ou non (DisableQuantum)* Donne un quantum infini aux threads s'exécutant dans le contexte de ce processus (à condition que ce processus soit associé à la priorité temps réel). Voir routine KeSetDisableQuantumProcess.
- En-tête dispatcher (Header) :: Structure de données qui encapsule les capacités de synchronisation que le noyau confère au processus : état de l'objet (signalé ou non), liste des threads en attente sur lui, etc.
- *Noeud préféré (IdealNode)* Sur les systèmes NUMA, indique la sélection du noeud idéal pour le processus.
- *InstrumentationCallback* Pointeur vers une routine à appeler lors d'une transition du processus vers le mode utilisateur. Voir service NtSetInformationProcess (ProcessInfoClass = 0x28).
- **Temps noyau (KernelTime)** Durée pendant laquelle les threads d'un processus ont été exécutés en mode noyau. Ce temps inclut notamment les appels système, le traitement des interruptions, ainsi que l'exécution de routines différées (comme les DPC). Autre mesure corrélée de manière significative à celle-ci : le temps utilisateur, comptabilisé séparément via le champ du même nom (`UserTime`). Les deux attributs, une fois cumulés, permettent d'avoir une vue d'ensemble du temps processeur qu'a consommé un processus depuis son démarrage. Voir fonction GetProcessTimes ; services NtQueryInformationProcess (ProcessInformationClass = ProcessTimes), KeQueryRuntimeProcess et KeUpdateRunTime ; compteur de performance Processus: % Temps privilégié.
- *Adresse LDT (LdtBaseAddress)* Contient éventuellement un pointeur sur la table des descripteurs locaux (LDT) du processus.
- **Verrou LDT (LdtProcessLock)** Primitive de synchronisation, ici un spinlock, employée de sorte à synchroniser l'accès à la table des descripteurs locaux (LDT) associée à un processus.
- *Flags de processus (ProcessFlags)*

- *Quantum (QuantumReset)* Valeur initiale du quantum par défaut des threads du processus. Voir routine KeSetQuantumProcess.
- *Liste des threads prêts (ReadyListHead)* Indique quels threads de ce processus sont dans l'état d'ordonnancement Prêt, éligible à ce titre pour l'exécution.
- **PID sécurisé (SecurePid)** Identifiant unique spécifique aux processus sécurisés.
- *Compteur de pile (StackCount)* Compteur de pile noyau résident. Voir routines KeStartThread et KeTerminateThread.
- *Graine de thread (ThreadSeed)*
- **Temps utilisateur (UserTime)** Durée pendant laquelle les threads d'un processus ont été exécutés en mode utilisateur, c'est-à-dire en dehors du noyau. Ce temps couvre l'exécution d'opérations directement liées à l'application, telles que ses instructions internes. Autre mesure associée à celle-ci : le temps noyau (`KernelTime`). Les deux attributs, une fois cumulés, permettent d'avoir une vue d'ensemble du temps processeur qu'a consommé un processus depuis son démarrage. Voir service NtQueryInformationProcess (ProcessInformationClass = ProcessTimes) ; routines KeQueryRuntimeProcess et KeUpdateRunTime ; compteur de performance Processus: % Temps utilisateur.

Tableau 8.2. ProcessFlags

Constante	Valeur	Description
PS_PROCESS_FLAGS_CREATE_REPORTED	0x00000001	-
PS_PROCESS_FLAGS_NO_DEBUG_INHERIT	0x00000002	-
PS_PROCESS_FLAGS_PROCESS_EXITING	0x00000004	Voir routine PsGetProcessExitProcessCalled.
PS_PROCESS_FLAGS_PROCESS_DELETE	0x00000008	-
PS_PROCESS_FLAGS_WOW64_SPLIT_-PAGES	0x00000010	-
PS_PROCESS_FLAGS_VM_DELETED	0x00000020	L'espace de session auquel appartient le processus a été supprimé.
PS_PROCESS_FLAGS_OUTSWAP_ENABLED	0x00000040	-
PS_PROCESS_FLAGS_OUTSWAPPED	0x00000080	-
PS_PROCESS_FLAGS_FORK_FAILED	0x00000100	-
PS_PROCESS_FLAGS_WOW64_4GB_VA_-SPACE	0x00000200	-
PS_PROCESS_FLAGS_ADDRESS_SPACE1	0x00000400	-
PS_PROCESS_FLAGS_ADDRESS_SPACE2	0x00000800	-
PS_PROCESS_FLAGS_SET_TIMER_-RESOLUTION	0x00001000	-
PS_PROCESS_FLAGS_BREAK_ON_-TERMINATION	0x00002000	-
PS_PROCESS_FLAGS_CREATING_SESSION	0x00004000	-
PS_PROCESS_FLAGS_USING_WRITE_-WATCH	0x00008000	-
PS_PROCESS_FLAGS_IN_SESSION	0x00010000	Le processus fait partie d'un espace de session.
PS_PROCESS_FLAGS_OVERRIDE_-ADDRESS_SPACE	0x00020000	-
PS_PROCESS_FLAGS_HAS_ADDRESS_-SPACE	0x00040000	Le processus a un espace d'adressage. Voir routine PspCreateProcess.
PS_PROCESS_FLAGS_LAUNCH_-PREFETCHED	0x00080000	-
PS_PROCESS_INJECT_INPAGE_ERRORS	0x00100000	-
PS_PROCESS_FLAGS_VM_TOP_DOWN	0x00200000	-

PS_PROCESS_FLAGS_IMAGE_NOTIFY_-DONE	0x00400000	-
PS_PROCESS_FLAGS_PDE_UPDATE_-NEEDED	0x00800000	-
PS_PROCESS_FLAGS_VDM_ALLOWED	0x01000000	-
PS_PROCESS_FLAGS_SMAP_ALLOWED	0x02000000	-
PS_PROCESS_FLAGS_CREATE_FAILED	0x04000000	-

PEB (*Process Environment Block*)

Conjoint de chaque processus, le bloc d'environnement de processus (PEB, *Process Environment Block*) contient des informations de contexte destinées au chargeur d'image, au gestionnaire de tas et à d'autres DLL système de Windows. Étroitement lié aux représentations des processus dans l'exécutif et dans le sous-système client-serveur (Csrss), qui toutes ne sont accessibles que depuis le mode noyau, le PEB existe dans l'espace d'adressage du processus. En effet, conçu pour être utilisé par des composants en dehors de ceux dont l'exécution est privilégiée, le PEB contient des données modifiables depuis le mode utilisateur, lisibles rapidement sans le concours des services système. Cela inclut des informations d'image (adresse de base, numéros de version), la liste des modules chargés en mémoire, des éléments se rapportant à la gestion du tas processus, ou encore les paramètres de démarrage et les variables d'environnement du programme.

Visualisation des données du bloc PEB

Pour voir concrètement quels attributs sont enregistrés à l'échelle du bloc d'environnement d'un processus, utilisez la commande dt des débogueurs Windows standards.

```
lkd> dtx nt!_PEB 0000003819f49000
(*((nt!_PEB *)0x3819f49000))                [Type: _PEB]
    [+0x000] InheritedAddressSpace : 0x0 [Type: unsigned char]
    [+0x001] ReadImageFileExecOptions : 0x0 [Type: unsigned char]
    [+0x002] BeingDebugged    : 0x0 [Type: unsigned char]
    [+0x003] BitField         : 0x84 [Type: unsigned char]
    [+0x003 ( 0: 0)] ImageUsesLargePages : 0x0 [Type: unsigned char]
    [+0x003 ( 1: 1)] IsProtectedProcess : 0x0 [Type: unsigned char]
    [+0x003 ( 2: 2)] IsImageDynamicallyRelocated : 0x1 [Type: unsigned char]
    [+0x003 ( 3: 3)] SkipPatchingUser32Forwarders : 0x0 [Type: unsigned char]
    [+0x003 ( 4: 4)] IsPackagedProcess : 0x0 [Type: unsigned char]
    [+0x003 ( 5: 5)] IsAppContainer   : 0x0 [Type: unsigned char]
    [+0x003 ( 6: 6)] IsProtectedProcessLight : 0x0 [Type: unsigned char]
    [+0x003 ( 7: 7)] IsLongPathAwareProcess : 0x1 [Type: unsigned char]
    [+0x004] Padding0         [Type: unsigned char [4]]
    [+0x008] Mutant           : 0xffffffffffffffff [Type: void *]
    [+0x010] ImageBaseAddress : 0x7ff74bb20000 [Type: void *]
    [+0x018] Ldr              : 0x7ffde3a053c0 [Type: _PEB_LDR_DATA *]
    [+0x020] ProcessParameters : 0x20a70581c00 [Type: _RTL_USER_PROCESS_PARAMETERS *]
    [+0x028] SubSystemData    : 0x7ffddeb3e120 [Type: void *]
    [+0x030] ProcessHeap      : 0x20a70580000 [Type: void *]
    [+0x038] FastPebLock      : 0x7ffde3a04fe0 [Type: _RTL_CRITICAL_SECTION *]
    [+0x040] AtlThunkSListPtr : 0x0 [Type: _SLIST_HEADER *]
    [+0x048] IFEOKey          : 0x0 [Type: void *]
    [+0x050] CrossProcessFlags : 0x0 [Type: unsigned long]
    [+0x050 ( 0: 0)] ProcessInJob     : 0x0 [Type: unsigned long]
    [+0x050 ( 1: 1)] ProcessInitializing : 0x0 [Type: unsigned long]
    [+0x050 ( 2: 2)] ProcessUsingVEH  : 0x0 [Type: unsigned long]
    [+0x050 ( 3: 3)] ProcessUsingVCH  : 0x0 [Type: unsigned long]
    [+0x050 ( 4: 4)] ProcessUsingFTH  : 0x0 [Type: unsigned long]
    [+0x050 ( 5: 5)] ProcessPreviouslyThrottled : 0x0 [Type: unsigned long]
    [+0x050 ( 6: 6)] ProcessCurrentlyThrottled : 0x0 [Type: unsigned long]
    [+0x050 ( 7: 7)] ProcessImagesHotPatched : 0x0 [Type: unsigned long]
    [+0x050 (31: 8)] ReservedBits0    : 0x0 [Type: unsigned long]
    [+0x054] Padding1         [Type: unsigned char [4]]
    [+0x058] KernelCallbackTable : 0x7ffde2416330 [Type: void *]
```

```
[+0x058] UserSharedInfoPtr : 0x7ffde2416330 [Type: void *]
[+0x060] SystemReserved   : 0x0 [Type: unsigned long]
[+0x064] AtlThunkSListPtr32 : 0x0 [Type: unsigned long]
[+0x068] ApiSetMap        : 0x20a70490000 [Type: void *]
[+0x070] TlsExpansionCounter : 0x0 [Type: unsigned long]
[+0x074] Padding2         [Type: unsigned char [4]]
[+0x078] TlsBitmap        : 0x7ffde3a05340 [Type: void *]
[+0x080] TlsBitmapBits    [Type: unsigned long [2]]
[+0x088] ReadOnlySharedMemoryBase : 0x7df4edfa0000 [Type: void *]
[+0x090] SharedData       : 0x0 [Type: void *]
[+0x098] ReadOnlyStaticServerData : 0x7df4edfa0750 [Type: void * *]
[+0x0a0] AnsiCodePageData : 0x7df5f00e0000 [Type: void *]
[+0x0a8] OemCodePageData  : 0x7df5f00f0228 [Type: void *]
[+0x0b0] UnicodeCaseTableData : 0x7df5f0100650 [Type: void *]
[+0x0b8] NumberOfProcessors : 0x2 [Type: unsigned long]
[+0x0bc] NtGlobalFlag     : 0x0 [Type: unsigned long]
[+0x0c0] CriticalSectionTimeout : {-25920000000000} [Type: _LARGE_INTEGER]
[+0x0c8] HeapSegmentReserve : 0x100000 [Type: unsigned __int64]
[+0x0d0] HeapSegmentCommit : 0x2000 [Type: unsigned __int64]
[+0x0d8] HeapDeCommitTotalFreeThreshold : 0x10000 [Type: unsigned __int64]
[+0x0e0] HeapDeCommitFreeBlockThreshold : 0x1000 [Type: unsigned __int64]
[+0x0e8] NumberOfHeaps    : 0x4 [Type: unsigned long]
[+0x0ec] MaximumNumberOfHeaps : 0x10 [Type: unsigned long]
[+0x0f0] ProcessHeaps     : 0x7ffde3a03c40 [Type: void * *]
[+0x0f8] GdiSharedHandleTable : 0x20a70950000 [Type: void *]
[+0x100] ProcessStarterHelper : 0x0 [Type: void *]
[+0x108] GdiDCAttributeList : 0x14 [Type: unsigned long]
[+0x10c] Padding3         [Type: unsigned char [4]]
[+0x110] LoaderLock       : 0x7ffde39ff4f8 [Type: _RTL_CRITICAL_SECTION *]
[+0x118] OSMajorVersion   : 0xa [Type: unsigned long]
[+0x11c] OSMinorVersion   : 0x0 [Type: unsigned long]
[+0x120] OSBuildNumber    : 0x47ba [Type: unsigned short]
[+0x122] OSCSDVersion     : 0x0 [Type: unsigned short]
[+0x124] OSPlatformId     : 0x2 [Type: unsigned long]
[+0x128] ImageSubsystem   : 0x2 [Type: unsigned long]
[+0x12c] ImageSubsystemMajorVersion : 0xa [Type: unsigned long]
[+0x130] ImageSubsystemMinorVersion : 0x0 [Type: unsigned long]
[+0x134] Padding4         [Type: unsigned char [4]]
[+0x138] ActiveProcessAffinityMask : 0x3 [Type: unsigned __int64]
[+0x140] GdiHandleBuffer  [Type: unsigned long [60]]
[+0x230] PostProcessInitRoutine : 0x0 [Type: void (__cdecl*)()]
[+0x238] TlsExpansionBitmap : 0x7ffde3a05320 [Type: void *]
[+0x240] TlsExpansionBitmapBits [Type: unsigned long [32]]
[+0x2c0] SessionId        : 0x1 [Type: unsigned long]
[+0x2c4] Padding5         [Type: unsigned char [4]]
[+0x2c8] AppCompatFlags   : {0x0} [Type: _ULARGE_INTEGER]
[+0x2d0] AppCompatFlagsUser : {0x0} [Type: _ULARGE_INTEGER]
[+0x2d8] pShimData        : 0x20a704d0000 [Type: void *]
[+0x2e0] AppCompatInfo    : 0x0 [Type: void *]
[+0x2e8] CSDVersion       : "" [Type: _UNICODE_STRING]
[+0x2f8] ActivationContextData : 0x20a704c0000 [Type: _ACTIVATION_CONTEXT_DATA *]
[+0x300] ProcessAssemblyStorageMap : 0x20a7058af50 [Type: _ASSEMBLY_STORAGE_MAP *]
[+0x308] SystemDefaultActivationContextData : 0x20a704b0000 [Type: _ACTIVATION_CONTEXT_DATA *]
[+0x310] SystemAssemblyStorageMap : 0x0 [Type: _ASSEMBLY_STORAGE_MAP *]
[+0x318] MinimumStackCommit : 0x0 [Type: unsigned __int64]
[+0x320] SparePointers    [Type: void * [4]]
[+0x340] SpareUlongs      [Type: unsigned long [5]]
[+0x358] WerRegistrationData : 0x0 [Type: void *]
[+0x360] WerShipAssertPtr : 0x0 [Type: void *]
[+0x368] pUnused          : 0x0 [Type: void *]
[+0x370] pImageHeaderHash : 0x0 [Type: void *]
[+0x378] TracingFlags     : 0x0 [Type: unsigned long]
[+0x378 ( 0: 0)] HeapTracingEnabled : 0x0 [Type: unsigned long]
[+0x378 ( 1: 1)] CritSecTracingEnabled : 0x0 [Type: unsigned long]
[+0x378 ( 2: 2)] LibLoaderTracingEnabled : 0x0 [Type: unsigned long]
```

```
    [+0x378 (31: 3)] SpareTracingBits : 0x0 [Type: unsigned long]
    [+0x37c] Padding6         [Type: unsigned char [4]]
    [+0x380] CsrServerReadOnlySharedMemoryBase : 0x7df4e0ee0000 [Type: unsigned __int64]
    [+0x388] TppWorkerpListLock : 0x0 [Type: unsigned __int64]
    [+0x390] TppWorkerpList   [Type: _LIST_ENTRY]
    [+0x3a0] WaitOnAddressHashTable [Type: void * [128]]
    [+0x7a0] TelemetryCoverageHeader : 0x0 [Type: void *]
    [+0x7a8] CloudFileFlags   : 0x0 [Type: unsigned long]
    [+0x7ac] CloudFileDiagFlags : 0x0 [Type: unsigned long]
    [+0x7b0] PlaceholderCompatibilityMode : 0 [Type: char]
    [+0x7b1] PlaceholderCompatibilityModeReserved : "" [Type: char [7]]
    [+0x7b8] LeapSecondData   : 0x7df5f00d0000 [Type: _LEAP_SECOND_DATA *]
    [+0x7c0] LeapSecondFlags  : 0x0 [Type: unsigned long]
    [+0x7c0 ( 0: 0)] SixtySecondEnabled : 0x0 [Type: unsigned long]
    [+0x7c0 (31: 1)] Reserved         : 0x0 [Type: unsigned long]
    [+0x7c4] NtGlobalFlag2    : 0x0 [Type: unsigned long]
```

Examen du bloc PEB

Vous pouvez afficher la structure PEB avec la commande !peb des débogueurs Windows standards. Pour obtenir l'adresse du PEB, utilisez la commande !process comme suit :

```
lkd> !process 0 0 cmd.exe
PROCESS ffffe000873b6080
    SessionId: 1  Cid: 0f64    Peb: f2e5583000  ParentCid: 08bc
    DirBase: 14a83f000  ObjectTable: ffffc000a2038a40  HandleCount: <Data Not Accessible>
    Image: cmd.exe
```

Ensuite, spécifiez cette adresse à la commande !peb comme suit :

```
lkd> .process /p ffffe000873b6080
Implicit process is now ffffe000`873b6080

lkd> !peb f2e5583000
PEB at 000000f2e5583000
    InheritedAddressSpace:    No
    ReadImageFileExecOptions: No
    BeingDebugged:            No
    ImageBaseAddress:         00007ff7d4130000
    Ldr                       00007ffa675a5200
    Ldr.Initialized:          Yes
    Ldr.InInitializationOrderModuleList: 000002c738a91a80 . 000002c738a96570
    Ldr.InLoadOrderModuleList:           000002c738a91be0 . 000002c738a96550
    Ldr.InMemoryOrderModuleList:         000002c738a91bf0 . 000002c738a96560
                    Base TimeStamp                     Module
            7ff7d4130000 5632d733 Oct 30 03:34:27 2015 C:\windows\system32\cmd.exe
            7ffa67460000 571af2eb Apr 23 05:58:35 2016 C:\windows\SYSTEM32\ntdll.dll
            7ffa66ed0000 5632d5aa Oct 30 03:27:54 2015 C:\windows\system32\KERNEL32.DLL
            7ffa644e0000 571af331 Apr 23 05:59:45 2016 C:\windows\system32\KERNELBASE.dll
            7ffa671c0000 5632d79e Oct 30 03:36:14 2015 C:\windows\system32\msvcrt.dll
            7ffa4cbd0000 5632d813 Oct 30 03:38:11 2015 C:\windows\SYSTEM32\winbrand.dll
    SubSystemData:     0000000000000000
    ProcessHeap:       000002c738a90000
    ProcessParameters: 000002c738a91270
    CurrentDirectory:  'C:\Users\arnaud\'
    WindowTitle:  'C:\windows\system32\cmd.exe'
    ImageFile:    'C:\windows\system32\cmd.exe'
    CommandLine:  '"C:\windows\system32\cmd.exe" '
    DllPath:      '< Name not readable >'
    Environment:  000002c738a958e0
        =::=::\
        =C:=C:\Users\arnaud
        ALLUSERSPROFILE=C:\ProgramData
        APPDATA=C:\Users\arnaud\AppData\Roaming
```

```
CommonProgramFiles=C:\Program Files\Common Files
CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files
CommonProgramW6432=C:\Program Files\Common Files
COMPUTERNAME=DESKTOP-JEROKEA
ComSpec=C:\windows\system32\cmd.exe
HOMEDRIVE=C:
HOMEPATH=\Users\arnaud
LOCALAPPDATA=C:\Users\arnaud\AppData\Local
LOGONSERVER=\\DESKTOP-JEROKEA
NUMBER_OF_PROCESSORS=4
OS=Windows_NT
Path=C:\windows\system32;C:\windows;C:\windows\System32\Wbem;C:\windows\System32\WindowsPowerShell\v1.0\
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE=AMD64
PROCESSOR_IDENTIFIER=Intel64 Family 6 Model 78 Stepping 3, GenuineIntel
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=4e03
ProgramData=C:\ProgramData
ProgramFiles=C:\Program Files
ProgramFiles(x86)=C:\Program Files (x86)
ProgramW6432=C:\Program Files
PROMPT=$P$G
PSModulePath=C:\Program Files\WindowsPowerShell\Modules;C:\windows\system32\WindowsPowerShell\v1.0\Modules
PUBLIC=C:\Users\Public
SESSIONNAME=Console
SystemDrive=C:
SystemRoot=C:\windows
TEMP=C:\Users\arnaud\AppData\Local\Temp
TMP=C:\Users\arnaud\AppData\Local\Temp
USERDOMAIN=DESKTOP-JEROKEA
USERDOMAIN_ROAMINGPROFILE=DESKTOP-JEROKEA
USERNAME=arnaud
USERPROFILE=C:\Users\arnaud
windir=C:\windows
```

Attributs du bloc PEB

Les attributs enregistrés à l'échelle du bloc PEB d'un processus incluent ceux que voici.

- **Masque d'affinité de processus image (ActiveProcessAffinityMask)**
- **Page de code ANSI (AnsiCodePageData)** Page de code utilisée utilisée pour les applications graphiques. Voir routine MmCreatePeb.
- **Indicateurs de compatibilité (AppCompatFlags)** Indicateurs visant à déterminer quels correctifs de compatibilité doivent être appliqués à une application.
- **Flag de débogage (BeingDebugged)** Indique si le processus est exécuté sous le contrôle d'un débogueur. Voir fonction IsDebuggerPresent.
- **Délai de section critique (CriticalSectionTimeout)** Limite le temps que peut passer le processus à attendre une section critique. Voir variable noyau MmCriticalSectionTimeout.
- **Version de maintenance (CSDVersion)** Nom de la version de maintenance la plus récemment installé, par exemple « Service Pack 1 ». Voir structure OSVERSIONINFOEX (szCSDVersion)
- **Synchronisation (FastPebLock)** Primitive de synchronisation utilisable pour sérialiser les opérations de lecture/écriture depuis et vers la structure PEB, de sorte que celles-ci s'effectuent de manière sûre et coordonnée. RtlAcquirePebLock et RtlReleasePebLock.
- **GdiHandleBuffer** Tient lieu de base de données utilisée par le sous-système graphique pour stocker les handles les plus fréquemment utilisés par le processus (au lieu de lire dans la table des handles).

- **Table des handles GDI partagés (GdiSharedHandleTable)** Handles des objets GDI créés dans la session à laquelle appartient le processus.
- **HeapDeCommitFreeBlockThreshold** Voir MmHeapDeCommitFreeBlockThreshold.
- **HeapDeCommitTotalFreeThreshold** Voir MmHeapDeCommitTotalFreeThreshold.
- **Sous-système (ImageSubsystem)** Environnement d'exécution attendu par un processus pour fonctionner correctement. Ce champ précise le type de sous-système nécessaire, comme une interface graphique, une console, ou un mode natif.
- **Version majeure du sous-système (ImageSubsystemMajorVersion)** Version majeure du sous-système requis par un processus, permettant de définir des contraintes de compatibilité essentielles avec l'environnement cible.
- **Version mineure du sous-système (ImageSubsystemMinorVersion)** version mineure du sous-système requis, ajoutant des exigences supplémentaires en matière de compatibilité. Par exemple, une combinaison de ImageSubsystemMajorVersion = 6 et ImageSubsystemMinorVersion = 1 correspond au sous-système Windows tel que l'intègre Windows 7.
- **Grandes pages (ImageUsesLargePages)** Indique si le processus fait usage de pages mémoire de grande taille.
- **Processus cloné (InheritedAddressSpace)** Indique si le processus a été cloné à partir d'un autre processus (RtlCloneUserProcess), impliquant que l'espace d'adressage du processus cloné est hérité de l'espace d'adressage du processus d'origine.
- **Modèle d'exécution (IsAppContainer)** Indique si le processus s'exécute ou non dans l'environnement sécurisé AppContainer.
- **Adresse de base (ImageBaseAddress)** Adresse à laquelle le fichier exécutable sous-jacent au processus a été chargée en mémoire. Utilisée pour calculer les adresses relatives des fonctions et données du programme. Voir champ SectionBaseAddress de la structure EPROCESS.
- **Réallocation dynamique (IsImageDynamicallyRelocated)** Indique si l'image exécutable sous-jacente au processus actuel a été a chargé à une adresse différente de son adresse de base préférée - à cause, par exemple, de l'ASLR ou d'un conflit avec un autre module. Lorsque cela se produit, Windows doit alors actualiser certaines adresses, notamment les références aux données et aux fonctions internes de l'exécutable, pour que elles-ci restent valides. En interne, de telles ajustements sont possibles à l'aide de la table de relocation de l'exécutable.
- **Chemins de fichiers longs (IsLongPathAwareProcess)** Indique si le processus peut gérer des chemins de fichiers longs, dépassant la limite classique de 260 caractères. Voir fonction RtlAreLongPathsEnabled.
- **Processus packagé (IsPackagedProcess)** Indique si un processus émane d'une application UWP ou d'un autre type d'application packagé, telles que celles distribuées via le Microsoft Store, et est à cet égard soumis à des restrictions d'accès aux ressources système plus strictes par rapport aux applications classiques de bureau.
- **Processus Protégé (IsProtectedProcess)** Limite (par le biais des droits d'accès accordés) les interactions que peuvent avoir sur celui-ci les autres processus du système.
- **Protection PPL (IsProtectedProcessLight)** Indique si le processus est protégé via le mécanisme Protected Process Light (PPL), une forme allégée de protection des processus introduite dans Windows pour renforcer la sécurité tout en autorisant certaines extensions contrôlées - notamment divers pilotes ou DLL signés.
- **Table des fonctions de rappel noyau-utilisateur (KernelCallbackTable)** Pointeur vers une table utilisée par la portion mode noyau du sous-système Windows (Win32k.sys) pour faciliter l'appel des procédures de fenêtre d'une interface graphique à partir d'un pilote de périphérique.

- **Synchronisation de niveau image (LoaderLock)** Section critique utilisée par le chargeur d'image lors de la création de processus.
- **Données du chargeur (Ldr)** Pointeur sur une structure de données PEB_LDR_DATA qui contient des pointeurs ou des références vers les modules (DLL) chargés dans l'espace d'adressage mode utilisateur du processus. Voir fonction LdrpInitializeProcess.
- **Nombre maximum de tas (MaximumNumberOfHeaps)** Nombre maximal de tas que le processus peut réclamer.
- **MinimumStackCommit** Voir routine MmCreatePeb, variable noyau MmMinimumStackCommitInBytes.
- **NtGlobalFlag** Indicateur global contrôlant certaines options de débogage ou de gestion des erreurs au niveau du processus, comme les vérifications de pile ou les erreurs fatales de mémoire.
- **Nombre de tas (NumberOfHeaps)** Nombre de tas que le processus a en sa possession.
- **Nombre de processeurs (NumberOfProcessors)** Nombre de processeurs sur lesquels le processus est autorisé à s'exécuter. Voir routine système *MmCreatePeb* et variable noyau *KeNumberProcessors*.
- **Page de code OEM (OemCodePageData)** Page de code OEM active. Les pages de code OEM, historiquement des jeux de caractères issus de MS-DOS, sont aujourd'hui utilisés principalement dans les consoles (cmd.exe) et pour la compatibilité avec des applications héritées. Peu utile pour les applications modernes, qui s'appuient sur Unicode (UTF-16). Voir fonction GetOEMCP.
- **OSBuildNumber** Voir fonction RtlGetNtVersionNumbers.
- **Version majeure du système (OSMajorVersion)** Partie d'un schéma de numérotation (ici, numéro majeur) servant à distinguer laquelle, parmi les déclinaisons générales de Microsoft Windows (XP, Vista, 7, 8, 10, 11, etc.) est en cours de fonctionnement sur la station de travail. Voir fonction RtlGetNtVersionNumbers, variable noyau NtMajorVersion.
- **Version mineure du système (OSMinorVersion)** Sous-partie d'un schéma de numérotation (ici, numéro mineur) indiquant quelle révision mineure déclinaison de Windows est en cours d'exécution sur la plateforme. Voir fonction RtlGetNtVersionNumbers, variable noyau NtMinorVersion.
- **Identifiant de plateforme (OSPlatformId)** Valeur permettant de différencier les systèmes Windows basés sur NT (qui incluent toutes les versions récentes) de ceux fondées sur DOS. Dans les déclinaisons modernes de Windows (XP et ultérieures), ce champ est toujours défini à `VER_PLATFORM_WIN32_NT` (0x2).
- **Routine post-initialisation (PostProcessInitRoutine)** Pointeur vers une fonction spécifique d'initialisation, exécutée après le chargement et le traitement de l'image principale du processus, permettant d'effectuer des ajustements complémentaires avant le démarrage effectif du code utilisateur.
- **Tas du processus (ProcessHeap)** Pointeur vers le tas mémoire principal du processus. Le tas principal est celui qui est généralement utilisé par défaut pour la plupart des allocations mémoire standard dans le processus.
- **Liste des tas du processus (ProcessHeaps)** Liste de pointeurs représentant tous les tas créés par le processus. En plus du tas principal, un processus peut créer et gérer plusieurs tas pour des besoins spécifiques, tels que l'isolation de certaines allocations ou l'amélioration des performances.
- **Flag En cours d'initialisation (ProcessInitializing)** Valeur booléenne que le chargeur emploie pour marquer le début (ProcessInitializing = 1) et la fin (ProcessInitializing = 0) des étapes d'initialisation d'un processus.
- **Marqueur de job (ProcessInJob)** Met en évidence l'appartenance du processus à un job.
- **Gestion des continuations vectorielles (ProcessUsingVCH)** Indique que le processus a enregistré au moins un gestionnaire de continuation vectorielle, lequel est appelé en dernier recours lorsque aucune autre

méthode de traitement des exception n'a fonctionné (VEH en premier lieu, SEH par la suite). Voir fonction AddVectoredContinueHandler.

- **Gestion des exceptions vectorielles (ProcessUsingVEH)** Signifie que le processus emploie la gestion vectorielle des exceptions (VEH, Vectored Exception Handling), en installant un ou plusieurs gestionnaires invoqués avant le mécanisme SEH classique. Voir fonction `AddVectoredExceptionHandler`.

- **Paramètres du processus (ProcessParameters)** Pointeur sur une structure de données RTL_USER_PROCESS_PARAMETERS qui comprend, entre autres, le chemin de DLL, le chemin de l'image exécutable et la ligne de commande ayant entrainé le processus. Voir fonction GetStartupInfo.

- **Contrôle des options d'exécution (ReadImageFileExecOptions)** Indique si les options d'exécution d'un processus, comme les options de compatibilité ou les paramètres de débogage, découlent de la configuration par défaut, ou si elles doivent être lues à partir à partir des métadonnées de l'exécutable associé.

- **PShimData** Informations utilisées par l'infrastructure shim intégrée à Microsoft .NET.

- **ID de session (SessionId)** ID de la session utilisateur à laquelle appartient un processus. Voir routine MmGetSessionId.

Tableau 8.3. AppCompatFlags

Valeur	Nom
0x00000001	KACF_OLDGETSHORTPATHNAME
0x00000002	KACF_VERSIONLIE_NOT_USED
0x00000008	KACF_GETDISKFREESPACE
0x00000020	KACF_FTMFROMCURRENTAPT
0x00000040	KACF_DISALLOWORBINDINGCHANGES
0x00000080	KACF_OLE32VALIDATEPTRS
0x00000100	KACF_DISABLECICERO
0x00000200	KACF_OLE32ENABLEASYNCDOCFILE
0x00000400	KACF_OLE32ENABLELEGACYEXCEPTIONHANDLING
0x00000800	KACF_RPCDISABLENDRCLIENTHARDENING
0x00001000	KACF_RPCDISABLENDRMAYBENULL_SIZEIS
0x00002000	KACF_DISABLEALLDDEHACK_NOT_USED
0x00004000	KACF_RPCDISABLENDR61_RANGE
0x00008000	KACF_RPC32ENABLELEGACYEXCEPTIONHANDLING
0x00010000	KACF_OLE32DOCFILEUSELEGACYNTFSFLAGS
0x00020000	KACF_RPCDISABLENDRCONSTIIDCHECK
0x00040000	KACF_USERDISABLEFORWARDERPATCH
0x00100000	KACF_OLE32DISABLENEW_WMPAINT_DISPATCH
0x00200000	KACF_ADDRESTRICTEDSIDINCOINITIALIZESECURITY
0x00400000	KACF_ALLOCDEBUGINFOFORCRITSECTIONS
0x00800000	KACF_OLEAUT32ENABLEUNSAFELOADTYPELIBRELATIVE
0x01000000	KACF_ALLOWMAXIMIZEDWINDOWGAMMA
0x80000000	KACF_DONOTADDTOCACHE

KernelCallbackTable.

```
0:007> dps 0x00007fff`8c1ef710
00007fff`8c1ef710  00007fff`8c166800 USER32!_fnCOPYDATA
00007fff`8c1ef718  00007fff`8c1da960 USER32!_fnCOPYGLOBALDATA
00007fff`8c1ef720  00007fff`8c1872f0 USER32!_fnDWORD
00007fff`8c1ef728  00007fff`8c1917c0 USER32!_fnNCDESTROY
00007fff`8c1ef730  00007fff`8c191560 USER32!_fnDWORDOPTINLPMSG
```

```
00007fff`8c1ef738  00007fff`8c1dab80 USER32!_fnINOUTDRAG
00007fff`8c1ef740  00007fff`8c191850 USER32!_fnGETTEXTLENGTHS
00007fff`8c1ef748  00007fff`8c19b480 USER32!_fnINCNTOUTSTRING
00007fff`8c1ef750  00007fff`8c164bd0 USER32!_fnINCNTOUTSTRINGNULL
00007fff`8c1ef758  00007fff`8c19b760 USER32!_fnINLPCOMPAREITEMSTRUCT
00007fff`8c1ef760  00007fff`8c166700 USER32!__fnINLPCREATESTRUCT
00007fff`8c1ef768  00007fff`8c1963a0 USER32!_fnINLPDELETEITEMSTRUCT
00007fff`8c1ef770  00007fff`8c196080 USER32!_fnINLPDRAWITEMSTRUCT
00007fff`8c1ef778  00007fff`8c164b60 USER32!_fnINLPHELPINFOSTRUCT
00007fff`8c1ef780  00007fff`8c164b60 USER32!_fnINLPHELPINFOSTRUCT
00007fff`8c1ef788  00007fff`8c1dab00 USER32!_fnINLPMDICREATESTRUCT
```

Quelques-uns des attributs du bloc PEB existent en tant que duplicata de variables établies parmi la sphère système ou dérivent s'il en est fait mention expressément de valeurs caractéristiques installées au niveau du fichier image.

Tableau 8.4. Valeurs initiales des champs du PEB

Champ	Valeur initiale
CriticalSectionTimeout	Variable noyau *MmCriticalSectionTimeout*
HeapDeCommitFreeBlockThreshold	Variable noyau *MmHeapDeCommitFreeBlockThreshold*
HeapDeCommitTotalFreeThreshold	Variable noyau *MmHeapDeCommitTotalFreeThreshold*
HeapSegmentCommit	Variable noyau *MmHeapSegmentCommit*
HeapSegmentReserve	Variable noyau *MmHeapSegmentReserve*
ImageBaseAddress	IMAGE_OPTIONAL_HEADER.ImageBase
MaximumNumberOfHeaps	(Taille d'une page mémoire - taille d'un bloc PEB) / 4
MinimumStackCommit	Variable noyau *MmMinimumStackCommitInBytes*
NtGlobalFlag	Variable noyau *NtGlobalFlag*
NumberOfProcessors	Variable noyau *KeNumberOfProcessors*
OSBuildNumber	(OptionalHeader.Win32VersionValue \>\> 16) & 0x3FFF ou variable noyau NtBuildNumber & 0x3FFF si aucune information sur le sujet n'est apportée dans le fichier image
OSMajorVersion	OptionalHeader.Win32VersionValue & 0xFF ou variable noyau NtMajorVersion si aucune information sur le sujet n'est apportée dans le fichier image
OSMinorVersion	(OptionalHeader.Win32VersionValue \>\> 8) & 0xFF ou variable noyau NtMinorVersion
OSPlatformId	(OptionalHeader.Win32VersionValue \>\> 30) ^ 0x2 ou 2 si aucune information sur le sujet n'est apportée dans le fichier image

RTL_USER_PROCESS_PARAMETERS.

```
lkd> dtx nt!_RTL_USER_PROCESS_PARAMETERS 0x0000020a`70581c00
(*((nt!_RTL_USER_PROCESS_PARAMETERS *)0x20a70581c00))                 [Type: _RTL_USER_PROCESS_PARAMETERS]
    [+0x000] MaximumLength    : 0x730 [Type: unsigned long]
    [+0x004] Length           : 0x730 [Type: unsigned long]
    [+0x008] Flags            : 0x6001 [Type: unsigned long]
    [+0x00c] DebugFlags       : 0x0 [Type: unsigned long]
    [+0x010] ConsoleHandle    : 0x0 [Type: void *]
    [+0x018] ConsoleFlags     : 0x0 [Type: unsigned long]
    [+0x020] StandardInput    : 0x0 [Type: void *]
    [+0x028] StandardOutput   : 0x10001 [Type: void *]
    [+0x030] StandardError    : 0x0 [Type: void *]
    [+0x038] CurrentDirectory [Type: _CURDIR]
    [+0x050] DllPath          : "" [Type: _UNICODE_STRING]
    [+0x060] ImagePathName    : "C:\Windows\system32\notepad.exe" [Type: _UNICODE_STRING]
    [+0x070] CommandLine      : ""C:\Windows\system32\notepad.exe" " [Type: _UNICODE_STRING]
    [+0x080] Environment      : 0x20a70580fe0 [Type: void *]
    [+0x088] StartingX        : 0x0 [Type: unsigned long]
    [+0x08c] StartingY        : 0x0 [Type: unsigned long]
    [+0x090] CountX           : 0x0 [Type: unsigned long]
```

```
   [+0x094] CountY           : 0x0 [Type: unsigned long]
   [+0x098] CountCharsX      : 0x0 [Type: unsigned long]
   [+0x09c] CountCharsY      : 0x0 [Type: unsigned long]
   [+0x0a0] FillAttribute    : 0x0 [Type: unsigned long]
   [+0x0a4] WindowFlags      : 0x401 [Type: unsigned long]
   [+0x0a8] ShowWindowFlags  : 0x1 [Type: unsigned long]
   [+0x0b0] WindowTitle      : "C:\Windows\system32\notepad.exe" [Type: _UNICODE_STRING]
   [+0x0c0] DesktopInfo      : "Winsta0\Default" [Type: _UNICODE_STRING]
   [+0x0d0] ShellInfo        : "" [Type: _UNICODE_STRING]
   [+0x0e0] RuntimeData      : "" [Type: _UNICODE_STRING]
   [+0x0f0] CurrentDirectores [Type: _RTL_DRIVE_LETTER_CURDIR [32]]
   [+0x3f0] EnvironmentSize  : 0xc16 [Type: unsigned __int64]
   [+0x3f8] EnvironmentVersion : 0x3 [Type: unsigned __int64]
   [+0x400] PackageDependencyData : 0x0 [Type: void *]
   [+0x408] ProcessGroupId   : 0x280 [Type: unsigned long]
   [+0x40c] LoaderThreads    : 0x0 [Type: unsigned long]
   [+0x410] RedirectionDllName : "" [Type: _UNICODE_STRING]
   [+0x420] HeapPartitionName : "" [Type: _UNICODE_STRING]
   [+0x430] DefaultThreadpoolCpuSetMasks : 0x0 [Type: unsigned __int64 *]
   [+0x438] DefaultThreadpoolCpuSetMaskCount : 0x0 [Type: unsigned long]
```

- **Ligne de commande (CommandLine)** Arguments de ligne de commande qui ont accompagnés le processus lors de sa création.

- **Informations de bureau (DesktopInfo)** Station de travail et espace de bureau virtuel dans lesquels s'exécute le processus. Voir attribut lpDesktop de la structure STARTUPINFO.

- **Environment d'exécution (Environment)** Variables d'environnement dont est accompagné le processus. Voir fonctions CreateProcess (paramètre lpEnvironment) et RtlSetCurrentEnvironment.

- **Répertoire actuel (CurrentDirectory)** Répertoire par défaut pour les opérations liées aux fichiers et aux répertoires effectuées par le processus. Voir fonctions GetCurrentDirectory, SetCurrentDirectory ; fonction RTL RtlSetCurrentDirectory.

La structure PEB_LDR_DATA regroupe des informations concernant les modules (DLL) chargés par un processus. L'attribut Ldr stocké dans le bloc d'environnement du processus est un pointeur vers cette structure.

```
0:007> dt ntdll!_PEB_LDR_DATA
   +0x000 Length           : Uint4B
   +0x004 Initialized      : UChar
   +0x008 SsHandle         : Ptr64 Void
   +0x010 InLoadOrderModuleList : _LIST_ENTRY
   +0x020 InMemoryOrderModuleList : _LIST_ENTRY
   +0x030 InInitializationOrderModuleList : _LIST_ENTRY
   +0x040 EntryInProgress  : Ptr64 Void
   +0x048 ShutdownInProgress : UChar
   +0x050 ShutdownThreadId : Ptr64 Void
```

Chaque entrée s'apparentant à un module est de type `LDR_DATA_TABLE_ENTRY`, laquelle structure renferme une multitude d'informations.

```
typedef struct _LDR_DATA_TABLE_ENTRY {
  LIST_ENTRY InLoadOrderLinks;
  LIST_ENTRY InMemoryOrderLinks;
  union {
    LIST_ENTRY InInitializationOrderLinks;
    LIST_ENTRY InProgressLinks;
  };
  PVOID DllBase;
  PLDR_INIT_ROUTINE EntryPoint;
  ULONG SizeOfImage;
  UNICODE_STRING FullDllName;
  UNICODE_STRING BaseDllName;
```

```
  union {
    UCHAR FlagGroup[4];
    ULONG Flags;
    struct {
      ULONG PackagedBinary : 1;
      ULONG MarkedForRemoval : 1;
      ULONG ImageDll : 1;
      ULONG LoadNotificationsSent : 1;
      ULONG TelemetryEntryProcessed : 1;
      ULONG ProcessStaticImport : 1;
      ULONG InLegacyLists : 1;
      ULONG InIndexes : 1;
      ULONG ShimDll : 1;
      ULONG InExceptionTable : 1;
      ULONG ReservedFlags1 : 2;
      ULONG LoadInProgress : 1;
      ULONG LoadConfigProcessed : 1;
      ULONG EntryProcessed : 1;
      ULONG ProtectDelayLoad : 1;
      ULONG ReservedFlags3 : 2;
      ULONG DontCallForThreads : 1;
      ULONG ProcessAttachCalled : 1;
      ULONG ProcessAttachFailed : 1;
      ULONG CorDeferredValidate : 1;
      ULONG CorImage : 1;
      ULONG DontRelocate : 1;
      ULONG CorILOnly : 1;
      ULONG ChpeImage : 1;
      ULONG ChpeEmulatorImage : 1;
      ULONG ReservedFlags5 : 1;
      ULONG Redirected : 1;
      ULONG ReservedFlags6 : 2;
      ULONG CompatDatabaseProcessed : 1;
      };
  };
  USHORT ObsoleteLoadCount;
  USHORT TlsIndex;
  LIST_ENTRY HashLinks;
  ULONG TimeDateStamp;
  struct _ACTIVATION_CONTEXT *EntryPointActivationContext;
  PVOID Lock;
  PLDR_DDAG_NODE DdagNode;
  LIST_ENTRY NodeModuleLink;
  struct _LDRP_LOAD_CONTEXT *LoadContext;
  PVOID ParentDllBase;
  PVOID SwitchBackContext;
  RTL_BALANCED_NODE BaseAddressIndexNode;
  RTL_BALANCED_NODE MappingInfoIndexNode;
  ULONG_PTR OriginalBase;
  LARGE_INTEGER LoadTime;
  ULONG BaseNameHashValue;
  LDR_DLL_LOAD_REASON LoadReason;
  ULONG ImplicitPathOptions;
  ULONG ReferenceCount;
  ULONG DependentLoadFlags;
  UCHAR SigningLevel;
  ULONG CheckSum;
  PVOID ActivePatchImageBase;
  LDR_HOT_PATCH_STATE HotPatchState;
} LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;
```

- **Nom de module (BaseDllName)** Nom de fichier du module sans son chemin d'accès complet. Par exemple, si le fichier chargé est `C:\Windows\System32\kernel32.dll`, le nom affiché alors sera `kernel32.dll`. Pareille information est essentiellement utile pour identifier un module indépendamment de son emplacement sur le disque. Voir routine MmLoadSystemImage.

- **Hachage du nom (BaseNameHashValue)** Valeur hachée du nom du module (`BaseDllName`), utilisée pour accélérer la recherche d'un module donné parmi tous ceux chargés en mémoire. Voir fonction RtlHashUnicodeString.
- **Adresse de base (DllBase)** Adresse dans l'espace d'adressage d'un processus à laquelle réside le module après avoir été chargé par le système.
- **Point d'entrée (EntryPoint)** Adresse de la première fonction exécutée après chargement du module, typiquement `DllMain` dans le cas d'une DLL.
- **Niveau de signature (SigningLevel)** Indique si un module a été signé numériquement et, si tel est le cas, quel type de signature a été utilisé.
- **Taille (SizeOfImage)** Taille du module une fois chargé en mémoire.
- **Horodatage (TimeDateStamp)** Date et heure à laquelle un module a été finalisé par l'éditeur de liens.

Accès au PEB

Une solution élégante et efficace permettant d'accéder au bloc d'environnement du processus en cours repose sur l'exploitation de la macro NtCurrentPeb,

```
#define NtCurrentPeb() (NtCurrentTeb()->ProcessEnvironmentBlock)
```

D'un point de vue technique, l'extraction de l'adresse du PEB à partir du TEB s'effectue sur le base de registres spécifiques à l'architecture du processeur. Ainsi, sur un système x86 (32 bits), une telle opération se concrétise comme suit.

```
mov eax, fs:[0x30]
```

Dans l'architecture x86-64 (64 bits), considérant que les responsabilités autrefois dévolues au registre FS ont été transférées à GS, la même information est obtenue à l'aide de l'instruction que voici.

```
mov rax, gs:[0x60]
```

Au sein d'un programme écrit en langage C, `NtCurrentPeb` se présente le plus souvent sous la forme d'une définition tirant parti d'intrinsèques spécifiques au compilateur.

```
#ifdef _M_X64
#define NtCurrentPeb() ((PEB*)__readgsqword(0x60))
#else
#define NtCurrentPeb() ((PEB*)__readfsdword(0x30))
#endif
```

Accéder au bloc d'environnement d'un processus tiers (autre que l'appelant) exige l'appel à la fonction NtQueryInformationProcess, accompagnée dans un tel contexte de la classe d'information `ProcessBasicInformation`. Par nature, l'adresse ainsi obtenue est valide seulement au sein du processus cible, rendant obligatoire l'utilisation de fonctions inter-processus (ReadProcessMemory / WriteProcessMemory) pour en extraire les données.

Bloc Win32Process

Hors le noyau et les couches supérieures du noyau (exécutif), une partie de la gestion des processus est confiée à un autre des composants fondamentaux du système : le sous-système d'environnement Windows (Csrss), lequel maintient une structure parallèle pour chaque processus exécutant un programme Windows. La partie mode noyau du sous-système Windows (Win32k.sys) crée une structure de donnée spéciale, appelée bloc Win32

(W32PROCESS), quand un processus a recours à ses services - la première fois qu'un thread appelle une fonction USER ou GDI implémentée en mode noyau.

Le tableau suivant liste les principaux services système impliquant le bloc Win32Process.

Tableau 8.5. services système associés au bloc Win32Process

Service	Description
PsGetProcessWin32Process	Retourne l'adresse du bloc Win32Process d'un processus
PsGetCurrentProcessWin32Process	Retourne l'adresse du bloc Win32Process du processus courant
PsSetProcessWin32Process	Définit le bloc Win32Process d'un processus

```
lkd> dtx win32k!_W32PROCESS 0xffffed2e`c0753620
(*((win32k!_W32PROCESS *)0xffffed2ec0753620))                 [Type: _W32PROCESS]
    [+0x000] Process          : 0xffffa704a9353340 [Type: _EPROCESS *]
    [+0x008] RefCount         : 0x1 [Type: unsigned long]
    [+0x00c] W32PF_Flags      : 0x1004c030 [Type: unsigned long]
    [+0x010] InputIdleEvent   : 0xffffffffffffffff [Type: _KEVENT *]
    [+0x018] StartCursorHideTime : 0xef51 [Type: unsigned long]
    [+0x020] NextStart        : 0xffffed2ec06de620 [Type: _W32PROCESS *]
    [+0x028] pDCAttrList      : 0x9c00c60 [Type: void *]
    [+0x030] pBrushAttrList   : 0x5670a20 [Type: void *]
    [+0x038] W32Pid           : 0xa9c [Type: unsigned long]
    [+0x03c] GDIHandleCount   : 603 [Type: long]
    [+0x040] GDIHandleCountPeak : 0x40b [Type: unsigned long]
    [+0x044] UserHandleCount  : 500 [Type: long]
    [+0x048] UserHandleCountPeak : 0x3ef [Type: unsigned long]
    [+0x050] GDIPushLock      [Type: _EX_PUSH_LOCK]
    [+0x058] GDIEngUserMemAllocTable [Type: _RTL_AVL_TABLE]
    [+0x0c0] GDIDcAttrFreeList [Type: _LIST_ENTRY]
    [+0x0d0] GDIBrushAttrFreeList [Type: _LIST_ENTRY]
    [+0x0e0] GDIW32PIDLockedBitmaps [Type: _LIST_ENTRY]
    [+0x0f0] hSecureGdiSharedHandleTable : 0x9a167a3c6b6f1617 [Type: void *]
    [+0x0f8] DxProcess        : 0xffffb98952fb0e30 [Type: void *]
    [+0x100] DCompositionProcess : 0xffffed2ec26ff610 [Type: void *]
    [+0x108] UMPDSandboxingEnabled : 0x0 [Type: unsigned long]
    [+0x110] pWakeReference   : 0x0 [Type: void *]
    [+0x118] defaultDpiContext : 0x22 [Type: unsigned long]
    [+0x11c] Dpi              : 0xc0 [Type: unsigned short]
    [+0x120] bChangedGdiGammaRamp : 0 [Type: int]
```

CSR_PROCESS

Le sous-système Windows adopte comme l'exécutif et le noyau sa propre perspective des processus, laquelle se traduit au moyen de structures CSR_PROCESS. En tant que telles, seules les applications Windows ont une structure CSR_PROCESS qui leur est associée. Le gestionnaire de session (Smss), par exemple, se présente sous la forme d'une application native et n'en a en l'occurence pas. Chaque session possédant sa propre instance du sous-système Windows, les structures CSR_PROCESS sont gérées par le processus Csrss hébergé au sein de chaque session individuelle.

Les processus apparentés au sous-système Windows étant protégés (voir plus loin dans ce chapitre pour plus d'informations sur les processus protégés), il est de ce fait impossible d'interagir avec eux par l'intermédiaire d'un débogueur en mode utilisateur (même avec des privilèges élevés ou de manière non invasive). La structure CSR_PROCESS faisant néanmoins partie des symboles rendues visibles par le module Csrss.

```
lkd> !process 0 0 Csrss.exe
PROCESS ffffa704a60f20c0
    SessionId: 0  Cid: 01d0    Peb: a7a3c39000  ParentCid: 01bc
    DirBase: 10659d002  ObjectTable: ffffb9894de86280  HandleCount: 364.
    Image: csrss.exe
```

```
PROCESS ffffa704a8ac70c0
    SessionId: 1  Cid: 0234    Peb: 43cd8d4000  ParentCid: 0214
    DirBase: 105e1e002  ObjectTable: ffffb9894dbcc7c0  HandleCount: 444.
    Image: csrss.exe

lkd> .process /p /r ffffa704a8ac70c0
Implicit process is now ffffa704`a8ac70c0
Loading User Symbols

lkd> dt csrss!_CSR_PROCESS
   +0x000 ClientId         : _CLIENT_ID
   +0x010 ListLink         : _LIST_ENTRY
   +0x020 ThreadList       : _LIST_ENTRY
   +0x030 NtSession        : Ptr64 _CSR_NT_SESSION
   +0x038 ClientPort       : Ptr64 Void
   +0x040 ClientViewBase   : Ptr64 Char
   +0x048 ClientViewBounds : Ptr64 Char
   +0x050 ProcessHandle    : Ptr64 Void
   +0x058 SequenceNumber   : Uint4B
   +0x05c Flags            : Uint4B
   +0x060 DebugFlags       : Uint4B
   +0x064 ReferenceCount   : Int4B
   +0x068 ProcessGroupId   : Uint4B
   +0x06c ProcessGroupSequence : Uint4B
   +0x070 LastMessageSequence : Uint4B
   +0x074 NumOutstandingMessages : Uint4B
   +0x078 ShutdownLevel    : Uint4B
   +0x07c ShutdownFlags    : Uint4B
   +0x080 Luid             : _LUID
   +0x088 ServerDllPerProcessData : [1] Ptr64 Void
```

- **Options d'arrêt (ShutdownFlags)** Aspects spécifiques du comportement du processus lors de l'arrêt ou du redémarrage du système, par exemple s'il faut réessayer de le terminer après qu'il ait échoué à se fermer proprement une première fois.

- **Niveau d'arrêt (ShutdownLevel)** Niveau de priorité dont le système fait usage en vue pour déterminer quand le processus souhaite se terminer par rapport aux autres en cas d'arrêt ou de redémarrage du système. Pour plus d'informations, voir fonction SetProcessShutdownParameters.

Classification des processus

À différents stades de son évolution, Windows a en vue de répondre aux besoins en cours introduit différentes sortes de processus :

- **Standard** Les processus standard s'appuient sur l'API Windows traditionnelle. Ils peuvent être exécutés sur différentes versions de Windows et ont un accès à un éventail de fonctionnalités relativement large. À noter, au chapitre des potentiels inconvénients, un certain manque de souplesse en termes de prise en charge des diverses tailles d'écrans et dispositifs disponibles sur le marché.

- **UWP** Comparativement aux processus Windows standards, les processus UWP sont conçus pour une meilleure adaptablité sur divers types de dispositifs Windows, incluant PC, tablettes, téléphones, consoles, et d'autres appareils. Ils sont en revanche soumis à des exigences de sécurité plus strictes, et ont un accès plus restreint aux fonctionnalités du système d'exploitation.

- **Protégé** Les processus protégés le sont contre la manipulation, y compris par des processus avec des privilèges élevés. Une telle approche est souvent utilisée dans le contexte de la protection des contenus multimédias (DRM) ou d'autres scénarios où la sécurité est cruciale.

- **Protégé léger** Étend les mécanismes de protection initiés avec les processus protégés en ajoutant des couches de sécurité supplémentaires.

- **Minimal** Ces processus, spécifiquement élaborés pour des scénarios où où une isolation totale et une exécution minimale sont requises, voient leur structure formelle réduite à sa plus simple expression : pas d'espace d'adressage mode utilisateur, ni d'association avec un fichier exécutable ou une bibliothèque spécifique.
- **Pico** Processus qui dépendent du sous-système Windows pour Linux (WSL).

Interfaces

La liste suivante énumère quels aspects prennent en charge les fonctions, les services et les routines Windows définies pour les processus.

- *Créer un nouveau processus et un nouveau thread* Fonctions CreateProcess et CreateProcessAsUser.
- *Mettre fin à un processus* Fonctions *ExitProcess* et *TerminateProcess* ; service système NtTerminateProcess.
- *Vider le cache d'instruction d'un processus* Fonction *FlushInstructionCache* ; service système *NtFlushInstructionCache*.
- *Obtenir un pseudo handle pour le processus courant* Fonction *GetCurrentProcess*.
- *Obtenir l'ID du processus courant* Fonction *GetCurrentProcessId*.
- *Obtenir l'ID d'un processus* Fonction *GetProcessId* ; service système *NtQueryInformationProcess* ; routine noyau PsGetProcessId.
- *Obtenir la liste des variables de l'environnement du processus courant* Fonction *GetEnvironmentStrings*.
- *Obtenir le code de fin d'un processus* Fonction *GetExitCodeProcess* ; service système *NtQueryInformationProcess* ; routine noyau *PsGetProcessExitStatus*.
- *Voir la version de Windows sur laquelle un processus est censé être exécuté* Fonction *GetProcessVersion* ; services système *NtQueryInformationProcess*.
- *Récupérer une référence (handle) sur un processus* Fonction *OpenProcess* ; service système *NtOpenProcess* ; routines noyau *ObOpenObjectByName* et *ObOpenObjectByPointer*.
- *Consulter ou modifier les paramètres de stratégie DEP définis pour un processus* Fonctions *GetProcessDEPPolicy* et *GetProcessDEPPolicy* ; services système *NtQueryInformationProcess* et *NtSetInformationProcess*.
- *Récupérer le nom complet de l'image exécutable pour un processus* Fonction QueryFullProcessImageName ; service système *NtQueryInformationProcess*.
- *Obtenir des informations génériques temporelles liés à l'exécution d'un processus* Fonction *GetProcessTimes* ; service système NtQueryInformationProcess.

Variables noyau

La tableau qui suit énumère quelques-unes des variables globales système ayant un rôle prépondérant parmi les stratégies de gestion des processus.

Tableau 8.6. Variables noyau associés aux processus

Variable	Description
PsActiveProcessHead	Tête de liste des processus actifs
PspActiveProcessMutex	Protège l'accès à la liste des processus actifs

Variable	Description
PsIdleProcess	Pointeur vers le bloc du processus inactif
PsInitialSystemProcess	Pointeur vers le bloc du processus système initial
PspCreateProcessNotifyRoutine	Tableau de pointeurs vers des routines à appeler lors de la création des processus
PspCreateProcessNotifyRoutineCount	Nombre de routines déclarées de notification de processus
PspInitialSystemProcessHandle	Handle du processus système initial
PspLoadImageNotifyRoutine	Tableau de pointeurs vers des routines à appeler lors du chargement d'image
PspLoadImageNotifyRoutineCount	Nombre de routines déclarées de notification de chargement d'image
PspCidTable	Table de handles pour ID client de processus et de thread

Visualisation des processus actifs

L'extension !process du débogueur noyau permet l'affichage d'informations concernant soit un processus spécifié, soit l'ensemble des processus. La sortie suivante illustre le résultat de la commande !process 0 0.

```
lkd> !process 0 0
**** NT ACTIVE PROCESS DUMP ****
PROCESS ffff918615087440
    SessionId: none  Cid: 0004    Peb: 00000000  ParentCid: 0000
    DirBase: 001ad002  ObjectTable: ffffa38044403240  HandleCount: 2815.
    Image: System

PROCESS ffff918615176040
    SessionId: none  Cid: 0044    Peb: 00000000  ParentCid: 0004
    DirBase: 0bd80002  ObjectTable: ffffa38044424f00  HandleCount:   0.
    Image: Registry

PROCESS ffff9186165e4580
    SessionId: none  Cid: 0204    Peb: 8db3dd5000  ParentCid: 0004
    DirBase: 1b35f002  ObjectTable: ffffa3804497c9c0  HandleCount:  52.
    Image: smss.exe

PROCESS ffff918616576080
    SessionId: 0  Cid: 0270    Peb: 23946f7000  ParentCid: 0268
    DirBase: 10580002  ObjectTable: ffffa38044f582c0  HandleCount: 481.
    Image: csrss.exe
.
.
.
```

Examen des processus via le Gestionnaire des tâches

Pour obtenir une liste des processus en cours sur la station de travail, ainsi que différents chiffres significatifs de leur activité, ouvrez le Gestionnaire des tâches puis sélectionnez l'onglet détails.

- *Autres opérations d'E/S* Nombre d'opérations d'E/S qui ne sont ni des lectures ni des écritures (fonctions de contrôle, par exemple) générées par le processus depuis son démarrage.
- *Charge dédiée* Quantité de mémoire virtuelle réservée par le système d'exploitation pour le compte du processus.
- *Description* Brève description textuelle aidant l'utilisateur à mieux comprendre à quoi correspond un processus ou éventuellement la fonction qu'il remplit.
- *Écritures d'E/S* Nombre d'opérations d'E/S écriture générées par le processus depuis son démarrage.

- *Élevé* Spécifie si le processus est exécuté avec élévation de privilèges ou non.
- *Erreurs de page* Nombre de défauts de page générés par le processus depuis son démarrage.
- *Handles* Nombre de handles ouverts par le processus.
- *ID de session* Identifiant unique de la session dans laquelle le processus est exécuté.
- *Lectures d'E/S* Nombre d'opérations d'E/S lecture générées par le processus depuis son démarrage.
- *Ligne de commande* Ligne de commande complète spécifiée pour la création du processus.
- *Mémoire (plage de travail privé)* Quantité de mémoire physique utilisée par le processus et qui ne peut être employée par d'autres processus.
- *Mémoire (plage de travail partagée)* Quantité de mémoire physique utilisée par le processus et qui peut être partagée avec d'autres processus.
- *Nom de l'image* Nom de l'image exécutée dans le processus.
- *Nom du chemin d'accès de l'image* Chemin d'accès du fichier exécutable dont le processus résulte.
- *Nom d'utilisateur* Nom de l'entité qui a donné lieu au processus. Il peut s'agir d'un nom se rapportant à un compte d'utilisateur (utilisateur interactif ou un utilisateur d'une autre session ouverte en parallèle), à l'un des comptes système (système local, service local, service réseau) ou au système d'exploitation lui-même.
- *Objets GDI* Nombre d'objets GDI (Graphics Device Interface) que le processus utilise.
- *Objets USER* Nombre d'objets Gestionnaire de fenêtrage (fenêtres, menus, curseurs, dispositions du clavier, moniteurs, etc.) utilisés par le processus.
- *Octets de lecture d'E/S* Nombre total d'octets lus par le processus lors d'opérations d'E/S.
- *Octets d'écriture d'E/S* Nombre total d'octets écrits par le processus lors d'opérations d'E/S.
- *Octets d'autres opérations d'E/S* Nombre total d'octets lus lus par le processus lors d'opérations d'E/S qui ne sont pas des opérations de lecture/écriture.
- *PID* Identifiant unique du processus.
- *Prévention de l'exécution des données* Spécifie si la prévention de l'exécution des données est activée ou désactivée pour le processus.
- *Processeur* Pourcentage d'utilisation du processeur par le processus, tous coeurs confondus.
- *Réserve paginée* Quantité de mémoire du noyau paginable allouée par le système d'exploitation, noyau, les pilotes ou tout autre code système pour le compte du processus.
- *Plage de travail (mémoire)* Quantité de mémoire physique utilisée par le processus.
- *Plage de travail maximale (mémoire)* Quantité maximale de mémoire physique utilisable par le processus.
- *Plateforme* Plateforme (32 bits ou 64 bits) sur laquelle le processus est en cours d'exécution.
- *Rés. non pag.* Quantité de mémoire du noyau paginable allouée par le système d'exploitation, les pilotes ou tout autre code système pour le compte du processus.
- *Threads* Nombre de threads actifs hébergés par le processus.

- *Temps processeur* Temps processeur total, en secondes, utilisé par un processus depuis son démarrage.

Les actions envisageables depuis le menu, le menu contextuel ou les boutons sont les suivants :

- *Ouvrir l'emplacement du fichier* Fait apparaître une fenêtre de l'Explorateur Windows s'ouvrant sur le répertoire parent du fichier d'application apparenté au processus choisi.
- *Fin de tâche* Met fin au processus sélectionné.
- *Terminer l'arborescence de processus* Met fin au processus sélectionné et à ceux qui en dépendent.
- *Déboguer* Fait passer le processus sous le contrôle d'un débogueur.
- *Créer un fichier de vidage* Crée un fichier qui contient des informations de débogage pour l'application.
- *Définir la priorité* Modifie la priorité d'exécution globale des threads appartenant au processus.
- *Définir l'affinité* Permet de privilégier l'exécution du processus sur un ou plusieurs processeurs. Par défaut, l'affinité est définie de telle sorte à utiliser tous les processeurs.
- *Propriétés* Affiche la boîte de dialogue Propriétés, qui contient des informations de base à propos d'un fichier, y compris sa taille, son nom et son type, pour le fichier d'application apparenté au processus.
- *Accéder aux services* Affiche l'onglet Services et éventuellement le service associé au processus.

Flux de CreateProcess

La liste qui vient énumère les grandes manœuvres au centre de la création des processus Windows. Les opérations effectués à chaque étape, de moindre envergure mais en beaucoup plus grand nombre, seront détaillées dans les sections qui suivent.

1. Conversion et validation des paramètres transmis par l'appelant.
2. Ouverture du fichier image (.exe) à exécuter dans le processus.
3. Création de l'objet processus de l'exécutif (EPROCESS)
4. Création du thread initial et des supports épaulant son exécution : pile, contexte, objet thread de l'exécutif (ETHREAD), etc.
5. Notification au sous-système Windows de l'existence d'un nouveau processus, et suite à cela, préparation à la configuration et à l'exécution de celui-ci et du nouveau processus qu'il incorpore.
6. Démarrage de l'exécution du thread initial (hormis si l'appelant a spécifié le flag CREATE_SUSPENDED).
7. Fin de l'initialisation de l'espace d'adressage (par exemple, chargement des DLL requises) et commencement de l'exécution du programme.

Attributs de CreateProcess

L'appel à la fonction CreateProcess s'accompagne des attributs que voici.

- Le nom via lequel se présente l'application et la ligne de commande dont elle émane.
- Différents attributs axés sur la manière dont les autres processus peuvent interagir avec lui (contrôle d'accès).
- Une valeur booléenne déterminant si les handles du processus appelant, qui pourraient potentiellement être hérités, doivent effectivement l'être dans le nouveau processus.

- Diverses variables d'environnement.
- Un repertoire à partir duquel les opérations d'E/S sont prises en compte par défaut : lecture ou d'écriture de fichiers, chargement de DLL, etc.
- Quelques options établies en vue de personnaliser le lancement de l'application : titre, position, taille de la fenêtre associée, etc.
- Une structure qui, une fois passée l'étape proprement dite de création du processus (autrement dit au retour de la fonction), regroupe un petit nombre d'informations le concernant : ID unique, celui de son premier thread, références (handles).
- Divers indicateurs qui affectent la création de processus : démarrage en mode suspendu, transfert de contrôle à un débogueur, gestion des erreurs, etc.

Hiérarchie de processus

Hormis de rares exceptions (sur lesquelles nous reviendrons plus loin), chaque processus Windows n'est doté d'une existence autonome que par le biais d'une autre entité de même nature. (Un processus est autrement dit toujours engendré par un autre.) Le processus à l'origine d'un autre processus est appelé le processus père, tandis que le nouveau processus est appelé le processus fils. Le processus père peut créer d'autres processus et avoir ainsi, à un instant donné, plusieurs fils en cours d'exécution. Les processus fils sont à leur tour capables de donner naissance à un ou à plusieurs processus distincts. Une telle façon de procéder conduit à une arborescence, imaginée sous forme d'un arbre de descendance servant à schématiser chaque processus unitaire et à mieux mettre en avant les liens de filiations entre plusieurs générations de processus.

L'utilitaire PsList montre lorsque sa sollicitation est accompagnée du commutateur /t l'arborescence des processus.

```
C:\>pslist.exe /t

pslist v1.3 - Sysinternals PsList
Copyright (C) 2000-2012 Mark Russinovich
Sysinternals - www.sysinternals.com

Process information for HP:

Name                             Pid Pri Thd  Hnd      VM      WS    Priv
Idle                               0   0   8    0      64       4       0
  System                           4   8 197 1691   60604   16764     424
    smss                         388  11   2   49 4194303     336     356
csrss                            540  13  12  479 4194303    1932    1472
wininit                          632  13   1   86 4194303     860     924
  services                       708   9   5  435 4194303    5260    3704
    svchost                      104   8  38 1973 4194303   44596   25772
      sihost                    4220   8  12  499 4194303   16804    7860
      taskhostw                 4260   8   9  344 4194303   11288    7996
      taskhostw                 5284   8   4  301 4194303    8688    7468
      taskeng                   7184   8   9  123 4194303    6864    1560
      .
      .
      .
```

La liste indente chaque processus afin de montrer sa relation parent/enfant. Les processus dont les parents n'existent plus sont cadrés à gauche (c'est le cas ici de csrss et wininit). À la différence d'autres systèmes d'exploitation, dont Unix et ses dérivés, Windows ne gère pas de lien au-delà de l'ID du processus parent. Pour mieux vous rendre compte de cet aspect, effectuez les étapes suivantes.

1. Ouvrez une fenêtre d'invite de commandes et saisissez la commande start cmd, ce qui a pour effet de démarrer une seconde invite de commandes.

2. Dans la fenêtre d'invite de commandes nouvellement apparue, saisissez notepad.exe de sorte à donner le départ à une instance de l'application Bloc-notes.

3. La fenêtre associée au Bloc-notes devrait normalement à ce stade être celle active. Revenez dans la seconde invite de commandes et saisissez exit. Remarquez que cela n'a aucune incidence sur le Bloc-notes.

4. Ouvrez le gestionnaire des tâches. Cliquez sur l'onglet Détails et répérez parmi les processus listés celui sous-jacent à la fenêtre d'invite de commandes Windows, Cmd.exe.

5. Cliquez avec le bouton droit de la souris sur le processus Cmd.exe, puis faites Terminer l'arborescence du processus.

La première fenêtre d'invite de commandes disparait, tandis que la fenêtre Bloc-notes reste présente ; elle était le petit fils du processus Interpréteur de commandes Windows auquel vous venez de mettre fin. Du fait que le processus intermédiaire (le parent du Bloc-notes) était déjà arrêté, il n'y avait plus de lien entre le processus Cmd initial et le processus Bloc-notes.

Le champ InheritedFromUniqueProcessId de la structure EPROCESS spécifie le PID du processus père d'un processus donné.

```
lkd> !process 0 0 notepad.exe
PROCESS fffffa8045158940
    SessionId: 1  Cid: 0100    Peb: 7f7f0708000  ParentCid: 0730
    DirBase: 13eb75000  ObjectTable: fffff8a0071ee180  HandleCount: <Data Not Accessible>
    Image: notepad.exe

lkd> dt nt!_EPROCESS InheritedFromUniqueProcessId fffffa8045158940
   +0x3d0 InheritedFromUniqueProcessId : 0x00000000`00000730 Void

lkd> !process 730
Searching for Process with Cid == 730
PROCESS fffffa8046876940
    SessionId: 1  Cid: 0730    Peb: 7f74639c000  ParentCid: 0720
    DirBase: 11aecb000  ObjectTable: fffff8a001a79dc0  HandleCount: <Data Not Accessible>
    Image: explorer.exe
    .
    .
    .
```

Quelques processus, du fait de faire partie des premiers conçus sur le système, n'ont pas de père. Lorsque cela est le cas, Windows donne à l'attribut InheritedFromUniqueProcessId de chacun des objets sous-jacents à ces processus une valeur nulle, signe qu'il est impossible de remonter plus loin dans l'arbre des processus. À titre d'exemple, voici ce que donne l'examen de la variable InheritedFromUniqueProcessId dans le contexte du processus initial (PID 0) et dans le contexte du processus inactif (PID 4).

```
lkd> dt nt!_EPROCESS InheritedFromUniqueProcessId poi(nt!PsInitialSystemProcess)
   +0x3d0 InheritedFromUniqueProcessId : (null)

lkd> dt nt!_EPROCESS InheritedFromUniqueProcessId poi(nt!PsIdleProcess)
   +0x3d0 InheritedFromUniqueProcessId : (null)
```

Processus inactif du système

Le processus inactif du système sert de réceptacle à un certain nombre de threads (un par processeur) exécutés en mode noyau dont la seule fonction est d'occuper le processeur tandis qu'il ne peut l'être autrement. De ce fait, si d'aventure, et pour quelque raison que ce soit, un processeur vient à n'être réclamé par aucune entité exécutable tierce, Windows s'en remet pour lui au processus inactif.

Sur une machine multi processeur ou à multiples coeurs, il existe une seule instance du processus inactif, mais qui fait auquel cas fonctionner autant de threads que de coeurs logiques.

La commande !process du débogueur noyau permet de tracer un profil relativement complet du processus inactif. Voici la marche à suivre :

1. Commencez par vous renseigner sur le contenu de la variable globale système PsIdleProcess, laquelle pointe vers l'objet processus de l'exécutif sous-jacent au processus inactif.

```
lkd> ? poi(nt!PsIdleProcess)
Evaluate expression: -8779538304512 = fffff803`dabcf200
```

2. Spécifiez en paramètre de la commande !process l'adresse du bloc EPROCESS du processus inactif, telle qu'obtenue de la façon qui précède, ou via l'expression poi, comme dans ce qui suit.

```
lkd> !process poi(nt!PsIdleProcess)
PROCESS fffff803dabcf200
    SessionId: none  Cid: 0000    Peb: 00000000  ParentCid: 0000
    DirBase: 00187000  ObjectTable: fffff8a000003000  HandleCount: <Data Not Accessible>
    Image: Idle
    VadRoot fffffa8044a8aea0 Vads 1 Clone 0 Private 5. Modified 1412. Locked 0.
    DeviceMap 0000000000000000
    Token                             fffff8a0000054c0
    ElapsedTime                       00:00:00.000
    UserTime                          00:00:00.000
    KernelTime                        00:00:00.000
    QuotaPoolUsage[PagedPool]         0
    QuotaPoolUsage[NonPagedPool]      0
    Working Set Sizes (now,min,max)  (5, 50, 450) (20KB, 200KB, 1800KB)
    PeakWorkingSetSize                0
    VirtualSize                       0 Mb
    PeakVirtualSize                   0 Mb
    PageFaultCount                    0
    MemoryPriority                    BACKGROUND
    BasePriority                      0
    CommitCharge                      0
    .
    .
    .
```

Selon l'utilitaire de visualisation de processus, le processus inactif du système prend tel ou tel nom. À titre d'exemple, divers outils intégrés à Windows, dont le gestionnaire des tâches, appelle ce composant Processus inactif du système, tandis que d'autres ont tendance à le présenter autrement : Idle, Idle Process, voire - ce qui est largement trompeur - System Process. Windows, de son côté, emploi comme nom interne Idle, ce dont vous pouvez vous assurer en regardant l'attribut ImageFileName de la structure EPROCESS du processus inactif.

```
lkd> dt nt!_EPROCESS ImageFileName poi(nt!PsIdleProcess)
   +0x438 ImageFileName : [15]  "Idle"
```

Si vous étudiez profondément la question du bloc EPROCESS du processus inactif, par exemple au moyen de la commande dt (l'extension !process ne montrant qu'une partie des données), une chose qui ne devrait pas manquer de vous sauter aux yeux est que parmi toutes les variables mémorisées par cette structure, bon nombre des valeurs associées sont nulles ou paraissent incohérentes. Pour l'essentiel, ces particularités tiennent au fait que le processus inactif n'est pas, à proprement parler, un processus. Il n'existe pas de fichier image (.exe) à se trouver en corrélation avec lui (si ce n'est le fichier de programme abritant le noyau), sa supervision n'est l'oeuvre que de composants bas niveau du système d'exploitation et, plus important encore, ledit processus résulte de mécanismes en oeuvre dès l'amorçage, soit donc avant même que le gestionnaire de processus ou le gestionnaire d'objets aient lieu.

Processus protégés

Dans la perspective du modèle Windows de sécurité, tout processus dont les informations définies à ce niveau (jeton de sécurité) l'apparentent à des privilèges suffisamment élevés peut interagir sans restriction avec les autres. Un administrateur ou utilisateur qui possède le privilège SeDebug (nécessaire pour déboguer une

application) peut par exemple demander des droits d'accès presque complets aux systèmes, et de la sorte se voir conférer des caractéristiques lui permettant d'accéder à la mémoire d'autres processus, introduire des chemins alternatifs parmi le code exécuté, suspendre ou reprendre des threads, etc. Ce comportement, par ailleurs pleinement justifié et qui se trouve être à la base de nombreux utilitaires ayant besoin de ces pouvoirs pour fournir leur fonctionnalité (par exemple n'importe quel utilitaire un tant soit peu avancé de visualisation de processus, y compris le Gestionnaire de tâches), s'accommode mal avec les exigences requises en vue de protéger l'accès aux contenus soumis à des droits numériques. C'est dans ce contexte et pour répondre aux attentes de l'industrie des médias que Windows emploie une catégorie spéciale de processus, dits protégés (*Protected Process*).

S'ils cohabitent effectivement avec leurs homologues standards, les processus protégés ajoutent des contraintes significatives en ce qui concerne les droits d'accès que d'autres processus peuvent demander sur eux, cela indépendamment des privilèges (même administratifs) entrés en compte lors de l'exécution. La liste qui suit énumère les limites instituées en la matière. (À titre informatif, notez que la première et les deux dernières entrées de cette série se rapportent aux droits génériques, les autres sont spécifiques aux processus.)

Droit d'accès	**Fonction**
READ_CONTROL	Empêche que la liste de contrôle d'accès (ACL) du processus protégé soit lue.
PROCESS_ALL_ACCESS	Empêche l'accès complet et illimité vers un procédé protégé.
PROCESS_CREATE_PROCESS	Empêche la création d'un processus enfant d'un procédé protégé.
PROCESS_CREATE_THREAD	Empêche la création d'un thread tiers à partir d'un processus protégé.
PROCESS_DUP_HANDLE	Empêche la duplication de handles détenus par un processus protégé.
PROCESS_QUERY_INFORMATION	Empêche l'obtention d'informations concernant un processus protégé. Les processus normaux peuvent utiliser PROCESS_-QUERY_LIMITED_INFORMATION, lequel droit d'accès permet un accès restreint aux informations sur le processus.
PROCESS_SET_QUOTA	Empêche la redéfinition des quotas en lien avec un processus protégé.
PROCESS_SET_INFORMATION	Empêche la modification des paramètres liés à un processus protégé.
PROCESS_VM_OPERATION	Empêche la modification des paramètres régulant la mémoire virtuelle appartenant à un processus protégé, tels par exemple les options de protection de la mémoire (VirtualProtectEx).
PROCESS_VM_READ	Empêche l'accès en lecture de la mémoire appartenant à un processus protégé.
PROCESS_VM_WRITE	Empêche l'accès en écriture de la mémoire appartenant à un processus protégé.
WRITE_DAC	Empêche l'accès à la liste de contrôle d'accès discrétionnaire du processus protégé.
WRITE_OWNER	Empêche que le processus protégé subisse un changement de propriétaire du processus protégé

En plus de la limitation des droits d'accès accordés dont bénéficient les processus protégés, Windows empêche d'aborder la mémoire virtuelle de tels processus également par l'intermédiaire du chemin de code protégé (PMP, Protected Media Path), dont une partie du dispositif surveille quels composants abrite l'espace noyau. Dans l'éventualité où un pilote non signé par Microsoft (dans le cadre du programme qualité WHQL) est présent, le système avertit le processus protégé que son contexte n'est plus potentiellement sûr, charge alors à celui-ci de prendre les décisions qui s'imposent - le plus souvent, arrêter la lecture du contenu protégé et stopper son exécution.

Bien que Microsoft rende visible dans l'API Media Foundation tout le nécessaire afin que des concepteurs de logiciel soient en mesure d'élaborer des applications utilisatrices de processus protégés, le système d'exploitation ne permet à un processus d'être protégé que si le fichier image correspondant est doté d'une signature numérique spéciale.

Les composants mode noyau peuvent vérifier qu'un processus est protégé via la routine PsIsProtectedProcess, laquelle s'appuie sur les informations stockées au niveau de l'attribut Protection du bloc EPROCESS.

```
lkd> !process 0 0 services.exe
PROCESS ffffe18577fa7080
    SessionId: 0  Cid: 02d8    Peb: 87930e8000  ParentCid: 028c
    DirBase: 2af5d000  ObjectTable: ffff9686818ba040  HandleCount: <Data Not Accessible>
    Image: services.exe

lkd> dt nt!_EPROCESS Protection. ffffe18577fa7080
   +0x6c2 Protection  :
      +0x000 Level       : 0x61 'a'
      +0x000 Type        : 0y001
      +0x000 Audit       : 0y0
      +0x000 Signer      : 0y0110
```

Une autre alternative en ce qui concerne le repérage des processus protégés consiste à partir des caractéristiques propres à ce genre d'unités fonctionnelles, à savoir que bien qu'il soit par principe impossible d'accéder à un processus protégé et de consulter ses paramètres d'exécution, l'absence justement de ces informations dans le gestionnaire de tâches (ou tout autre utilitaire du même acabit) permet de reconnaître qu'un processus est protégé. Effectuez afin de vous en assurer les procédures que voici.

- Démarrez le Gestionnaire des tâches, puis cliquez sur l'onglet Détails.
- Effectuez un clic droit sur la ligne des colonnes et, dans le menu contextuel qui s'affiche, choisissez l'option Sélectionner des colonnes.
- Dans la boite de dialogue Sélectionner des colonnes, cochez l'option Ligne de commande.

Les processus en face desquels le gestionnaire des tâches s'est trouvé dans l'impossibilité de déterminer la ligne de commande sont précisément des processus protégés.

Processus protégés légers (PPL)

Né initialement de sorte à renforcer la sécurité autour de la gestion des droits numériques , le modèle dont émanent les processus protégés s'est par la suite ouvert - avec Windows 8.1 - à l'intégration d'autres services.

Dans la perspective PPL, les processus qui bénéficient d'un niveau de protection supérieur ont le droit d'accéder intégralement aux processus classés à un niveau moindre, tandis que l'inverse n'est pas autorisé. Cette structuration en niveaux de protection vise à renforcer la sécurité en limitant l'accès aux ressources sensibles aux processus moins dignes de confiance.

Tableau 8.7. Signataires PPL

Nom	Niveau	Description
WinSystem	7	Processus système et processus minimaux
WinTcb	6	Composants Windows critiques
Windows	5	composants Windows qui manipulent des données sensibles
Lsa	4	Lsass.exe s'il est configuré pour s'exécuter de manière protégée
Antimalware	3	Services et processus anti-malware, y compris tierces parties
CodeGen	2	Génération de code natif .NET (Ngen.exe)
Authenticode	1	Contenu DRM
None	0	Non valide, pas de protection associée.

Au niveau de la sphère noyau, WinSystem représente le signataire bénéficiant du plus haut niveau de protection envisageable. En ce qui concerne les processus en mode utilisateur, c'est auquel cas WinTCB (Windows Trusted

Computer Base) qui remporter la première place. Notez, à cet égard, que ledit niveau est concrétement le plus haut à pouvoir être associé à un processus protégé léger, et que les processus protégés l'emportent toujours sur leurs homologues légers.

Les processus protégés (légers ou non), eu égard à leur place dans les scénarios où la sécurité est une préoccupation majeure, sont limités dans leur capacité à charger des bibliothèques dynamiques (DLL). De telles mesures visent à empêcher que le processus protégé ne soit amené - à cause d'un bug ou remplacement d'un fichier sur disque - à charger une DLL non fiable. De ce fait, toutes les DLL chargées par les processus PPL doivent être correctement signées.

Sur le plan pratique, la création d'un processus protégé exige l'inclusion de l'attribut `CREATE_PROTECTED_PROCESS` lors de l'appel à `CreateProcess`, lequel indique au système d'exploitation que le processus nouvellement créé doit bénéficier d'une attention particulière.

Processus minimaux

Bien que de nombreux processus soient associés à l'exécution de code en mode utilisateur, tous ne partagent pas pour autant cette caractéristique. Quelques-uns, du fait de leurs orientations spécifiques, remplissent des fonctions distinctes, et n'ont en réalité aucun lien avec les mesures normalement employées pour exécuter des programmes. Dans un tel cas, Windows se remet alors à une forme spéciale de processus, dite minimale, dont découle notamment deux spécificités essentielles :

- Pas d'espace d'adressage en mode utilisateur, par extension aucune des structures véhiculées à l'ordinaire par ce biais : PEB, TEB et autres.
- Absence de lien entre le processus concerné et un quelconque fichier exécutable ou bibliothèque sur disque.

Dans la configuration standard, Windows a deux processus minimaux : System et Memory Compression.

Processus UWP

L'essor des dispositifs mobiles, tels que smartphones et tablettes, a par rapport aux méthodes traditionnelles associées aux ordinateurs de bureau, apporté des changements significatifs dans l'acquisition et la gestion des logiciels : centralisation du catalogue (*app store*), automatisation des procédures d'installations et de mises à jour, adaptation de l'interface utilisateur à plusieurs tailles d'écran, soumission à des contrôles de sécurité plus stricts, etc. Spécifiquement à Windows, la prise en compte de tous ces aspects émane d'un écosystème commun : UWP (Universal Windows Platform).

Les applications UWP cumulent plusieurs traits caractéristiques qui les différencient des processus standards, liés notamment à la manière dont celles-ci interagissent avec le système d'exploitation.

- Un processus UWP fonctionne dans un environnement clos (AppContainer) qui restreint ses actions et son accès aux ressources du système.
- La supervision du cycle de vie des processus UWP est le fait d'un composant dédié (PLM, Process Lifetime Manager), qui s'exécute dans le contexte du processus Explorer.exe.
- Tout package UWP comprend un ensemble de directives visant à spécifier ce à quoi l'application souhaite accéder, tells que la caméra ou l'emplacement de certains répertoires.
- Les applications UWP peuvent utiliser Windows Runtime, laquelle plateforme met en avant une architecture orientée objet, une interopérabilité entre langages, et une sécurité renforcée. L'API associée est utilisable depuis plusieurs langages de programmation, dont C, C#, Visual Basic, C et JavaScript.

Processus Pico

Windows, afin de permettre l'exécution de processus conçus pour d'autres systèmes d'exploitation (par exemple Linux), implémente un cadre de fonctionnement tiers, baptisé Pico, concrètement une interface minimaliste par

dessus certains services fondamentaux du noyau. (Notez, à titre informatif, que le terme Pico n'est en rien ici un acronyme, mais un sigle choisi pour évoquer l'extreme légèreté dudit mécanisme.)

Sur le plan pratique, un processus créé sous le modèle Pico ne repose sur aucun sous-système Windows classique, mais est géré par un composant spécialisé appelé fournisseur, chargé d'interpréter les appels système issus du processus, d'en créer et terminer ses threads, ou encore d'assurer la communication avec son environnement d'origine, par exemple un noyau Linux simulé dans le cas de WSL.

Bien qu'extérieurs aux conventions habituelles de Windows, les processus Pico bénéficient pleinement des services de base offerts par le noyau, y compris la planification, la mémoire virtuelle et les mécanismes de sécurité. Ils s'intègrent ainsi de manière transparente au système, tout en conservant un comportement fidèle à celui pour lequel ils ont à l'origine été pensés.

Identité de processus

Partie intégrante de la façon dont un processus apparaît au système d'exploitation et transparaît pour l'utilisateur, l'identité d'un processus se manifeste au travers différentes propriétés, parmi lesquelles l'ID unique du processus, l'ID du processus créateur, l'image exécutée, la station fenêtre où est exécuté le processus.

Identifiant de processus

Pour différencier sans ambiguïté les processus ayant lieu sur la station de travail, Windows octroie à chacun un numéro unique, appelé dans ce contexte identifiant (PID, *Process Identifier*), qui permet d'identifier formellement un processus parmi tous les autres, et ainsi pouvoir le désigner clairement auprès des différentes commandes, applications et interfaces programmatiques s'appliquant à un processus donné.

De la naissance d'un processus à son démantèlement, c'est le PID lui étant associé qui le désigne de manière non équivoque sur l'ensemble du système. Chaque nouveau processus reçoit comme valeur de PID (entier non signé de 32 bits) la dernière valeur attribuée lors de l'opération de création précédente augmentée de 4. (Toute valeur de PID est par conséquent divisible par ce chiffre.)

Chaque ID de processus a pour particularité d'être temporaire et unique. Temporaire, parce que les valeurs utilisées à cet égard peuvent être réutilisés. Le système d'exploitation se montre en outre garant de l'unicité des ID de processus, de sorte qu'aucuns ne se chevauchent. En plus de cela, les ID processus et les ID thread n'appartenant pas à la même catégorisation, ils ne peuvent jamais entrer en conflit.

En interne, du fait que les processus et les threads mode utilisateur sous Windows constituent la principale clientèle du sous-système d'environnement du même nom, les ID véhiculés par ces unités fonctionnelles sont parfois confondus au sein de la même appellation, à savoir ID client. C'est auquel cas le contexte seul qui permet de déduire s'il est question de l'ID d'un processus, de celui d'un thread, ou des deux à la fois.

Les applications Windows peuvent solliciter la valeur de PID d'un processus donné au moyen des fonctions GetProcessId et GetProcessIdOfThread. La premiere requiert en paramètre d'entrée un handle sur un processus, lequel doit véhiculer les droits d'accès PROCESS_QUERY_INFORMATION ou PROCESS_QUERY_LIMITED_INFORMATION, et repose sur la fonction native NtQueryInformationProcess. La seconde prend en entrée un handle sur un thread, qui doit être détendeur des accès THREAD_QUERY_INFORMATION ou THREAD_QUERY_LIMITED_INFORMATION, et s'appuie sur NtQueryInformationThread. La fonction GetCurrentProcessId retourne l'ID du processus appelant, et se base à cet effet sur un attribut idoine du bloc d'environnement du processus en cours d'execution, à savoir la valeur nommée UniqueProcess dans la sous-structure ClientId.

Les pilotes de périphérique ont à disposition les interfaces noyau PsGetProcessId, PsGetCurrentProcessId, PsGetCurrentThreadId et IoGetRequestorProcessId.

En interne, l'identifiant sert de clé d'accès à la table des processus.

Tableau 8.8. Routines concernant les identifiants de processus

PsLookupProcessByProcessId	Retourne un pointeur sur le bloc EPROCESS du processus sous-jacent au PID spécifié
PsGetCurrentProcessId	Retourne l'ID du processus auquel appartient le thread appelant

Visualisation des informations d'identité d'un processus

Pour visualiser les informations d'identité d'un processus pour lequel vous savez le savez le nom, utilisez la commande *!process* du débogueur noyau :

```
lkd> !process 0 0 notepad.exe
PROCESS fffffa800ee5b400
    SessionId: 1  Cid: 0e28    Peb: 7f7ab227000  ParentCid: 00e0
    DirBase: 20150000  ObjectTable: fffff8a001a02600  HandleCount: <Data Not Accessible>
    Image: notepad.exe
```

Vous pouvez pour un même but extraire l'attribut approprié du bloc EPROCESS :

```
lkd> dt nt!_EPROCESS UniqueProcessId fffffa800ee5b400
   +0x2e0 UniqueProcessId : 0x00000000`00000e28 Void
```

Si vous disposez de l'identifiant de processus mais pas du nom de l'image exécutée, vous pouvez encore utiliser la commande *!process*, cette fois en lui passant en paramètre l'ID du processus.

```
lkd> !process 0e28
Searching for Process with Cid == e28
PROCESS fffffa800ee5b400
    SessionId: 1  Cid: 0e28    Peb: 7f7ab227000  ParentCid: 00e0
    DirBase: 20150000  ObjectTable: fffff8a001a02600  HandleCount: <Data Not Accessible>
    Image: notepad.exe
    VadRoot fffffa800d158b60 Vads 67 Clone 0 Private 202. Modified 135. Locked 0.
    DeviceMap fffff8a0012708a0
    Token                             fffff8a002f08060
    ElapsedTime                       01:18:39.416
    UserTime                          00:00:00.000
    KernelTime                        00:00:00.000
    QuotaPoolUsage[PagedPool]         201928
    QuotaPoolUsage[NonPagedPool]      8448
    Working Set Sizes (now,min,max)  (2269, 50, 345) (9076KB, 200KB, 1380KB)
    PeakWorkingSetSize                2456
    VirtualSize                       98 Mb
    PeakVirtualSize                   110 Mb
    PageFaultCount                    2551
    MemoryPriority                    BACKGROUND
    BasePriority                      8
    CommitCharge                      344

        THREAD fffffa800e613b00  Cid 0e28.0e0c  Teb: 000007f7ab22e000 Win32Thread: fffff90100720b90 WAIT: (WrUserRequest)
 UserMode Non-Alertable
            fffffa800e1decd0  SynchronizationEvent
        Not impersonating
        DeviceMap                 fffff8a0012708a0
        Owning Process            fffffa800ee5b400       Image:         notepad.exe
        Attached Process          N/A            Image:         N/A
        Wait Start TickCount      136236         Ticks: 190174 (0:00:49:26.733)
        Context Switch Count      2403           IdealProcessor: 1
        UserTime                  00:00:00.031
        KernelTime                00:00:00.046
        Win32 Start Address 0x000007f7ab6e5a40
        Stack Init fffff8800e4a9c90 Current fffff8800e4a94b0
        Base fffff8800e4aa000 Limit fffff8800e4a4000 Call 0
        Priority 10 BasePriority 8 UnusualBoost 0 ForegroundBoost 2 IoPriority 2 PagePriority 5
        Kernel stack not resident.
```

Heures de processus

Tout programme en cours d'exécution par Windows a un commencement et une fin, ce dont le système prend acte en enregistrant pour chaque processus une heure de création, et s'il y a lieu une heure d'arrêt.

Visualisation des heures de processus

Pour visualiser les heures de création et de fin d'un processus, commencez par repérer un processus au moyen de la commande *!process*.

```
lkd> !process 0 0 notepad.exe
PROCESS fffffa800d4dc080
    SessionId: 1  Cid: 07a0    Peb: 7f6fee2f000  ParentCid: 0a38
    DirBase: 12ddc000  ObjectTable: fffff8a0087d0400  HandleCount: <Data Not Accessible>
    Image: notepad.exe
```

Ceci fait, analysez les attributs *CreateTime* et *ExitTime* du bloc du processus.

```
lkd> dt nt!_EPROCESS CreateTime ExitTime fffffa800d4dc080
   +0x2d0 CreateTime : _LARGE_INTEGER 0x01cf32f1`9c0821f7
   +0x610 ExitTime   : _LARGE_INTEGER 0x0
```

Utilisez la commande *.formats*, ou alternativement la commande *!filetime* pour convertir les données obtenues en un format humainement lisible.

```
lkd> .formats 0x01cf32f1`9c0821f7
Evaluate expression:
  Hex:     01cf32f1`9c0821f7
  Decimal: 130378927503319543
  Octal:   0007171457063402020767
  Binary:  00000001 11001111 00110010 11110001 10011100 00001000 00100001 11110111
  Chars:   ..2...!.
  Time:    Wed Feb 26 13:52:30.331 2014 (UTC + 1:00)
  Float:   low -4.50425e-022 high 7.61129e-038
  Double:  5.82338e-300

lkd> !filetime 0x01cf32f1`9c0821f7
 2/26/2014 13:52:30.331 (W. Europe Standard Time)
```

Quand un service de création de processus est appelé, survient dans le traitement la routine noyau *PspCreateProcess*, responsable de la création d'un objet exécutif idoine, qui lit l'heure actuelle du système (routine noyau *KeQuerySystemTime*) et affecte cette valeur à l'attribut *CreateTime* du bloc processus (EPROCESS CreateTime). L'heure d'arrêt d'un processus, elle, est fonction du moment où se termine le dernier thread lui appartenant.

Tableau 8.9. Eléments auxiliaires sous-jacents aux heures de processus

Elément	Type	Lieu	Description	Autres références
CreateTime	LARGE_INTEGER	EPROCESS	Heure de création de processus	PspCreateProcess, PsGetProcessCreateTimeQuadPart
ExitTime	LARGE_INTEGER	EPROCESS	Heure de fin du processus	PspExitThread, PsGetProcessExitTime

Image exécutée

Une étape initiale pour créer un processus est de repérer l'image Windows hôte des données et instructions principales de ce processus - lesquelles sont stockées sur disque dans un fichier exécutable, et de créer un objet section (appelé objet mappage de fichier dans l'API Windows) pour mapper l'image dans l'espace d'adressage du nouveau processus. À ce stade, c'est à dire une fois ouvert un fichier exécutable Windows valide et initialisées les données de l'objet section sous-jacent, le système d'exploitation crée l'objet processus de l'exécutif, et renseigne pour l'occasion le champ ImageFileName avec le nom adéquat.

Tableau 8.10. Eléments auxiliaires sous-jacents à l'image exécutée dans un processus

Elément	Type	Lieu	Description	Autres références
ImageFileName	UCHAR [16]	EPROCESS	Nom de l'image exécutée	PspCreateProcess, PsGetProcessImageFileName

Visualisation de l'image support d'un processus

Les informations renvoyées par la commande *!process* du débogueur noyau permettent de connaitre quel module est exécutée par un processus.

```
lkd> !process 0 0
**** NT ACTIVE PROCESS DUMP ****
PROCESS fffffa800cc64040
    SessionId: none  Cid: 0004    Peb: 00000000  ParentCid: 0000
    DirBase: 00187000  ObjectTable: fffff8a000003000  HandleCount: <Data Not Accessible>
    Image: System

PROCESS fffffa800efde940
    SessionId: none  Cid: 0124    Peb: 7f72573b000  ParentCid: 0004
    DirBase: 0894c000  ObjectTable: fffff8a00177d1c0  HandleCount: <Data Not Accessible>
    Image: smss.exe

PROCESS fffffa800e2fd4c0
    SessionId: 0  Cid: 017c    Peb: 7f734769000  ParentCid: 0170
    DirBase: 169fd000  ObjectTable: fffff8a001793500  HandleCount: <Data Not Accessible>
    Image: csrss.exe
```

Le nom après le mot clé Image est le nom du module qui possède le processus. Dans la dernière entrée de l'exemple précédent, le propriétaire est csrss.exe (processus du sous-système Windows). Dans la seconde, le propriétaire est smss.exe (processus gestionnaire de session d'utilisateur). Dans la première entrée, le propriétaire est le système d'exploitation lui-même.

Processus créateur

Visualisation des liens de parenté entre deux processus

La commande *!process* du débogueur noyau permet de connaitre l'ID du processus créateur :

```
lkd> !process 0 0 notepad.exe
PROCESS fffffa800ee5b400
    SessionId: 1  Cid: 0e28    Peb: 7f7ab227000  ParentCid: 00e0
    DirBase: 20150000  ObjectTable: fffff8a001a02600  HandleCount: <Data Not Accessible>
    Image: notepad.exe
```

Dans l'exemple qui précède, l'ID client du processus créateur de l'instance notepad.exe est 0xe0. Un même résultat aurait pu être obtenu comme suit :

```
lkd> dt nt!_EPROCESS InheritedFromUniqueProcessId fffffa800ee5b400
   +0x3d0 InheritedFromUniqueProcessId : 0x00000000`000000e0 Void
```

Une fois découvert ses données d'identification, vous pouvez en savoir plus le processus créateur une nouvelle fois avec la commande *!process* :

```
lkd> !process e0 0
Searching for Process with Cid == e0
PROCESS fffffa800ee02940
    SessionId: 1  Cid: 00e0    Peb: 7f74f928000  ParentCid: 0bd4
    DirBase: 0f0d8000  ObjectTable: 00000000  HandleCount:   0.
    Image: explorer.exe
```

A ce stade, nous savons que le processus notepad.exe a été créé par l'Explorateur Windows (explorer.exe).

Création de processus

Il y a création d'un processus Windows lorsque l'un déjà en place passe le relais à l'une ou l'autre des fonctions répertoriées dans le tableau ci-dessous.

Tableau 8.11. Fonctions Windows de création de processus

Fonction	Description
CreateProcess	Crée un nouveau processus et un nouveau thread en utilisant les informations de sécurité du processus appelant
CreateProcessAsUser	Crée un nouveau processus et un nouveau thread en utilisant les informations de sécurité du jeton spécifié
CreateProcessWithLogonW	Crée un nouveau processus et un nouveau thread en utilisant le nom et le mot de passe utilisateur spécifiés
CreateProcessWithTokenW	Crée un nouveau processus et un nouveau thread en utilisant l'identification de sécurité représentée par le jeton de sécurité spécifié, avec des options supplémentaires comme le chargement du profil utilisateur

Si toutes ces interfaces ont un but commun, à savoir créer un conteneur pour un ensemble de ressources employées lors de l'exécution de l'instance d'un programme, chacune se différencie au niveau des informations d'identification qu'elle emploie, donc en définitive du contexte de sécurité dans lequel elle opère.

La création d'un processus Windows se compose de plusieurs étapes, exécutées dans trois parties du système d'exploitation : la bibliothèque fondamentale du sous-système Windows (Kernel32), le processus serveur de l'environnement principal (Csrss) et l'exécutif Windows (gestionnaire de processus en tête).

Contrairement à d'autres systèmes d'exploitation où chaque processus est au départ la copie conforme d'un autre (via par exemple sous les dérivées d'Unix l'appel système fork), la création d'un processus Windows est réalisée ex nihilo.

Processus et synchronisation

Comme d'autres ressources système exposées en tant qu'objets, les processus Windows incorporent des sémantiques de synchronisation qui confèrent à chacun la possibilité d'être attendu. Un processus peut de la sorte se synchroniser avec un autre au moyen des fonctions usuelles définies à cet égard (WaitForSingleObject et WaitForMultipleObjects). Chaque processus durant tout son cycle de vie est à l'état non signalé, et passe à l'état signalé quand il prend fin. Ce passage à l'état signalé est alors irréversible, contrairement à d'autres primitives. Tous les threads qui attendaient ce processus sont à ce moment libérés de leur état d'attente. Les threads, qui peuvent également faire office de points de synchronisation, ont un comportement identique aux processus vis à vis de l'état vis-à-vis de l'état signalé/non signalé.

Gérer les processus

Dans le Gestionnaire des tâches, les champs de l'onglet Détails fournissent de nombreuses informations sur les processus actifs. Voici les champs affichés par défaut :

- *Nom de l'image* Nom du processus en cours d'exécution.
- *Nom d'utilisateur* Nom de l'utilisateur pour le compte duquel le processus s'exécute.
- *Processeur* Pourcentage d'utilisation du processeur par le processus.
- *Mémoire (jeu de travail privé)* Espace occupé dans la mémoire par le processus.
- *Description* Une description du processus.

Cliquer sur Affichage puis Sélectionner les colonnes de façon à voir apparaitre une boite de dialogue vous permettant d'ajouter des colonnes dans l'affichage Détails.

Handles d'instances

Chaque fichier exécutable chargé dans l'espace d'adressage d'un processus est identifié par un handle d'instance, dont la valeur correspond à l'adresse virtuelle courante où le module est chargé.

Une application peut obtenir le handle d'un module dans son espace d'adressage en passant en paramètre le nom du module à la fonction *GetModuleHandle*. Dans l'éventualité où NULL est donné comme nom, c'est alors l'adresse de base du processus appelant qui est retournée. Les handles renvoyées par GetModuleHandle ne peuvent pas être hérités ou utilisés par un autre processus.

L'appel à GetModuleHandle n'a aucune incidence sur la comptabilisation de ressources qu'effectue le gestionnaire d'objets, qui en la circonstance ne met pas à jour la valeur du compteur de références pour l'objet. Une version alternative de la fonction, GetModuleHandleEx, existe de façon à ajuster ce comportement, et en l'occurrence dynamiser le compteur de références. Il est de cette façon possible d'agir sur les facteurs qui conditionnent le système à conserver, ou au contraire démanteler, tel ou tel fichier représenté dans la mémoire du processus.

Pseudo-handles

Les pseudo-handles sont des constantes spéciales que le système considère en tout temps comme une référence sur un objet faisant partie de l'espace d'adressage du processus appelant. Cela peut comprendre le processus lui-même (GetCurrentProcess), un de ses threads (GetCurrentThread), le jeton d'accès assigné a l'application au moment de l'exécution (GetCurrentProcessToken), et d'autres. De par leur nature, les pseudo-handles ne font pas partie des caractéristiques héritables entre processus. Un processus peut par contre dupliquer un pseudo-handle à l'aide de la fonction DuplicateHandle et de la sorte en obtenir un valide dans le contexte d'autres processus.

Les fonctions qui traitent avec des pseudos-handles incluent GetCurrentProcess, GetCurrentThread, GetCurrentProcessToken, GetCurrentThreadToken, GetCurrentThreadEffectiveToken.

Notifications de niveau processus

Le gestionnaire de processus Windows implémente un mécanisme de notification permettant aux pilotes et autres codes système de se voir notifier d'événements tels la création et la suppression de processus et de threads, le chargement d'un image, ou encore la modification du registre. Très pratiques, ces fonctionnalités sont le plus souvent utilisé dans les logiciels de sécurité, lesquels ont besoin de surveiller étroitement l'ensemble de ces activités.

Notification de création et suppression des processus

Un thread peut s'interfacer avec le système de notification de création/destruction de processus via la routine *PsSetCreateProcessNotifyRoutine*, laquelle permet d'ajouter ou supprimer une entrée de la liste des routines à solliciter quand un processus prend vie ou meurt. PsSetCreateProcessNotifyRoutine prend en entrée deux paramètres ; le premier est le point d'entrée (adresse de base) d'une routine dans l'espace d'adressage du noyau, le second paramètre est un attribut qui contrôle la façon dont le système gère la dite routine, l'intègre ou l'élimine des notifications. (Autrement dit, la méthode par laquelle un thread attache une routine de notification est la même que celle utilisée pour son détachement, seul change le deuxième argument de *PsSetCreateProcessNotifyRoutine*.)

La fonction *PsSetCreateProcessNotifyRoutineEx*, version exécutive de *PsSetCreateProcessNotifyRoutine*, offre des fonctionnalités équivalentes à son homologue, mais est plus souple et offre un contrôle avant lancement du processus.

Le système d'exploitation exécute la routine de notification de création/suppression de processus au niveau IRQL PASSIVE_LEVEL en désactivant au préalable la livraison des APC mode noyau normaux. Quand il y a création de processus, la routine de notification s'exécute dans le contexte du thread qui a créé le nouveau processus. Lors d'une suppression de processus, elle s'exécute dans le contexte du dernier thread à prendre fin dans ce processus.

L'exécutif Windows gère les routines de rappel relatives aux notifications en stockant les adresses de pointeurs vers ces fonctions dans une zone mémoire de l'espace noyau, identifiée par une variable globale système nommée *PspCreateProcessNotifyRoutine*. Chaque fois que des services système de création ou suppression de processus entrent en lice, le noyau consulte cette table afin de pouvoir, grâce aux pointeurs qui y sont stockés, appeler les routines de rappel correspondantes.

Terminaison de processus

Chaque processus sous Windows se termine soit conformément à une circonstance ordinaire (arrêt normal), soit en réaction à une anomalie sérieuse, tellement en définitive qu'elle ne permet pas au programme d'aller de l'avant et poursuivre son exécution (arrêt forcé). Parmi les événements susceptibles de mettre fin à un processus (ou de précipiter la fin d'un processus), citons notamment :

- Un thread met explicitement un terme au processus auquel il appartient via la fonction ExitProcess
- Un thread force la terminaison du processus via l'appel à TerminateProcess
- Le dernier thread du processus se termine
- L'utilisateur arrête le système ou se déconnecte de la session.

Lorsqu'un processus prend fin via la fonction ExitProcess, le système déclenche la procédure de rappel de chaque librairie dynamique attachée en indiquant une valeur correspondant à une opération de détachement, permettant l'exécution de code additionnel avant que le processus ne soit véritablement supprimé. En revanche, dans le cas d'un processus dont la terminaison résulte de l'appel à TerminateProcess, les DLL ne sont pas avisées de la fin du processus, situation qui pourra quelquefois compromettre l'état de certaines données.

Quand un processus prend fin, il le fait en retournant un code de sortie indiquant la façon dont il s'est terminé. Par convention, ce code est utilisé de façon à mettre en évidence le bon déroulé (ou non) des opérations qui précédent. Un code de sortie à zéro indique généralement une exécution correcte, alors qu'un code différent de zéro, interpretable à ce moment comme un code d'erreur, souligne un dysfonctionnement. Dans ce dernier cas, la valeur renvoyée peut également apporter un complément d'information concernant la nature de l'erreur. Le code de sortie d'un processus Windows, lisible via la fonction GetExitCodeProcess, se concrétise sous l'une ou l'autre des formes suivantes :

- L'argument passé à la fonction ExitProcess ou à la fonction TerminateProcess
- La valeur retournée depuis la fonction principale (main, WinMain)
- La valeur d'exception pour une exception non traitée qui a entrainé la fin du processus

Lorsque le dernier thread d'un processus se termine, le noyau met à jour l'état de signal de l'objet exécutif idoine (avec un l'occurence un passage à l'état signalé), ce qui a pour conséquence de libérer tous les threads en attente de ce processus.

Alors que les handles qu'un processus a ouvert sont automatiquement supprimées quand il prend fin, les objets eux-mêmes continuent d'exister dans la mémoire du système, et cela jusqu'à ce que plus aucune référence ne les concernent. De ce fait, un objet peut subsister au delà du cycle de vie d'un processus, y compris d'un processus qui lui a donné naissance.

L'objet processus exécutif pour un processus en cours de terminaison devient signalé, satisfaisant les fils d'exécution possiblement en attente de la fin du processus. Même si le système entre dans une phase de

terminaison pour un processus, il ne met pas fin au processus enfant, ni ne génère de notifications pour que le cas soit spécifiquement traité.

Bureau

Dans la perspective Windows, la notion de Bureau (*Desktop*) fait référence à une surface d'affichage logique pour les objets d'interface utilisateur comme les fenêtres, les menus déroulants, les boîtes de dialogue, etc. Le bureau sert aussi de délimiteur pour les messages graphiques envoyés par les applications : ces dernières doivent être rattachées au même bureau afin de pouvoir communiquer entre elles via les messages graphiques.

Chaque objet bureau se présente en interne sous la forme d'une entité système sécurisable (et du reste sécurisée), contrôlée à ce titre par un descripteur de sécurité. Il est de fait possible pour un processus de créer un bureau avec des permissions d'accès limités. Lorsqu'un bureau est crée, le système l'associe automatiquement à la station-fenêtre où le processus appelant s'exécute.

Un processus est attaché à une station fenêtre et un thread à un bureau. Le système détermine le bureau à affecter au thread selon les règles suivantes : (1) par défaut, un thread hérite du bureau où le processus auquel il appartient s'exécute ; (2) un thread ayant revendiqué un bureau spécifique (via la fonction SetThreadDesktop) est rattaché au bureau demandé ; (3) dans l'éventualité où un thread n'a pas hérité de bureau ni n'en a sollicité un, le système procède de la manière suivante : si le nom d'un bureau a été spécifié pendant la création du processus (champ lpDesktop de la structure STARTUPINFO lors de l'appel à CreateProcess), le thread se connecte à ce bureau. Autrement, le thread se connecte au bureau par défaut de la station fenêtre à laquelle le processus est associé.

- CloseDesktop Ferme un handle ouvert sur un objet bureau.
- CreateDesktop Crée un objet bureau.
- EnumDesktops Enumère les bureaux associés à la station-fenêtre dans laquelle s'exécute le processus appelant.
- EnumDesktopWindows Enumère les fenêtres visibles de premier niveau associées à un bureau donné.
- GetThreadDesktop Retourne un handle pour l'objet bureau affecté au thread spécifié.
- GetUserObjetInformation Retourne des informations concernant une station-fenêtre ou un bureau.
- OpenInputDesktop Ouvre le bureau qui reçoit les entrées de l'utilisateur.
- SetThreadDesktop Assigne le bureau spécifié au thread appelant.
- SwitchDesktop Rend un bureau visible et l'active, permettant ainsi à ce bureau de recevoir des entrées de l'utilisateur.

L'isolation du processus dans un bureau spécifique permet d'empêcher l'interaction du processus avec d'autres applications (via des appels aux fonctions SendMessage ou PostMessage). De ce fait, la procédure d'interception des messages de fenêtre (WndProc) d'un processus s'exécutant sur un bureau particulier peut uniquement recevoir des messages destinés aux fenêtres créées dans le même bureau.

Trois bureaux sont créés par défaut dans dans la station fenêtre interactive : un bureau d'application (Default), un autre dévolu à un éventuel économiseur d'écran (ScreenSaver), et un autre encore, dit bureau sécurisé, dont le contrôle est en l'occurence géré par le processus d'ouverture de session (Winlogon).

Le bureau par défaut est créé lorsque Winlogon démarre le processus initial en tant qu'utilisateur connecté (Userinit.exe). À ce stade, le bureau par défaut devient actif et est alors utilisé pour interagir avec l'utilisateur.

Chaque fois qu'un économiseur d'écran sécurisé prend le relais, le système bascule automatiquement sur un bureau dédié, nommé ScreenSaver, dont la fonction première est de protéger les processus qu'abrite le bureau

par défaut contre les utilisateurs non autorisés. Les économiseurs d'écran non sécurisés s'exécutent sous le bureau Default.

Le bureau Winlogon est actif pendant la connexion d'un utilisateur. Le système bascule sur le bureau par défaut lorsque le shell indique qu'il est prêt à afficher quelque chose, ou après trente secondes, selon la première éventualité. Au cours de la session de l'utilisateur, le système bascule vers le bureau Winlogon lorsque l'utilisateur appuie sur la séquence de touches CTRL + ALT + SUPPR ou lorsque la boîte de dialogue Contrôle de compte d'utilisateur (UAC) est ouverte.

Un processus peut utiliser la fonction CreateDesktop ou CreateDesktopEx pour créer un nouveau bureau. Le nombre maximum de bureaux auquel le système peut donner lieu est limité par la taille du tas du bureau, soit 48 Mo. Il est possible d'augmenter le nombre de bureaux pouvant être créées en réduisant la taille du tas réservé à chaque bureau dans la station fenêtre interactive. Cette valeur est spécifiée dans la sous-chaine SharedSection située dans la valeur de registre HKLM\SYSTEM\CurrentControlSet\Control\Session Manager \SubSystems\Windows.

Le tableau suivant répertorie les droits d'accès spécifiques à l'objet bureau.

Tableau 8.12. Droits d'accès spécifiques à l'objet bureau

Droits d'accès	Constante	Description
DESKTOP_CREATEMENU	0x0004	Obligatoire pour créer un menu sur le bureau.
DESKTOP_CREATEWINDOW	0x0002	Obligatoire pour créer une fenêtre sur le bureau.
DESKTOP_ENUMERATE	0x0040	Obligatoire pour énumérer les bureaux de la station-fenêtre.
DESKTOP_HOOKCONTROL	0x0008	Obligatoire pour établir une procédure de hook
DESKTOP_JOURNALPLAYBACK	0x0020	Voir fonction JournalPlaybackProc.
DESKTOP_JOURNALRECORD	0x0010	Voir fonction JournalRecordProc.
DESKTOP_READOBJECTS	0x0001	-
DESKTOP_SWITCHDESKTOP	0x0100	Obligatoire pour permuter vers un bureau.- Voir fonction SwitchDesktop.
DESKTOP_WRITEOBJECTS	0x0080	-

Station-fenêtre

Windows offre différentes catégories d'objets permettant de gérer l'interface graphique utilisateur, parmi lesquels la station fenêtre et le bureau. La station est un objet conteneur qui, entre autres, gère le presse papier ainsi que les atoms globaux. Une station peut gérer un certain nombre de bureaux.

- CreateWindowStation Crée une station-fenêtre.
- GetProcessWindowStation Retourne un handle vers la station-fenêtre à laquelle appartient le processus appelant.
- OpenWindowStation Ouvre la station-fenêtre spécifiée.
- SetProcessWindowStation Associe la station-fenêtre spécifiée au processus appelant.
- GetUserObjetInformation et SetUserObjetInformation Retourne ou modifie, respectivement, certaines informations concernant une station-fenêtre ou un bureau.

Tableau 8.13. Droits d'accès spécifiques à l'objet station-fenêtre

Droits d'accès	Constante	Description
WINSTA_ACCESSCLIPBOARD	0x0004	Obligatoire pour utiliser le presse-papier.

WINSTA_ACCESSGLOBALATOMS	0x0020	-
WINSTA_ALL_ACCESS	0x037F	Tous les droits envisageables concernant une station fenêtre.
WINSTA_CREATEDESKTOP	0x0008	Obligatoire pour créer de nouveaux objets bureau.
WINSTA_ENUMDESKTOPS	0x0001	-
WINSTA_ENUMERATE	0x0100	Obligatoire pour énumèrer les stations fenêtre.
WINSTA_EXITWINDOWS	0x0040	-
WINSTA_READATTRIBUTES	0x0002	-
WINSTA_READSCREEN	0x0200	-
WINSTA_WRITEATTRIBUTES	0x0010	-

Enumération de processus

- EnumProcesses Retourne l'ID de chaque processus en cours d'exécution
- Process32First Retourne des informations concernant le premier processus d'une vue créée par CreateToolhelp32Snapshot

Sécurité des processus

Tableau 8.14. Droits d'accès concernant les processus

Constante	Valeur	Description
PROCESS_ALL_ACCESS	0x1F0FFF	Procure un accès complet et illimité au processus.
PROCESS_CREATE_PROCESS	0x0080	Requis pour créer un processus.
PROCESS_CREATE_THREAD	0x0002	Requis pour créer un thread.
PROCESS_DUP_HANDLE	0x0040	Requis pour dupliquer un handle.
PROCESS_QUERY_INFORMATION	0x0400	Requis pour obtenir certaines informations relatives à un processus, telles que sa classe de priorité.
PROCESS_SET_INFORMATION	0x0200	Requis pour définir certaines informations relatives à un processus, telles que sa classe de priorité.
PROCESS_SET_PORT	0x0800	Voir service NtSetInformationProcess (ProcessInformationClass = ProcessExceptionPort et ProcessDebugPort).
PROCESS_SET_QUOTA	0x0100	Requis pour définir certaines informations relatives aux quotas de processus.- Voir service NtSetInformationProcess (ProcessInformationClass = ProcessQuotaLimits).
PROCESS_SET_SESSIONID	0x0004	Voir service NtSetInformationProcess (ProcessInformationClass = ProcessSessionInformation).
PROCESS_SUSPEND_RESUME	0x0800	Requis pour suspendre ou reprendre l'exécution d'un processus.
PROCESS_TERMINATE	0x0001	Requis pour mettre fin à un processus.- Controlé par NtTerminateProcess.
PROCESS_VM_OPERATION	0x0008	Requis pour effectuer certaines opérations concernant l'espace d'adressage d'un processus.
PROCESS_VM_READ	0x0010	Requis pour lire la mémoire d'un processus.

Constante	Valeur	Description
PROCESS_VM_WRITE	0x0020	Requis pour écrire dans la mémoire d'un processus.

Service NtQueryInformationProcess

Tableau 8.15. ProcessInformationClass

Nom	Valeur	Type de données associé
ProcessBasicInformation	00	PROCESS_BASIC_INFORMATION, PROCESS_EXTENDED_BASIC_INFORMATION
ProcessQuotaLimits	01	QUOTA_LIMITS, QUOTA_LIMITS_EX
ProcessIoCounters	02	IO_COUNTERS
ProcessVmCounters	03	VM_COUNTERS, VM_COUNTERS_EX
ProcessTimes	04	KERNEL_USER_TIMES
ProcessBasePriority	05	KPRIORITY
ProcessRaisePriority	06	ULONG
ProcessDebugPort	07	HANDLE
ProcessExceptionPort	08	HANDLE
ProcessAccessToken	09	PROCESS_ACCESS_TOKEN
ProcessLdtInformation	10	PROCESS_LDT_INFORMATION
ProcessLdtSize	11	PROCESS_LDT_SIZE
ProcessDefaultHardErrorMode	12	ULONG
ProcessIoPortHandlers	13	-
ProcessPooledUsageAndLimits	14	POOLED_USAGE_AND_LIMITS
ProcessWorkingSetWatch	15	PROCESS_WS_WATCH_INFORMATION
ProcessUserModeIOPL	16	-
ProcessEnableAlignmentFaultFixup	17	BOOLEAN
ProcessPriorityClass	18	PROCESS_PRIORITY_CLASS
ProcessWx86Information	19	ULONG
ProcessHandleCount	20	ULONG
ProcessAffinityMask	21	KAFFINITY
ProcessPriorityBoost	22	ULONG
ProcessDeviceMap	23	PROCESS_DEVICEMAP_INFORMATION, PROCESS_DEVICEMAP_INFORMATION_EX
ProcessSessionInformation	24	PROCESS_SESSION_INFORMATION
ProcessForegroundInformation	25	PROCESS_FOREGROUND_BACKGROUND
ProcessWow64Information	26	ULONG
ProcessImageFileName	27	UNICODE_STRING
ProcessLUIDDeviceMapsEnabled	28	-
ProcessBreakOnTermination	29	-
ProcessDebugObjectHandle	30	-
ProcessDebugFlags	31	-
ProcessHandleTracing	32	-
ProcessIoPriority	33	-
ProcessExecuteFlags	34	-
ProcessTlsInformation	35	-
ProcessCookie	36	ULONG
ProcessImageInformation	37	SECTION_IMAGE_INFORMATION
ProcessCycleTime	38	PROCESS_CYCLE_TIME_INFORMATION
ProcessPagePriority	39	PAGE_PRIORITY_INFORMATION
ProcessInstrumentationCallback	40	-

Nom	Valeur	Type de données associé
ProcessThreadStackAllocation	41	-
ProcessWorkingSetWatchEx	42	PROCESS_WS_WATCH_INFORMATION_EX
ProcessImageFileNameWin32	43	-
ProcessImageFileMapping	44	-
ProcessAffinityUpdateMode	45	-
ProcessMemoryAllocationMode	46	-
ProcessGroupInformation	47	-
ProcessTokenVirtualizationEnabled	48	-
ProcessOwnerInformation	49	-
ProcessWindowInformation	50	-
ProcessHandleInformation	51	-
ProcessMitigationPolicy	52	PROCESS_MITIGATION_POLICY
ProcessDynamicFunctionTableInformation	53	-
ProcessHandleCheckingMode	54	-
ProcessKeepAliveCount	55	PROCESS_KEEPALIVE_COUNT_INFORMATION
ProcessRevokeFileHandles	56	PROCESS_REVOKE_FILE_HANDLES_INFORMATION
ProcessWorkingSetControl	57	-
ProcessHandleTable	58	-
ProcessCheckStackExtentsMode	59	-
ProcessCommandLineInformation	60	UNICODE_STRING
ProcessProtectionInformation	61	PS_PROTECTION
ProcessMemoryExhaustion	62	PROCESS_MEMORY_EXHAUSTION_INFO
ProcessFaultInformation	63	PROCESS_FAULT_INFORMATION
ProcessTelemetryIdInformation	64	-
ProcessCommitReleaseInformation	65	-
ProcessReserved1Information	66	-
ProcessReserved2Information	67	-
ProcessSubsystemProcess	68	-
ProcessInPrivate	70	-
ProcessRaiseUMExceptionOnInvalidHandleClose	71	-

```
typedef struct _PROCESS_BASIC_INFORMATION {
    NTSTATUS ExitStatus;
    PPEB PebBaseAddress;
    ULONG_PTR AffinityMask;
    KPRIORITY BasePriority;
    ULONG_PTR UniqueProcessId;
    ULONG_PTR InheritedFromUniqueProcessId;
} PROCESS_BASIC_INFORMATION;
```

- **Masque d'affinité (*AffinityMask*)** Voir fonction GetProcessAffinityMask.
- **Code de fin (*ExitStatus*)** Voir fonction GetExitCodeProcess.
- **ID client (*UniqueProcessId*)** Voir fonction GetProcessId.

Tableau 8.16. Valeurs initiales des champs de la structure PROCESS_BASIC_INFORMATION

Champ	Valeur initiale
AffinityMask	KPROCESS Affinity
BasePriority	KPROCESS BasePriority
ExitStatus	EPROCESS ExitStatus
InheritedFromUniqueProcessId	EPROCESS InheritedFromUniqueProcessId

Champ	Valeur initiale
PebBaseAddress	EPROCESS Peb
UniqueProcessId	EPROCESS UniqueProcessId

Tableau 8.17. Valeurs initiales des champs pour la classe ProcessImageFileName

Champ	Valeur initiale
ProcessInformation	EPROCESS SeAuditProcessCreationInfo.ImageFileName

Tableau 8.18. Valeurs initiales des champs pour la classe ProcessDefaultHardErrorMode

Champ	Valeur initiale
ProcessInformation	EPROCESS DefaultHardErrorProcessing

Tableau 8.19. Valeurs initiales des champs pour la classe ProcessIoCounters

Champ	Valeur initiale
ReadOperationCount	EPROCESS ReadOperationCount
WriteOperationCount	EPROCESS WriteOperationCount
OtherOperationCount	EPROCESS OtherOperationCount
ReadTransferCount	EPROCESS ReadTransferCount
WriteTransferCount	EPROCESS WriteTransferCount
OtherTransferCount	EPROCESS OtherTransferCount

PROCESS_PRIORITY_CLASS.

```
typedef struct _PROCESS_PRIORITY_CLASS {
    BOOLEAN Foreground;
    UCHAR PriorityClass;
} PROCESS_PRIORITY_CLASS, *PPROCESS_PRIORITY_CLASS;
```

PROCESS_LDT_INFORMATION.

```
typedef struct _PROCESS_LDT_INFORMATION {
    ULONG Start;
    ULONG Length;
    LDT_ENTRY LdtEntries[1];
} PROCESS_LDT_INFORMATION, *PPROCESS_LDT_INFORMATION;
```

PROCESS_LDT_SIZE.

```
typedef struct _PROCESS_LDT_SIZE {
    ULONG Length;
} PROCESS_LDT_SIZE, *PPROCESS_LDT_SIZE;
```

KERNEL_USER_TIMES.

```
typedef struct _KERNEL_USER_TIMES {
    LARGE_INTEGER CreateTime;
    LARGE_INTEGER ExitTime;
    LARGE_INTEGER KernelTime;
    LARGE_INTEGER UserTime;
} KERNEL_USER_TIMES, *PKERNEL_USER_TIMES;
```

Tableau 8.20. Valeurs initiales des champs de la structure VM_COUNTERS_EX

Champ	Valeur initiale
PeakVirtualSize	EPROCESS PeakVirtualSize
VirtualSize	EPROCESS VirtualSize
PageFaultCount	EPROCESS Vm.PageFaultCount
PeakWorkingSetSize	EPROCESS Vm.PeakWorkingSetSize << PAGE_SHIFT
WorkingSetSize	EPROCESS Vm.WorkingSetSize << PAGE_SHIFT
QuotaPeakPagedPoolUsage	EPROCESS ProcessQuotaPeak[1]

Champ	Valeur initiale
QuotaPagedPoolUsage	EPROCESS ProcessQuotaUsage[1]
QuotaPeakNonPagedPoolUsage	EPROCESS ProcessQuotaPeak[0]
QuotaNonPagedPoolUsage	EPROCESS ProcessQuotaUsage[0]
PagefileUsage	EPROCESS ProcessQuotaUsage[2] << PAGE_SHIFT
PeakPagefileUsage	EPROCESS ProcessQuotaPeak[2] << PAGE_SHIFT
PrivateUsage	EPROCESS CommitCharge << PAGE_SHIFT

PROCESS_SESSION_INFORMATION.

```
typedef struct _PROCESS_SESSION_INFORMATION
{
    ULONG SessionId;
} PROCESS_SESSION_INFORMATION, *PPROCESS_SESSION_INFORMATION;
```

Tableau 8.21. Valeurs initiales des champs de la structure PROCESS_SESSION_INFORMATION

Champ	Valeur initiale
SessionId	EPROCESS Session.SessionId

Tableau 8.22. Valeurs initiales des champs pour la classe ProcessTimes

Champ	Valeur initiale
CreateTime	EPROCESS CreateTime
ExitTime	EPROCESS ExitTime
KernelTime	KPROCESS KernelTime, KTHREAD KernelTime
UserTime	KPROCESS UserTime, KTHREAD UserTime

POOLED_USAGE_AND_LIMITS.

```
typedef struct _POOLED_USAGE_AND_LIMITS {
    SIZE_T PeakPagedPoolUsage;
    SIZE_T PagedPoolUsage;
    SIZE_T PagedPoolLimit;
    SIZE_T PeakNonPagedPoolUsage;
    SIZE_T NonPagedPoolUsage;
    SIZE_T NonPagedPoolLimit;
    SIZE_T PeakPagefileUsage;
    SIZE_T PagefileUsage;
    SIZE_T PagefileLimit;
} POOLED_USAGE_AND_LIMITS, *PPOOLED_USAGE_AND_LIMITS;
```

PROCESS_FOREGROUND_BACKGROUND.

```
typedef struct _PROCESS_FOREGROUND_BACKGROUND {
    BOOLEAN Foreground;
} PROCESS_FOREGROUND_BACKGROUND, *PPROCESS_FOREGROUND_BACKGROUND;

typedef struct _QUOTA_LIMITS_EX {
  SIZE_T PagedPoolLimit;
  SIZE_T NonPagedPoolLimit;
  SIZE_T MinimumWorkingSetSize;
  SIZE_T MaximumWorkingSetSize;
  SIZE_T PagefileLimit;
  LARGE_INTEGER TimeLimit;
  SIZE_T WorkingSetLimit;
  SIZE_T Reserved2;
  SIZE_T Reserved3;
  SIZE_T Reserved4;
  ULONG Flags;
  RATE_QUOTA_LIMIT CpuRateLimit;
} QUOTA_LIMITS_EX, *PQUOTA_LIMITS_EX;
```

- **Maxima de working set (*MaximumWorkingSetSize*)** Voir fonction GetProcessWorkingSetSizeEx.
- **Minima de working set (*MinimumWorkingSetSize*)** Voir fonction GetProcessWorkingSetSizeEx.

Liste des processus actifs

Windows garde trace des processus existants au moyen d'une liste à chainage double, chaque processus pointant ainsi sur le processus qui le suit et celui qui le précède. Le système d'exploitation parcourt ladite liste à l'aide de la variable noyau PsActiveProcessHead, qui offre un pointeur vers la tête de la liste. Chaque processus a dans son bloc EPROCESS une structure `LIST_ENTRY` nommée ActiveProcessLinks qui renferme des pointeurs vers, respectivement, les processus suivant (champ Flink) et précédant (champ Blink).

L'exécutif Windows actualise la liste des processus actifs lors de chaque création (routine privée PspCreateProcess) et suppression (PspProcessDelete) de processus.

Variables d'environnement de processus

Dans la plupart des cas, les variables d'environnement d'un processus sont transmises par voie de filiation, du père au fils, et forment alors une combinaison de variables système (qui s'appliquent à tous les utilisateurs) et d'autres variables spécifiques à l'utilisateur.

Une application console est en mesure d'obtenir les variables d'environnement dont elle est accompagnée à l'aide du troisième paramètre formel issu de leur fonction principale.

```
int main(int argc, char *argv[], char *envp[]);
int wmain(int argc, wchar_t *argv[], wchar_t *env[]);
```

Parmi ces déclarations, `env` (comme `envp`) se rapporte à une liste de chaînes de caractères, où chaque chaîne représente une variable d'environnement.

Dans le cas d'une application graphique, l'examen de l'environnement implique la fonction `GetEnvironmentStrings`, laquelle retourne un bloc de mémoire contenant les variables dont dépend le processus en cours.

Répertoires de travail d'un processus

Chaque processus sous Windows a un répertoire qui lui est associé, dit alors répertoire de travail ou répertoire actuel. Emplacement, en premier lieu, à partir duquel le processus a entamé son exécution, il s'agit de façon conjointe du répertoire par défaut pour toutes les opérations de lecture ou d'écriture de fichiers effectuées par ce processus, à moins que des chemins de fichiers absolus ne soient spécifiés.

Les fonctions `GetCurrentDirectory` et `SetCurrentDirectory` permettent, respectivement, d'obtenir le chemin complet du répertoire de travail du processus appelant, et de changer le répertoire de travail du processus en spécifiant un nouveau chemin.

Dans les coulisses des threads

Après nous être intéressés aux coulisses des processus, il est désormais temps de porter le regard sur les rouages et les mécanismes en oeuvre pour la gestion des threads. Sauf mention explicite du contraire, le contenu de ce segment s'applique indifféremment aux threads mode utilisateur et aux threads mode noyau.

Généralités concernant les threads

Chaque processus peut contenir un ou plusieurs threads. Windows emploie en la matière une correspondance un-vers-un où chaque thread au niveau utilisateur est en relation avec un thread noyau qui lui est associé. Un support pour différentes fonctionnalités du modèle plusieurs-vers-un est toutefois également de la partie, ce par l'intermédiaire des fibres.

Fonctions

La liste qui suit énumère quelques-uns des interfaces que Windows définit relativement aux threads. Ne sont pas données ici les fonctions concernant l'ordonnancement et les priorités de thread, lesquelles sont traitées plus loin à la section "Ordonnancement des threads".

- *CreateThread* Crée un nouveau thread
- *CreateRemoteThread* Crée un thread dans un autre processus
- *ExitThread* Termine normalement l'exécution d'un thread
- *TerminateThread* Met fin à un thread spécifié (comme ExitThread) mais sans exécuter de code de nettoyage
- *OpenThread* Ouvre un thread existant
- *GetCurrentThreadId* Retourne l'ID du thread courant
- *GetThreadId* Retourne l'ID du thread courant
- *GetExitCodeThread* Retourne le code de sortie d'un thread
- *Get/SetThreadContext* Retourne ou modifie les valeurs des registres processeur d'un thread

Structure de données

Au même titre qu'un processus Windows est dans les sphères supérieures du noyau une structure régie par des services définies au niveau de l'exécutif (EPROCESS), chaque thread est en interne du système représenté par un objet thread exécutif (ETHREAD). A cela s'ajoute une structure parallèle dans le processus serveur de sous-système d'environnement Windows (Csrss) et, s'il y a lieu, une autre crée par le gestionnaire graphique dans le cas où le thread à impliqué dans son traitement une fonction USER ou GDI.

Bloc thread de l'exécutif (ETHREAD)

Le bloc ETHREAD se présente comme suit.

```
lkd> dt nt!_ETHREAD
   +0x000 Tcb              : _KTHREAD
   +0x5d8 CreateTime       : _LARGE_INTEGER
   +0x5e0 ExitTime         : _LARGE_INTEGER
   +0x5e0 KeyedWaitChain   : _LIST_ENTRY
   +0x5f0 ChargeOnlySession : Ptr64 Void
   +0x5f8 PostBlockList    : _LIST_ENTRY
   +0x5f8 ForwardLinkShadow : Ptr64 Void
   +0x600 StartAddress     : Ptr64 Void
   +0x608 TerminationPort  : Ptr64 _TERMINATION_PORT
   +0x608 ReaperLink       : Ptr64 _ETHREAD
   +0x608 KeyedWaitValue   : Ptr64 Void
   +0x610 ActiveTimerListLock : Uint8B
   +0x618 ActiveTimerListHead : _LIST_ENTRY
   +0x628 Cid              : _CLIENT_ID
   +0x638 KeyedWaitSemaphore : _KSEMAPHORE
   +0x638 AlpcWaitSemaphore : _KSEMAPHORE
   +0x658 ClientSecurity   : _PS_CLIENT_SECURITY_CONTEXT
   +0x660 IrpList          : _LIST_ENTRY
   +0x670 TopLevelIrp      : Uint8B
   +0x678 DeviceToVerify   : Ptr64 _DEVICE_OBJECT
   +0x680 Win32StartAddress : Ptr64 Void
   +0x688 LegacyPowerObject : Ptr64 Void
   +0x690 ThreadListEntry  : _LIST_ENTRY
   +0x6a0 RundownProtect   : _EX_RUNDOWN_REF
   +0x6a8 ThreadLock       : _EX_PUSH_LOCK
   +0x6b0 ReadClusterSize  : Uint4B
   +0x6b4 MmLockOrdering   : Int4B
```

```
+0x6b8 CmLockOrdering   : Int4B
+0x6bc CrossThreadFlags : Uint4B
+0x6bc Terminated       : Pos 0, 1 Bit
+0x6bc ThreadInserted   : Pos 1, 1 Bit
+0x6bc HideFromDebugger : Pos 2, 1 Bit
+0x6bc ActiveImpersonationInfo : Pos 3, 1 Bit
+0x6bc HardErrorsAreDisabled : Pos 4, 1 Bit
+0x6bc BreakOnTermination : Pos 5, 1 Bit
+0x6bc SkipCreationMsg  : Pos 6, 1 Bit
+0x6bc SkipTerminationMsg : Pos 7, 1 Bit
+0x6bc CopyTokenOnOpen  : Pos 8, 1 Bit
+0x6bc ThreadIoPriority : Pos 9, 3 Bits
+0x6bc ThreadPagePriority : Pos 12, 3 Bits
+0x6bc RundownFail      : Pos 15, 1 Bit
+0x6bc UmsForceQueueTermination : Pos 16, 1 Bit
+0x6bc IndirectCpuSets  : Pos 17, 1 Bit
+0x6bc ReservedCrossThreadFlags : Pos 18, 14 Bits
+0x6c0 SameThreadPassiveFlags : Uint4B
+0x6c0 ActiveExWorker   : Pos 0, 1 Bit
+0x6c0 MemoryMaker      : Pos 1, 1 Bit
+0x6c0 StoreLockThread  : Pos 2, 1 Bit
+0x6c0 ClonedThread     : Pos 3, 1 Bit
+0x6c0 KeyedEventInUse  : Pos 4, 1 Bit
+0x6c0 SelfTerminate    : Pos 5, 1 Bit
+0x6c0 RespectIoPriority : Pos 6, 1 Bit
+0x6c0 ReservedSameThreadPassiveFlags : Pos 7, 25 Bits
+0x6c4 SameThreadApcFlags : Uint4B
+0x6c4 OwnsProcessAddressSpaceExclusive : Pos 0, 1 Bit
+0x6c4 OwnsProcessAddressSpaceShared : Pos 1, 1 Bit
+0x6c4 HardFaultBehavior : Pos 2, 1 Bit
+0x6c4 StartAddressInvalid : Pos 3, 1 Bit
+0x6c4 EtwCalloutActive : Pos 4, 1 Bit
+0x6c4 SuppressSymbolLoad : Pos 5, 1 Bit
+0x6c4 Prefetching      : Pos 6, 1 Bit
+0x6c4 OwnsVadExclusive : Pos 7, 1 Bit
+0x6c5 SystemPagePriorityActive : Pos 0, 1 Bit
+0x6c5 SystemPagePriority : Pos 1, 3 Bits
+0x6c8 CacheManagerActive : UChar
+0x6c9 DisablePageFaultClustering : UChar
+0x6ca ActiveFaultCount : UChar
+0x6cb LockOrderState   : UChar
+0x6d0 AlpcMessageId    : Uint8B
+0x6d8 AlpcMessage      : Ptr64 Void
+0x6d8 AlpcReceiveAttributeSet : Uint4B
+0x6e0 ExitStatus       : Int4B
+0x6e8 AlpcWaitListEntry : _LIST_ENTRY
+0x6f8 CacheManagerCount : Uint4B
+0x6fc IoBoostCount     : Uint4B
+0x700 BoostList        : _LIST_ENTRY
+0x710 DeboostList      : _LIST_ENTRY
+0x720 BoostListLock    : Uint8B
+0x728 IrpListLock      : Uint8B
+0x730 ReservedForSynchTracking : Ptr64 Void
+0x738 CmCallbackListHead : _SINGLE_LIST_ENTRY
+0x740 ActivityId       : Ptr64 _GUID
+0x748 SeLearningModeListHead : _SINGLE_LIST_ENTRY
+0x750 VerifierContext  : Ptr64 Void
+0x758 KernelStackReference : Uint4B
+0x760 AdjustedClientToken : Ptr64 Void
+0x768 WorkingOnBehalfClient : Ptr64 Void
+0x770 PropertySet      : _PS_PROPERTY_SET
+0x788 PicoContext      : Ptr64 Void
+0x790 UserFsBase       : Uint4B
+0x798 UserGsBase       : Uint8B
+0x7a0 EnergyValues     : Ptr64 _THREAD_ENERGY_VALUES
```

```
+0x7a8 CmCellReferences : Uint4B
+0x7b0 SelectedCpuSets  : Uint8B
+0x7b0 SelectedCpuSetsIndirect : Ptr64 Uint8B
+0x7b8 Silo             : Ptr64 _EJOB
```

- *Informations d'emprunt d'identité (ActiveImpersonationInfo)* Jeton d'accès et niveau d'emprunt d'identité (si le thread emploie ledit mécanisme). Voir routine système *PsIsThreadImpersonating*.
- *ActiveTimerListLock* Verrou de type spin lock utilisée pour protéger la liste des objets timer que le thread possède (ActiveTimerListHead). Voir routine ExTimerRundown.
- *Point d'arrêt à la fin (BreakOnTermination)* Voir service NtQueryInformationThread (ThreadInformationClass = ThreadBreakOnTermination) ; routine PspTerminateThreadByPointer.
- *ID client (Cid)* ID du thread et ID du processus par l'intermédiaire duquel il s'exécute. Voir routines système PsGetCurrentThreadProcessId, PsGetThreadId, PsGetThreadProcessId.
- *Contremarque de temps à la création (CreateTime)* Heures de début du thread. Voir service PspCreateThread.
- *DeviceToVerify* Voir routine IoGetDeviceToVerify.
- *ExitStatus* Voir routine PspExitThread, constante STATUS_THREAD_IS_TERMINATING.
- *Heure de fin (_ExitTime*)_ Date et heures où le thread a pris fin. Voir fonction GetThreadTimes ; service NtQueryInformationThread (ThreadInformationClass = ThreadBasicInformation) ; structure THREAD_BASIC_INFORMATION ; routine PspExitThread.
- *HardErrorsAreDisabled* Contrôle la manière dont le système se comporte à l'issue d'une erreur d'E/S de périphérique générée par le thread. Voir routines IoRaiseHardError, IoSetThreadHardErrorMode, PsGetThreadHardErrorsAreDisabled et PsSetThreadHardErrorsAreDisabled.
- *HideFromDebugger* Voir service NtSetInformationThread quand ThreadInformationClass vaut ThreadHideFromDebugger.
- *Informations d'E/S (IrpList)* Liste de paquets de requête d'E/S en instance. Voir service NtQueryInformationThread (ThreadInformationClass = ThreadIsIoPending).
- *KeyedEventInUse* Variable booléenne qui, du moment qu'elle est armée, indique que le thread exécute une routine apparentée à l'attente d'objets événement à clé. Voir service NtReleaseKeyedEvent et NtWaitForKeyedEvent.
- *KeyedWaitChain* Voir service natif NtWaitForKeyedEvent.
- *Valeur de clé attendue (KeyedWaitValue)* Voir services NtReleaseKeyedEvent et NtWaitForKeyedEvent.
- *Prefetching* Voir routine PsIsCurrentThreadPrefetching.
- *ReadClusterSize* Voir routine PspCreateThread ; variable noyau MmReadClusterSize.
- *RundownProtect* Voir routine PspCreateThread et ExInitializeRundownProtection.
- *Adresse de départ (StartAddress)* Adresse de la routine de démarrage du thread.
- *Thread système (SystemThread)* Indique si le thread fait partie de ceux exécutés dans l'espace noyau (par opposition aux threads mode utilisateur), et considérés à ce titre comme partie intégrante du système d'exploitation. Voir routines système PsCreateSystemThread et PsIsSystemThread.
- *Port de terminaison (TerminationPort)* Voir service NtRegisterThreadTerminatePort.
- *UserFsBase* et *UserGsBase* Contrôle l'intégrité du flux d'exécution des programmes mode utilisateur (RFG, Return Flow Guard).

- *État terminé (Terminated)* État interne utilisé de façon à mettre en évidence le fait que le thread prend fin. Voir service NtQueryInformationThread (ThreadInformationClass = ThreadIsTerminated). Voir routine *PsIsThreadTerminating*.
- *Verrou (ThreadLock)* Assure la synchronisation du basculement de contexte de thread.
- TopLevelIrp Voir routine IoGetTopLevelIrp et IoSetTopLevelIrp.
- *Win32StartAddress* Voir fonction BaseProcessStartup, service NtQueryInformationThread (ThreadInformationClass = ThreadQuerySetWin32StartAddress).

Tableau 8.23. ETHREAD CrossThreadFlags

Constante	Valeur
PS_CROSS_THREAD_FLAGS_TERMINATED	0x00000001
PS_CROSS_THREAD_FLAGS_DEADTHREAD	0x00000002
PS_CROSS_THREAD_FLAGS_HIDEFROMDBG	0x00000004
PS_CROSS_THREAD_FLAGS_IMPERSONATING	0x00000008
PS_CROSS_THREAD_FLAGS_SYSTEM	0x00000010
PS_CROSS_THREAD_FLAGS_HARD_ERRORS_DISABLED	0x00000020
PS_CROSS_THREAD_FLAGS_BREAK_ON_TERMINATION	0x00000040
PS_CROSS_THREAD_FLAGS_SKIP_CREATION_MSG	0x00000080
PS_CROSS_THREAD_FLAGS_SKIP_TERMINATION_MSG	0x00000100

Bloc thread du noyau (KTHREAD)

Le noyau Windows gère chaque thread par l'intermédiaire d'une structure KTHREAD (Kernel Thread), laquelle héberge des informations liées essentiellement aux stratégies motrices pour le compte de telles entités : ordonnancement (priorité de base et priorité courante, quantum, affinité avec un processeur), synchronisation (entête dispatcher, état signalé ou non), primitives d'attente (liste des objets que le thread attend), liaison vers le mode noyau (pointeur vers table des services système), stockage (pointeur de pile noyau), interceptions (trame d'interception, mise en file d'attente et livraison des appels de procédure asynchrones), état (prêt, en exécution, en attente), et ainsi de suite.

Pour plus de détails sur la structure interne d'un bloc KTHREAD, utilisez la commande dt du debogueur noyau.

```
lkd> dt nt!_KTHREAD @$Thread
   +0x000 Header           : _DISPATCHER_HEADER
   +0x018 SListFaultAddress : (null)
   +0x020 QuantumTarget    : 0x791ddc0
   +0x028 InitialStack     : 0xfffff800`f2189c10 Void
   +0x030 StackLimit       : 0xfffff800`f2183000 Void
   +0x038 StackBase        : 0xfffff800`f218a000 Void
   +0x040 ThreadLock       : 0
   +0x048 CycleTime        : 0x000d6221`e0a1dd00
   +0x050 CurrentRunTime   : 0xffffffff
   +0x054 ExpectedRunTime  : 0
   +0x058 KernelStack      : 0xfffff800`f2189ba0 Void
   +0x060 StateSaveArea    : 0xfffff800`f2189c40 _XSAVE_FORMAT
   +0x068 SchedulingGroup  : (null)
   +0x070 WaitRegister     : _KWAIT_STATUS_REGISTER
   +0x071 Running          : 0x1 ''
   +0x072 Alerted          : [2]  ""
   +0x074 AutoBoostActive  : 0y1
   +0x074 ReadyTransition  : 0y0
   +0x074 WaitNext         : 0y0
   +0x074 SystemAffinityActive : 0y1
   +0x074 Alertable        : 0y0
   +0x074 UserStackWalkActive : 0y0
   +0x074 ApcInterruptRequest : 0y0
   +0x074 QuantumEndMigrate : 0y0
```

```
+0x074 UmsDirectedSwitchEnable : 0y0
+0x074 TimerActive      : 0y0
+0x074 SystemThread     : 0y1
+0x074 ProcessDetachActive : 0y0
+0x074 CalloutActive    : 0y0
+0x074 ScbReadyQueue    : 0y0
+0x074 ApcQueueable     : 0y0
+0x074 ReservedStackInUse : 0y0
+0x074 UmsPerformingSyscall : 0y0
+0x074 TimerSuspended   : 0y0
+0x074 SuspendedWaitMode : 0y0
+0x074 SuspendSchedulerApcWait : 0y0
+0x074 Reserved         : 0y000000000000 (0)
+0x074 MiscFlags        : 0n1033
+0x078 AutoAlignment    : 0y1
+0x078 DisableBoost     : 0y0
+0x078 ThreadFlagsSpare0 : 0y0
+0x078 AlertedByThreadId : 0y0
+0x078 QuantumDonation  : 0y0
+0x078 EnableStackSwap  : 0y1
+0x078 GuiThread        : 0y0
+0x078 DisableQuantum   : 0y0
+0x078 ChargeOnlySchedulingGroup : 0y0
+0x078 DeferPreemption  : 0y0
+0x078 QueueDeferPreemption : 0y0
+0x078 ForceDeferSchedule : 0y0
+0x078 SharedReadyQueueAffinity : 0y0
+0x078 FreezeCount      : 0y0
+0x078 TerminationApcRequest : 0y0
+0x078 AutoBoostEntriesExhausted : 0y1
+0x078 KernelStackResident : 0y1
+0x078 CommitFailTerminateRequest : 0y0
+0x078 ProcessStackCountDecremented : 0y0
+0x078 RestrictedGuiThread : 0y0
+0x078 ThreadFlagsSpare : 0y0000
+0x078 EtwStackTraceApcInserted : 0y00000000 (0)
+0x078 ThreadFlags      : 0n98337
+0x07c Tag              : 0 ''
+0x07d SystemHeteroCpuPolicy : 0 ''
+0x07e UserHeteroCpuPolicy : 0y0001000 (0x8)
+0x07e ExplicitSystemHeteroCpuPolicy : 0y0
+0x07f Spare0           : 0 ''
+0x080 SystemCallNumber : 0
+0x084 Spare10          : 0
+0x088 FirstArgument    : (null)
+0x090 TrapFrame        : (null)
+0x098 ApcState         : _KAPC_STATE
+0x098 ApcStateFill     : [43]  "???"
+0x0c3 Priority         : 0 ''
+0x0c4 UserIdealProcessor : 0
+0x0c8 WaitStatus       : 0n0
+0x0d0 WaitBlockList    : (null)
+0x0d8 WaitListEntry    : _LIST_ENTRY [ 0x00000000`00000000 - 0x00000000`00000000 ]
+0x0d8 SwapListEntry    : _SINGLE_LIST_ENTRY
+0x0e8 Queue            : (null)
+0x0f0 Teb              : (null)
+0x0f8 RelativeTimerBias : 0
+0x100 Timer            : _KTIMER
+0x140 WaitBlock        : [4] _KWAIT_BLOCK
+0x140 WaitBlockFill4   : [20]  ""
+0x154 ContextSwitches  : 0x1e3e35a
+0x140 WaitBlockFill5   : [68]  ""
+0x184 State            : 0x2 ''
+0x185 Spare13          : 0 ''
+0x186 WaitIrql         : 0x2 ''
```

```
+0x187 WaitMode         : 0 ''
+0x140 WaitBlockFill6   : [116]  ""
+0x1b4 WaitTime         : 0x1d8b3783
+0x140 WaitBlockFill7   : [164]  ""
+0x1e4 KernelApcDisable : 0n0
+0x1e6 SpecialApcDisable : 0n0
+0x1e4 CombinedApcDisable : 0
+0x140 WaitBlockFill8   : [40]  ""
+0x168 ThreadCounters   : (null)
+0x140 WaitBlockFill9   : [88]  ""
+0x198 XStateSave       : (null)
+0x140 WaitBlockFill10  : [136]  ""
+0x1c8 Win32Thread      : (null)
+0x140 WaitBlockFill11  : [176]  ""
+0x1f0 Ucb              : (null)
+0x1f8 Uch              : (null)
+0x200 TebMappedLowVa   : (null)
+0x208 QueueListEntry   : _LIST_ENTRY [ 0x00000000`00000000 - 0x00000000`00000000 ]
+0x218 NextProcessor    : 0
+0x218 NextProcessorNumber : 0y00000000000000000000000000000000 (0)
+0x218 SharedReadyQueue : 0y0
+0x21c QueuePriority    : 0n0
+0x220 Process          : 0xfffff800`f043d940 _KPROCESS
+0x228 UserAffinity     : _GROUP_AFFINITY
+0x228 UserAffinityFill : [10]  "???"
+0x232 PreviousMode     : 0 ''
+0x233 BasePriority     : 0 ''
+0x234 PriorityDecrement : 0 ''
+0x234 ForegroundBoost  : 0y0000
+0x234 UnusualBoost     : 0y0000
+0x235 Preempted        : 0 ''
+0x236 AdjustReason     : 0 ''
+0x237 AdjustIncrement  : 0 ''
+0x238 AffinityVersion  : 0x453c
+0x240 Affinity         : _GROUP_AFFINITY
+0x240 AffinityFill     : [10]  "???"
+0x24a ApcStateIndex    : 0x1 ''
+0x24b WaitBlockCount   : 0 ''
+0x24c IdealProcessor   : 0
+0x250 NpxState         : 0x1c
+0x258 SavedApcState    : _KAPC_STATE
+0x258 SavedApcStateFill : [43]  "???"
+0x283 WaitReason       : 0 ''
+0x284 SuspendCount     : 0 ''
+0x285 Saturation       : 0 ''
+0x286 SListFaultCount  : 0
+0x288 SchedulerApc     : _KAPC
+0x288 SchedulerApcFill0 : [1]  "???"
+0x289 ResourceIndex    : 0 ''
+0x288 SchedulerApcFill1 : [3]  "???"
+0x28b QuantumReset     : 0x7f ''
+0x288 SchedulerApcFill2 : [4]  "???"
+0x28c KernelTime       : 0x5c4500e
+0x288 SchedulerApcFill3 : [64]  "???"
+0x2c8 WaitPrcb         : (null)
+0x288 SchedulerApcFill4 : [72]  "???"
+0x2d0 LegoData         : (null)
+0x288 SchedulerApcFill5 : [83]  "???"
+0x2db CallbackNestingLevel : 0 ''
+0x2dc UserTime         : 0
+0x2e0 SuspendEvent     : _KEVENT
+0x2f8 ThreadListEntry  : _LIST_ENTRY [ 0xffffac81`9f00ceb8 - 0xfffff800`f043d970 ]
+0x308 MutantListHead   : _LIST_ENTRY [ 0xfffff800`f043ec48 - 0xfffff800`f043ec48 ]
+0x318 AbEntrySummary   : 0x3f '?'
+0x319 AbWaitEntryCount : 0 ''
```

```
+0x31a Spare20          : 0
+0x31c SecureThreadCookie : 0
+0x320 LockEntries      : [6] _KLOCK_ENTRY
+0x560 PropagateBoostsEntry : _SINGLE_LIST_ENTRY
+0x568 IoSelfBoostsEntry : _SINGLE_LIST_ENTRY
+0x570 PriorityFloorCounts : [16]  ""
+0x580 PriorityFloorSummary : 0
+0x584 AbCompletedIoBoostCount : 0n0
+0x588 AbCompletedIoQoSBoostCount : 0n0
+0x58c KeReferenceCount : 0n0
+0x58e AbOrphanedEntrySummary : 0 ''
+0x58f AbOwnedEntryCount : 0 ''
+0x590 ForegroundLossTime : 0
+0x598 GlobalForegroundListEntry : _LIST_ENTRY [ 0x00000000`00000001 - 0x00000000`00000000 ]
+0x598 ForegroundDpcStackListEntry : _SINGLE_LIST_ENTRY
+0x5a0 InGlobalForegroundList : 0
+0x5a8 ReadOperationCount : 0n0
+0x5b0 WriteOperationCount : 0n0
+0x5b8 OtherOperationCount : 0n0
+0x5c0 ReadTransferCount : 0n0
+0x5c8 WriteTransferCount : 0n0
+0x5d0 OtherTransferCount : 0n0
+0x5d8 QueuedScb        : (null)
```

- *AdjustReason* Donne la raison pour laquelle le système a ajusté la priorité d'ordonnancement du thread.
- *ApcQueueable* Indique si la mise en file d'APC est possible pour ce thread.
- *État APC (ApcState)*
- **Masque d'affinité (Affinity)** Masque de bits qui spécifie les processeurs sur lesquels les threads du processus sont autorisés à fonctionner. Voir fonction SetThreadAffinityMask ; services NtQueryInformationThread(ThreadInformationClass = ThreadBasicInformation), NtSetInformationThread (ThreadInformationClass = ThreadAffinityMask) ; routine KeSetAffinityThread.
- *État d'attente altertable (Alertable)* Détermine si le thread peut entrer en état d'attente d'alertable, et de la sorte recevoir des APC.
- *AutoAlignment* Indique comment sont gérées les erreurs d'alignement de données, soit traitées automatiquement par le noyau, soit soulevées en tant qu'exceptions. Voir routines KeQueryAutoAlignmentThread et KeSetAutoAlignmentThread.
- *Priorité de base (_BasePriority*)_ Priorité de base actuelle du thread ; cette valeur peut différer de la priorité initiale du thread. Voir routine KeQueryBasePriorityThread.
- *Changements de contexte (ContextSwitches)* Nombre de basculements de contexte.
- **Contrôle d'accélération de priorité (DisableBoost)** Indique si un thread est éligible ou non à une augmentation temporaire de sa priorité d'ordonnancement. Voir fonctions GetThreadPriorityBoost et SetThreadPriorityBoost ; service NtQueryInformationThread (ThreadInformationClass = ThreadPriorityBoost) ; routine KeSetDisableBoostThread.
- **Compteur de gels (FreezeCount)** Nombre de fois qu'un thread a été mis en pause. Voir routines système PsGetThreadFreezeCount et KeFreezeAllThreads.
- **Thread GUI (GuiThread)** Indique si le thread est exécutée dans le contexte d'une application graphique, ce qui lui permet de bénéficier d'une pile noyau plus large. Voir routine PsConvertToGuiThread.
- **Entête dispatcher (Header)** Confère au thread la possibilité d'être attendu. Voir fonctions WaitForSingleObject et consorts.

- **Processeur idéal (IdealProcessor)** Processeur privilégié pour l'exécution du thread. Voir fonction SetThreadIdealProcessor, service NtSetInformationThread (ThreadInformationClass = ThreadIdealProcessor) et routine KeSetIdealProcessorThread.
- **Pile initiale (InitialStack)** Contient un pointeur vers la pile mode noyau du thread. Voir routine IoGetInitialStack.
- **Pile résidente (KernelStackResident)** Force la pile noyau du thread à résider en mémoire physique, y compris s'il est inactif. Voir routine KiInSwapKernelStacks.
- *Temps d'exécution en mode noyau (KernelTime)* Voir routine KeQueryRuntimeThread et KeUpdateRunTime ; compteur de performance Thread: % Temps privilégié.
- *Liste de mutants (_MutantListHead*)_ Liste des objets mutant que le thread a en sa possession.
- *Compteur d'E/S autre (OtherOperationCount)* Nombre d'opérations d'E/S qui ne sont ni des lectures ni des écritures (fonctions de contrôle, par exemple) générées par le processus depuis son démarrage.
- *Priorité actuelle (_Priority*)_ Priorité dynamique actuelle du thread. Voir routines KeQueryPriorityThread et KeSetPriorityThread.
- *Processus (Process)* Pointeur vers le bloc KPROCESS du processus dans lequel le thread s'exécute. Voir routine PsGetCurrentThreadProcess.
- *Mode précédent (PreviousMode)* Voir routines ExGetPreviousMode, KeGetPreviousModeByThread et PsGetCurrentThreadPreviousMode.
- *Liste de files (_Queue*)_ Pointeur vers objet file auquel est associé le thread.
- *Compteur d'E/S lecture (ReadOperationCount)* Nombre d'opérations d'E/S lecture générées par le processus depuis son démarrage. Voir service NtReadFile ; routine KeQueryValuesProcess.
- *SpecialApcDisable* Voir routine KeEnterGuardedRegion.
- **Base de pile (StackBase)** Adresse de base de la pile noyau du thread. Cette valeur correspond à la limite supérieure de l'espace mémoire réservé à la pile, qui décroît vers des adresses inférieures à mesure que des données y sont empilées. Voir routine PsGetCurrentThreadStackBase.
- **Limite de pile (StackLimit)** Adresse minimale jusqu'à laquelle la pile noyau du thread peut s'étendre. Cette valeur correspond à la limite inférieure de l'espace mémoire réservé à la pile, qui décroît à mesure que de nouvelles données y sont ajoutées. Lorsque la pile atteint cette adresse et qu'aucune extension n'est possible, une exception de débordement de pile est générée. Voir routine PsGetCurrentThreadStackLimit.
- **État d'ordonnancement (State)** État actuel du thread en regard de la stratégie système d'ordonnancement, déterminant par exemple s'il est en cours d'exécution, prêt à être exécuté, en attente, suspendu, etc. Voir KTHREAD_STATE.
- *Compteur de suspension (SuspendCount)* Compteur enregistrant le nombre de fois que le thread a été suspendu. Voir fonctions Windows CreateThread (dwCreationFlags = CREATE_SUSPENDED), ResumeThread et SuspendThread ; services NtResumeThread et NtSuspendThread ; routines KeResumeThread, PsResumeThread et PsSuspendThread.
- *Affinité système (SystemAffinityActive)* Indique si l'affinité système à l'échelle du système est appliquée à ce thread. Voir KeRevertToUserAffinityThread et KeSetSystemAffinityThread.
- *Pointeur vers TEB (_Teb*)_ Pointeur vers le bloc d'environnement (TEB) auquel est associé le thread. Voir routines PsGetCurrentThreadTeb et PsGetThreadTeb.
- *Trame d'interception (TrapFrame)*

- *Affinité de niveau utilisateur (UserAffinity)* Masque d'affinité du thread du moment qu'il s'exécute en mode utilisateur.
- *Temps d'exécution en mode utilisateur (UserTime)* Comptabilise le temps que le thread a passé en mode utilisateur. Voir routines KeQueryRuntimeThread et KeUpdateRunTime ; compteur de performance Thread: % Temps utilisateur.
- *Mode d'attente (WaitMode)* Mode processeur (utilisateur ou noyau) dans lequel l'attente du thread a lieu. Voir routines KeDelayExecutionThread (paramètre WaitMode), KeWaitForMultipleObjects (paramètre WaitMode), KeWaitForSingleObject (paramètre WaitMode) ; structure KPROCESSOR_MODE.
- *Compteur de blocs d'attente (WaitBlockCount)* Nombre de blocs d'attente que le thread renferme.
- *Raison d'attente (WaitReason)* Indique la raison pour laquelle le thread se trouve actuellement en état d'attente. Voir routines KeWaitForMultipleObjects (paramètre WaitReason) et KeWaitForSingleObject (paramètre WaitReason) ; structure de données KWAIT_REASON.
- *WaitTime* Heure à laquelle le thread est entré dans l'état d'attente. Fournit une donnée temporelle évaluée par le gestionnaire d'équilibre afin de déterminer si un thread a besoin d'accélération de priorité. Quand ledit composant détecte un thread qui a été en attente pendant une période prolongée, il augmente la priorité du thread, définit son quantum à une valeur appropriée et l'insère dans la liste des Prêts.
- *Win32Thread* Voir routines PsGetCurrentThreadWin32Thread et PsSetThreadWin32Thread.
- *Compteur d'E/S écriture (WriteOperationCount)* Nombre d'opérations d'E/S écriture auquel le thread a donné lieu. Voir routine KeQueryValuesProcess.

Tableau 8.24. KTHREAD WaitStatus

Constante	Valeur
STATUS_KERNEL_APC	0x00000100
STATUS_USER_APC	0x000000C0
STATUS_ALERTED	0x00000101
STATUS_TIMEOUT	0x00000102
STATUS_ABANDONED	0x00000080

Tableau 8.25. KWAIT_REASON

Indice	Raison d'attente
00	Executive
01	FreePage
02	PageIn
03	PoolAllocation
04	DelayExecution
05	Suspended
06	UserRequest
07	WrExecutive
08	WrFreePage
09	WrPageIn
10	WrPoolAllocation
11	WrDelayExecution
12	WrSuspended
13	WrUserRequest
14	WrEventPair
15	WrQueue
16	WrLpcReceive

Indice	Raison d'attente
17	WrLpcReply
18	WrVirtualMemory
19	WrPageOut
20	WrRendezvous
21	Spare2
22	Spare3
23	Spare4
24	Spare5
25	Spare6
26	WrKernel
27	WrResource
28	WrPushLock
29	WrMutex
30	WrQuantumEnd
31	WrDispatchInt
32	WrPreempted
33	WrYieldExecution
34	WrFastMutex
35	WrGuardedMutex
36	WrRundown

TEB (*Thread Environment Block*)

Le bloc d'environnement de thread (TEB, *Thread Environment Block*) renferme des informations de contexte sur lesquelles s'appuient les composants de Windows exécutés en mode utilisateur, dont le chargeur d'image, le gestionnaire de tas et, dans une large mesure, les modules (DLL) chargés suite à la naissance d'un processus. Comme le bloc PEB, duquel il est du reste un direct parent, le bloc TEB existe dans l'espace d'adressage du processus, le rendant de ce fait facilement et rapidement accessible.

La commande !teb des débogueurs Windows standards procure une vue simplifiée du bloc d'environnement d'un thread.

```
lkd> !teb
TEB at 0000006894b8f000
    ExceptionList:        0000000000000000
    StackBase:            0000006895000000
    StackLimit:           0000006894ff0000
    SubSystemTib:         0000000000000000
    FiberData:            0000000000001e00
    ArbitraryUserPointer: 0000000000000000
    Self:                 0000006894b8f000
    EnvironmentPointer:   0000000000000000
    ClientId:             0000000000001680 . 0000000000001584
    RpcHandle:            0000000000000000
    Tls Storage:          00000256a7a13cb0
    PEB Address:          0000006894b7e000
    LastErrorValue:       0
    LastStatusValue:      c0000139
    Count Owned Locks:    0
    HardErrorMode:        0

lkd> dt nt!_TEB
   +0x000 NtTib            : _NT_TIB
   +0x038 EnvironmentPointer : Ptr64 Void
   +0x040 ClientId         : _CLIENT_ID
   +0x050 ActiveRpcHandle  : Ptr64 Void
```

```
+0x058 ThreadLocalStoragePointer : Ptr64 Void
+0x060 ProcessEnvironmentBlock : Ptr64 _PEB
+0x068 LastErrorValue   : Uint4B
+0x06c CountOfOwnedCriticalSections : Uint4B
+0x070 CsrClientThread  : Ptr64 Void
+0x078 Win32ThreadInfo  : Ptr64 Void
+0x080 User32Reserved   : [26] Uint4B
+0x0e8 UserReserved     : [5] Uint4B
+0x100 WOW32Reserved    : Ptr64 Void
+0x108 CurrentLocale    : Uint4B
+0x10c FpSoftwareStatusRegister : Uint4B
+0x110 ReservedForDebuggerInstrumentation : [16] Ptr64 Void
+0x190 SystemReserved1  : [38] Ptr64 Void
+0x2c0 ExceptionCode    : Int4B
+0x2c4 Padding0         : [4] UChar
+0x2c8 ActivationContextStackPointer : Ptr64 _ACTIVATION_CONTEXT_STACK
+0x2d0 InstrumentationCallbackSp : Uint8B
+0x2d8 InstrumentationCallbackPreviousPc : Uint8B
+0x2e0 InstrumentationCallbackPreviousSp : Uint8B
+0x2e8 TxFsContext      : Uint4B
+0x2ec InstrumentationCallbackDisabled : UChar
+0x2ed Padding1         : [3] UChar
+0x2f0 GdiTebBatch      : _GDI_TEB_BATCH
+0x7d8 RealClientId     : _CLIENT_ID
+0x7e8 GdiCachedProcessHandle : Ptr64 Void
+0x7f0 GdiClientPID     : Uint4B
+0x7f4 GdiClientTID     : Uint4B
+0x7f8 GdiThreadLocalInfo : Ptr64 Void
+0x800 Win32ClientInfo  : [62] Uint8B
+0x9f0 glDispatchTable  : [233] Ptr64 Void
+0x1138 glReserved1      : [29] Uint8B
+0x1220 glReserved2      : Ptr64 Void
+0x1228 glSectionInfo    : Ptr64 Void
+0x1230 glSection        : Ptr64 Void
+0x1238 glTable          : Ptr64 Void
+0x1240 glCurrentRC      : Ptr64 Void
+0x1248 glContext        : Ptr64 Void
+0x1250 LastStatusValue  : Uint4B
+0x1254 Padding2         : [4] UChar
+0x1258 StaticUnicodeString : _UNICODE_STRING
+0x1268 StaticUnicodeBuffer : [261] Wchar
+0x1472 Padding3         : [6] UChar
+0x1478 DeallocationStack : Ptr64 Void
+0x1480 TlsSlots         : [64] Ptr64 Void
+0x1680 TlsLinks         : _LIST_ENTRY
+0x1690 Vdm              : Ptr64 Void
+0x1698 ReservedForNtRpc : Ptr64 Void
+0x16a0 DbgSsReserved    : [2] Ptr64 Void
+0x16b0 HardErrorMode    : Uint4B
+0x16b4 Padding4         : [4] UChar
+0x16b8 Instrumentation  : [11] Ptr64 Void
+0x1710 ActivityId       : _GUID
+0x1720 SubProcessTag    : Ptr64 Void
+0x1728 PerflibData      : Ptr64 Void
+0x1730 EtwTraceData     : Ptr64 Void
+0x1738 WinSockData      : Ptr64 Void
+0x1740 GdiBatchCount    : Uint4B
+0x1744 CurrentIdealProcessor : _PROCESSOR_NUMBER
+0x1744 IdealProcessorValue : Uint4B
+0x1744 ReservedPad0     : UChar
+0x1745 ReservedPad1     : UChar
+0x1746 ReservedPad2     : UChar
+0x1747 IdealProcessor   : UChar
+0x1748 GuaranteedStackBytes : Uint4B
+0x174c Padding5         : [4] UChar
```

```
+0x1750 ReservedForPerf  : Ptr64 Void
+0x1758 ReservedForOle   : Ptr64 Void
+0x1760 WaitingOnLoaderLock : Uint4B
+0x1764 Padding6         : [4] UChar
+0x1768 SavedPriorityState : Ptr64 Void
+0x1770 ReservedForCodeCoverage : Uint8B
+0x1778 ThreadPoolData   : Ptr64 Void
+0x1780 TlsExpansionSlots : Ptr64 Ptr64 Void
+0x1788 DeallocationBStore : Ptr64 Void
+0x1790 BStoreLimit      : Ptr64 Void
+0x1798 MuiGeneration    : Uint4B
+0x179c IsImpersonating  : Uint4B
+0x17a0 NlsCache         : Ptr64 Void
+0x17a8 pShimData        : Ptr64 Void
+0x17b0 HeapVirtualAffinity : Uint2B
+0x17b2 LowFragHeapDataSlot : Uint2B
+0x17b4 Padding7         : [4] UChar
+0x17b8 CurrentTransactionHandle : Ptr64 Void
+0x17c0 ActiveFrame      : Ptr64 _TEB_ACTIVE_FRAME
+0x17c8 FlsData          : Ptr64 Void
+0x17d0 PreferredLanguages : Ptr64 Void
+0x17d8 UserPrefLanguages : Ptr64 Void
+0x17e0 MergedPrefLanguages : Ptr64 Void
+0x17e8 MuiImpersonation : Uint4B
+0x17ec CrossTebFlags    : Uint2B
+0x17ec SpareCrossTebBits : Pos 0, 16 Bits
+0x17ee SameTebFlags     : Uint2B
+0x17ee SafeThunkCall    : Pos 0, 1 Bit
+0x17ee InDebugPrint     : Pos 1, 1 Bit
+0x17ee HasFiberData     : Pos 2, 1 Bit
+0x17ee SkipThreadAttach : Pos 3, 1 Bit
+0x17ee WerInShipAssertCode : Pos 4, 1 Bit
+0x17ee RanProcessInit   : Pos 5, 1 Bit
+0x17ee ClonedThread     : Pos 6, 1 Bit
+0x17ee SuppressDebugMsg : Pos 7, 1 Bit
+0x17ee DisableUserStackWalk : Pos 8, 1 Bit
+0x17ee RtlExceptionAttached : Pos 9, 1 Bit
+0x17ee InitialThread    : Pos 10, 1 Bit
+0x17ee SessionAware     : Pos 11, 1 Bit
+0x17ee LoadOwner        : Pos 12, 1 Bit
+0x17ee LoaderWorker     : Pos 13, 1 Bit
+0x17ee SpareSameTebBits : Pos 14, 2 Bits
+0x17f0 TxnScopeEnterCallback : Ptr64 Void
+0x17f8 TxnScopeExitCallback : Ptr64 Void
+0x1800 TxnScopeContext  : Ptr64 Void
+0x1808 LockCount        : Uint4B
+0x180c WowTebOffset     : Int4B
+0x1810 ResourceRetValue : Ptr64 Void
+0x1818 ReservedForWdf   : Ptr64 Void
+0x1820 ReservedForCrt   : Uint8B
+0x1828 EffectiveContainerId : _GUID
```

- **Trame active (ActiveFrame)** Trame d'exception actuellement active, correspondant dans la perspective SEH au dernier gestionnaire d'exception installé pour un thread. Voir fonction RtlGetFrame. (S'applique uniquement aux systèmes x86.)

- **Données RPC (ActiveRpcHandle)** Métadonnées associées à l'appel RPC en cours d'exécution sur un thread.

- **ID client (Cid)** Structure contenant l'identifiant du thread ainsi que celui du processus par l'intermédiaire duquel il il s'exécute. Ces valeurs correspondent à celles retournées respectivement par les fonctions GetCurrentThreadId et GetCurrentProcessId.

- **Compteur de sections critique (CountOfOwnedCriticalSections)** Nombre de sections critiques qu'un thread a actuellement en sa possession (c'est-à-dire verrouillées). Une telle valeur est notamment utile pour

détecter d'éventuelles fuites de synchronisation, par exemple lorsqu'un thread se termine sans avoir au préalable libéré les sections critiques qui lui étaient associées.

- **Informations Csrss (CsrClientThread)** Informations spécifiques aux threads enregistrés avec le sous-système Csrss. Voir structure CSR_THREAD.
- *Locale Courante (_CurrentLocale*)_ Ensemble de définitions de textes et de formats utiles à la régionalisation de Microsoft Windows et des logiciels tiers. Voir fonctions GetThreadLocale et SetThreadLocale.
- **Transaction active (CurrentTransactionHandle)** Handle représentant l'opération transactionnelle active, si le thread en exécute une. Voir fonctions CreateFileTransacted et RtlSetCurrentTransaction.
- **Informations de débogage (DbgSsReserved)** Structure interne qu'alimente le sous-système de débogage (DbgSs, Debug Subsystem) dès lors qu'un processus fait l'objet d'un examen.
- **Bloc d'environnement (EnvironmentPointer)** Pointeur vers un bloc mémoire contenant les variables d'environnement d'un thread. Sur les versions modernes de Windows, ce champ ne semble plus être utilisé.
- **Code d'exception (ExceptionCode)** Dernier code d'exception généré par le thread.
- **Liste d'exceptions (NtTib.ExceptionList)** Liste des exceptions utilisée dans le cadre de la gestion structurée des exceptions. Voir fonction AddVectoredExceptionHandler.
- **Table de ventilation OpenGL (glDispatchTable)** Tableau de pointeurs vers les appels OpenGL standards (glClear, glDrawArrays et bien d'autres).
- **Taille garantie de la pile (GuaranteedStackBytes)** Taille minimale de la pile réservée et garantie comme disponible pour un thread. Voir fonction SetThreadStackGuarantee.
- **Mode d'erreur (HardErrorMode)** Chaque thread a un mode d'erreur associée qui indique au système comment l'application va répondre à de graves erreurs. Voir fonctions GetErrorMode et SetErrorMode ; serviceS RtlGetThreadErrorMode et RtlSetThreadErrorMode.
- **Emprunt d'identité (IsImpersonating)** Indicateur mettant en évidence que le thread emprunte momentanément l'identité d'un client.
- **Mode fibre (HasFiberData)** Détermine si le thread se trouve ou non être une fibre (fiber). Voir fonction *IsThreadAFiber*.
- *Paramètres régionaux durant l'emprunt d'identité (ImpersonationLocale)* Paramètres de régionalisation du thread tandis qu'il emprunte l'identité d'un client.
- **Dernière erreur (LastErrorValue)** Code d'erreur de la dernière fonction Windows à avoir été sollicité par le thread. Voir fonctions GetLastError et SetLastError, services RtlGetLastWin32Error et RtlSetLastWin32Error.
- **Dernier code d'état NT (LastStatusValue)** Code d'état NT (NTSTATUS) du dernier service natif à avoir été sollicité par le thread. Voir service RtlGetLastNtStatus.
- **Cache de localisation (NlsCache)** Informations de localisation (langue, formats de date, etc.) pour un thread.
- **PEB (ProcessEnvironmentBlock)** Pointeur vers le bloc d'environnement du processus auquel le thread appartient. Voir fonction RtlGetCurrentPeb.
- **Bloc OLE (ReservedForOle)** Pointeur vers une structure de données sur laquelle s'appuie le modèle d'objet composant (COM) pour gérer l'état d'un thread. Voir fonctions CoInitialize(Ex), structure `SOleThreadData`.
- **Tampon Unicode - Corps (StaticUnicodeBuffer)** Tableau de caractères Unicode sur lequel un thread peut s'appuyer pour des opérations temporaires de manipulation de chaînes, sans allocation dynamique.

- **Tampon Unicode - Entête (StaticUnicodeString)** Structure associée au tampon statique Unicode (StaticUnicodeBuffer), encapsulant la longueur effective, la capacité maximale et l'adresse du tampon proprement dite.
- **Pointeur TLS (ThreadLocalStoragePointer)** Pointeur vers les données locales au thread.
- **Emplacements TLS supplémentaires (TlsExpansionSlots)** Valeurs des emplacements TLS (Thread Local Storage) supplémentaires utilisés lorsque la capacité initiale définie à cet égard (64 slots) est dépassée. Le tableau ainsi constitué est alloué dynamiquement à partir du tas.
- **Emplacements TLS standards (TlsSlots)** Valeurs des 64 premiers emplacements TLS (Thread Local Storage) alloués pour un thread. Voir fonctions TlsGetValue et TlsSetValue.
- *Informations client User32 (User32Reserved)*
- **Informations client Win32 (Win32ClientInfo)** Bloc de données sur lequel le système s'appuie pour suivre les événements graphiques qui ont eu lieu au cours d'un thread : dernier message traité, valeurs de retour de quelques fonctions (par exemple, PeekMessage), coordonnées du curseur, association fenêtre, etc. L'existence d'une telle structure - et à cette place, c'est-à-dire intégrée au bloc d'environnement d'un thread - permet d'optimiser les performances en évitant de fréquentes transitions vers le noyau.
- *Données Winsock (WinSockData)* Regroupe différentes informations et données concernant Winsock.

Le tableau suivant énumère les routines noyau qui ont un rapport direct avec la structure TEB.

Tableau 8.26. Routines concernant le bloc TEB

Routine	Description
PsGetCurrentThreadTeb	Retourne le bloc TEB du thread courant.
PsGetThreadTeb	Retourne le bloc TEB d'un thread spécifié.

Région et bloc de contrôle du processeur

Essentielle à la manière dont chaque unité fonctionnelle (comprendre threads, mais aussi les processus qui les englobent) se manifeste à l'intérieur du noyau et de la couche d'abstraction matérielle, la région de contrôle du processeur (PCR, Processor Control Region), et son extension le bloc de contrôle du processeur (PCB, Processor Control Block), renferment des informations sur l'état de chaque processeur du système : IRQL courant, pointeur vers table des descripteurs d'interruption (IDT), thread en cours d'exécution, prochain thread, listes lookaside, spinlocks en file globaux, etc.

```
lkd> dtx nt!_KPCR @$pcr
(*((nt!_KPCR *)0xfffff80528109000))                [Type: _KPCR]
    [+0x000] NtTib            [Type: _NT_TIB]
    [+0x000] GdtBase          : 0xfffff8052f65afb0 [Type: _KGDTENTRY64 *]
    [+0x008] TssBase          : 0xfffff8052f659000 [Type: _KTSS64 *]
    [+0x010] UserRsp          : 0x0 [Type: unsigned __int64]
    [+0x018] Self             : 0xfffff80528109000 [Type: _KPCR *]
    [+0x020] CurrentPrcb      : 0xfffff80528109180 [Type: _KPRCB *]
    [+0x028] LockArray        : 0xfffff80528109870 [Type: _KSPIN_LOCK_QUEUE *]
    [+0x030] Used_Self        : 0x8dbb2e9000 [Type: void *]
    [+0x038] IdtBase          : 0xfffff8052f658000 [Type: _KIDTENTRY64 *]
    [+0x040] Unused           [Type: unsigned __int64 [2]]
    [+0x050] Irql             : 0x0 [Type: unsigned char]
    [+0x051] SecondLevelCacheAssociativity : 0x10 [Type: unsigned char]
    [+0x052] ObsoleteNumber   : 0x0 [Type: unsigned char]
    [+0x053] Fill0            : 0x0 [Type: unsigned char]
    [+0x054] Unused0          [Type: unsigned long [3]]
    [+0x060] MajorVersion     : 0x1 [Type: unsigned short]
    [+0x062] MinorVersion     : 0x1 [Type: unsigned short]
    [+0x064] StallScaleFactor : 0x450 [Type: unsigned long]
    [+0x068] Unused1          [Type: void * [3]]
```

```
[+0x080] KernelReserved   [Type: unsigned long [15]]
[+0x0bc] SecondLevelCacheSize : 0x400000 [Type: unsigned long]
[+0x0c0] HalReserved      [Type: unsigned long [16]]
[+0x100] Unused2          : 0x0 [Type: unsigned long]
[+0x108] KdVersionBlock   : 0x0 [Type: void *]
[+0x110] Unused3          : 0x0 [Type: void *]
[+0x118] PcrAlign1        [Type: unsigned long [24]]
[+0x180] Prcb             [Type: _KPRCB]
```

- **Bloc de contrôle (CurrentPrcb)** Pointeur vers le bloc de contrôle auquel est associé un processeur, incluant des informations d'état détaillées, les files de planification, des compteurs d'activité, et d'autres structures clés pour l'ordonnancement et la gestion du processeur.
- **Table des descripteurs (GdtBase)** Pointeur vers la table globale des descripteurs (GDT, Global Descriptor Table), employée par le processeur pour la segmentation mémoire. Chaque entrée de la GDT décrit un segment mémoire avec son étendue et ses droits d'accès. Voir structure KGDTENTRY.
- **Table des descripteurs d'interruption (IdtBase)** Adresse de la table des descripteurs d'interruption (IDT, *Interrupt Descriptor Table*). Chaque entrée au sein d'une telle structure décrit comment traiter une interruption ou exception donnée (adresse du gestionnaire, type, niveau de privilège requis, etc.). En interne, la valeur observée dans ce contexte reflète celle associée au registre matériel IDTR (*Interrupt Descriptor Table Register*), propre à chaque processeur et accessible via l'instruction `SIDT`. Dans un système multiprocesseur, chaque processeur peut éventuellement disposer d'une IDT distincte ; Windows emploie cependant une IDT commune (partagée) afin de rationaliser les mécanismes de gestion qui en résultent.
- **Niveau d'interruption (Irql)** Niveau d'interruption actuel sur un processeur, utilisé par le noyau Windows pour contrôler l'accès au processeur, la gestion des interruptions matérielles/logicielle, et la synchronisation des ressources.
- **Bloc KD (KdVersionBlock)** Regroupe différentes informations orientées débogage. Voir structures DBGKD_GET_VERSION64 et KDDEBUGGER_DATA64 ; variable noyau KdDebuggerDataBlock.
- **Spinlocks (LockArray)** Pointeur vers un tableau de spinlocks en file utilisé pour la synchronisation multi processeur. Chaque entrée parmi ce tableau représente une file d'attente associée à un verrou particulier. Voir structure `KSPIN_LOCK_QUEUE`.
- **Version majeure (MajorVersion)** Numéro interne utilisé en tant en tant qu'information de version, ici un numéro de version majeur, dont la valeur est calquée directement sur celle de la constante `PCR_MAJOR_VERSION`, définie à 1.
- **Version mineure (MinorVersion)** Numéro interne utilisé en tant en tant qu'information de version, ici un numéro de version mineur, dont la valeur est calquée directement sur la constante `PCR_MINOR_VERSION`, définie à 1.
- *Bloc TIB (NtTib)* Voir structure NT_TIB.
- *Numéro de processeur (Number)* Voir service NtGetCurrentProcessorNumber, routine KeGetCurrentProcessorNumber.
- *Degré d'associativité du cache L2 (SecondLevelCacheAssociativity)*
- *Taille du cache de second niveau (SecondLevelCacheSize)*
- **Pointeur KPCR (Self)** Pointeur vers la structure KPCR elle-même (autoréférence).
- **Facteur d'échelle (StallScaleFactor)** Facteur d'échelle utilisé pour convertir des délais exprimés en cycles CPU en durées temporelles.
- **Segment d'état de tâche (TssBase)** Adresse du segment d'état de tâche (TSS, Task state segment) pour un processeur, utilisé pour le traitement des interruptions.

- **Pointeur de pile utilisateur (UserRsp)** Pointeur sur la pile mode utilisateur du thread courant, utilisé pour revenir à l'espace utilisateur lors d'un retour de contexte depuis le mode noyau. Le bloc KPRCB a le format suivant.

```
lkd> dt nt!_KPRCB
   +0x000 MxCsr            : Uint4B
   +0x004 LegacyNumber     : UChar
   +0x005 ReservedMustBeZero : UChar
   +0x006 InterruptRequest : UChar
   +0x007 IdleHalt         : UChar
   +0x008 CurrentThread    : Ptr64 _KTHREAD
   +0x010 NextThread       : Ptr64 _KTHREAD
   +0x018 IdleThread       : Ptr64 _KTHREAD
   +0x020 NestingLevel     : UChar
   +0x021 ClockOwner       : UChar
   +0x022 PendingTickFlags : UChar
   +0x022 PendingTick      : Pos 0, 1 Bit
   +0x022 PendingBackupTick : Pos 1, 1 Bit
   +0x023 IdleState        : UChar
   +0x024 Number           : Uint4B
   +0x028 RspBase          : Uint8B
   +0x030 PrcbLock         : Uint8B
   +0x038 PriorityState    : Ptr64 Char
   +0x040 ProcessorState   : _KPROCESSOR_STATE
   +0x5f0 CpuType          : Char
   +0x5f1 CpuID            : Char
   +0x5f2 CpuStep          : Uint2B
   +0x5f2 CpuStepping      : UChar
   +0x5f3 CpuModel         : UChar
   +0x5f4 MHz              : Uint4B
   +0x5f8 HalReserved      : [8] Uint8B
   +0x638 MinorVersion     : Uint2B
   +0x63a MajorVersion     : Uint2B
   +0x63c BuildType        : UChar
   +0x63d CpuVendor        : UChar
   +0x63e CoresPerPhysicalProcessor : UChar
   +0x63f LogicalProcessorsPerCore : UChar
   +0x640 ParentNode       : Ptr64 _KNODE
   +0x648 GroupSetMember   : Uint8B
   +0x650 Group            : UChar
   +0x651 GroupIndex       : UChar
   +0x652 PrcbPad05        : [2] UChar
   +0x654 InitialApicId    : Uint4B
   +0x658 ScbOffset        : Uint4B
   +0x65c ApicMask         : Uint4B
   +0x660 AcpiReserved     : Ptr64 Void
   +0x668 CFlushSize       : Uint4B
   +0x66c PrcbPad10        : Uint4B
   +0x670 LockQueue        : [17] _KSPIN_LOCK_QUEUE
   +0x780 PPLookasideList  : [16] _PP_LOOKASIDE_LIST
   +0x880 PPNxPagedLookasideList : [32] _GENERAL_LOOKASIDE_POOL
   +0x1480 PPNPagedLookasideList : [32] _GENERAL_LOOKASIDE_POOL
   +0x2080 PPPagedLookasideList : [32] _GENERAL_LOOKASIDE_POOL
   +0x2c80 PrcbPad20        : Uint8B
   +0x2c88 DeferredReadyListHead : _SINGLE_LIST_ENTRY
   +0x2c90 MmPageFaultCount : Int4B
   +0x2c94 MmCopyOnWriteCount : Int4B
   +0x2c98 MmTransitionCount : Int4B
   +0x2c9c MmDemandZeroCount : Int4B
   +0x2ca0 MmPageReadCount  : Int4B
   +0x2ca4 MmPageReadIoCount : Int4B
   +0x2ca8 MmDirtyPagesWriteCount : Int4B
   +0x2cac MmDirtyWriteIoCount : Int4B
   +0x2cb0 MmMappedPagesWriteCount : Int4B
   +0x2cb4 MmMappedWriteIoCount : Int4B
```

```
+0x2cb8 KeSystemCalls    : Uint4B
+0x2cbc KeContextSwitches : Uint4B
+0x2cc0 LdtSelector      : Uint2B
+0x2cc2 PrcbPad40        : Uint2B
+0x2cc4 CcFastReadNoWait : Uint4B
+0x2cc8 CcFastReadWait   : Uint4B
+0x2ccc CcFastReadNotPossible : Uint4B
+0x2cd0 CcCopyReadNoWait : Uint4B
+0x2cd4 CcCopyReadWait   : Uint4B
+0x2cd8 CcCopyReadNoWaitMiss : Uint4B
+0x2cdc IoReadOperationCount : Int4B
+0x2ce0 IoWriteOperationCount : Int4B
+0x2ce4 IoOtherOperationCount : Int4B
+0x2ce8 IoReadTransferCount : _LARGE_INTEGER
+0x2cf0 IoWriteTransferCount : _LARGE_INTEGER
+0x2cf8 IoOtherTransferCount : _LARGE_INTEGER
+0x2d00 PacketBarrier    : Int4B
+0x2d04 TargetCount      : Int4B
+0x2d08 IpiFrozen        : Uint4B
+0x2d10 IsrDpcStats      : Ptr64 Void
+0x2d18 DeviceInterrupts : Uint4B
+0x2d1c LookasideIrpFloat : Int4B
+0x2d20 InterruptLastCount : Uint4B
+0x2d24 InterruptRate    : Uint4B
+0x2d28 LastNonHrTimerExpiration : Uint8B
+0x2d30 PrcbPad41        : [20] Uint4B
+0x2d80 DpcData          : [2] _KDPC_DATA
+0x2dd0 DpcStack         : Ptr64 Void
+0x2dd8 MaximumDpcQueueDepth : Int4B
+0x2ddc DpcRequestRate   : Uint4B
+0x2de0 MinimumDpcRate   : Uint4B
+0x2de4 DpcLastCount     : Uint4B
+0x2de8 ThreadDpcEnable  : UChar
+0x2de9 QuantumEnd       : UChar
+0x2dea DpcRoutineActive : UChar
+0x2deb IdleSchedule     : UChar
+0x2dec DpcRequestSummary : Int4B
+0x2dec DpcRequestSlot   : [2] Int2B
+0x2dec NormalDpcState   : Int2B
+0x2dee ThreadDpcState   : Int2B
+0x2dec DpcNormalProcessingActive : Pos 0, 1 Bit
+0x2dec DpcNormalProcessingRequested : Pos 1, 1 Bit
+0x2dec DpcNormalThreadSignal : Pos 2, 1 Bit
+0x2dec DpcNormalTimerExpiration : Pos 3, 1 Bit
+0x2dec DpcNormalDpcPresent : Pos 4, 1 Bit
+0x2dec DpcNormalLocalInterrupt : Pos 5, 1 Bit
+0x2dec DpcNormalSpare   : Pos 6, 10 Bits
+0x2dec DpcThreadActive  : Pos 16, 1 Bit
+0x2dec DpcThreadRequested : Pos 17, 1 Bit
+0x2dec DpcThreadSpare   : Pos 18, 14 Bits
+0x2df0 LastTimerHand    : Uint4B
+0x2df4 LastTick         : Uint4B
+0x2df8 ClockInterrupts  : Uint4B
+0x2dfc ReadyScanTick    : Uint4B
+0x2e00 InterruptObject  : [256] Ptr64 Void
+0x3600 TimerTable       : _KTIMER_TABLE
+0x5800 DpcGate          : _KGATE
+0x5818 PrcbPad52        : Ptr64 Void
+0x5820 CallDpc          : _KDPC
+0x5860 ClockKeepAlive   : Int4B
+0x5864 PrcbPad60        : [2] UChar
+0x5866 NmiActive        : Uint2B
+0x5868 DpcWatchdogPeriod : Int4B
+0x586c DpcWatchdogCount : Int4B
+0x5870 KeSpinLockOrdering : Int4B
```

```
+0x5874 PrcbPad70        : [1] Uint4B
+0x5878 CachedPtes       : Ptr64 Void
+0x5880 WaitListHead     : _LIST_ENTRY
+0x5890 WaitLock         : Uint8B
+0x5898 ReadySummary     : Uint4B
+0x589c AffinitizedSelectionMask : Int4B
+0x58a0 QueueIndex       : Uint4B
+0x58a4 PrcbPad75        : [3] Uint4B
+0x58b0 TimerExpirationDpc : _KDPC
+0x58f0 ScbQueue         : _RTL_RB_TREE
+0x5900 DispatcherReadyListHead : [32] _LIST_ENTRY
+0x5b00 InterruptCount   : Uint4B
+0x5b04 KernelTime       : Uint4B
+0x5b08 UserTime         : Uint4B
+0x5b0c DpcTime          : Uint4B
+0x5b10 InterruptTime    : Uint4B
+0x5b14 AdjustDpcThreshold : Uint4B
+0x5b18 DebuggerSavedIRQL : UChar
+0x5b19 GroupSchedulingOverQuota : UChar
+0x5b1a DeepSleep        : UChar
+0x5b1b PrcbPad80        : [5] UChar
+0x5b20 DpcTimeCount     : Uint4B
+0x5b24 DpcTimeLimit     : Uint4B
+0x5b28 PeriodicCount    : Uint4B
+0x5b2c PeriodicBias     : Uint4B
+0x5b30 AvailableTime    : Uint4B
+0x5b34 KeExceptionDispatchCount : Uint4B
+0x5b38 StartCycles      : Uint8B
+0x5b40 TaggedCyclesStart : Uint8B
+0x5b48 TaggedCycles     : [2] Uint8B
+0x5b58 GenerationTarget : Uint8B
+0x5b60 AffinitizedCycles : Uint8B
+0x5b68 PrcbPad81        : [29] Uint4B
+0x5bdc MmSpinLockOrdering : Int4B
+0x5be0 PageColor        : Uint4B
+0x5be4 NodeColor        : Uint4B
+0x5be8 NodeShiftedColor : Uint4B
+0x5bec SecondaryColorMask : Uint4B
+0x5bf0 PrcbPad83        : Uint4B
+0x5bf8 CycleTime        : Uint8B
+0x5c00 Cycles           : [4] [2] Uint8B
+0x5c40 PrcbPad84        : [16] Uint4B
+0x5c80 CcFastMdlReadNoWait : Uint4B
+0x5c84 CcFastMdlReadWait : Uint4B
+0x5c88 CcFastMdlReadNotPossible : Uint4B
+0x5c8c CcMapDataNoWait  : Uint4B
+0x5c90 CcMapDataWait    : Uint4B
+0x5c94 CcPinMappedDataCount : Uint4B
+0x5c98 CcPinReadNoWait  : Uint4B
+0x5c9c CcPinReadWait    : Uint4B
+0x5ca0 CcMdlReadNoWait  : Uint4B
+0x5ca4 CcMdlReadWait    : Uint4B
+0x5ca8 CcLazyWriteHotSpots : Uint4B
+0x5cac CcLazyWriteIos   : Uint4B
+0x5cb0 CcLazyWritePages : Uint4B
+0x5cb4 CcDataFlushes    : Uint4B
+0x5cb8 CcDataPages      : Uint4B
+0x5cbc CcLostDelayedWrites : Uint4B
+0x5cc0 CcFastReadResourceMiss : Uint4B
+0x5cc4 CcCopyReadWaitMiss : Uint4B
+0x5cc8 CcFastMdlReadResourceMiss : Uint4B
+0x5ccc CcMapDataNoWaitMiss : Uint4B
+0x5cd0 CcMapDataWaitMiss : Uint4B
+0x5cd4 CcPinReadNoWaitMiss : Uint4B
+0x5cd8 CcPinReadWaitMiss : Uint4B
```

```
+0x5cdc CcMdlReadNoWaitMiss : Uint4B
+0x5ce0 CcMdlReadWaitMiss : Uint4B
+0x5ce4 CcReadAheadIos   : Uint4B
+0x5ce8 MmCacheTransitionCount : Int4B
+0x5cec MmCacheReadCount : Int4B
+0x5cf0 MmCacheIoCount   : Int4B
+0x5cf4 PrcbPad91        : [3] Uint4B
+0x5d00 PowerState       : _PROCESSOR_POWER_STATE
+0x5ed0 ScbList          : _LIST_ENTRY
+0x5ee0 PrcbPad92        : [7] Uint4B
+0x5efc KeAlignmentFixupCount : Uint4B
+0x5f00 DpcWatchdogDpc   : _KDPC
+0x5f40 DpcWatchdogTimer : _KTIMER
+0x5f80 Cache            : [5] _CACHE_DESCRIPTOR
+0x5fbc CacheCount       : Uint4B
+0x5fc0 CachedCommit     : Uint4B
+0x5fc4 CachedResidentAvailable : Uint4B
+0x5fc8 HyperPte         : Ptr64 Void
+0x5fd0 WheaInfo         : Ptr64 Void
+0x5fd8 EtwSupport       : Ptr64 Void
+0x5fe0 InterruptObjectPool : _SLIST_HEADER
+0x5ff0 HypercallPageList : _SLIST_HEADER
+0x6000 HypercallCachedPages : Ptr64 Void
+0x6008 VirtualApicAssist : Ptr64 Void
+0x6010 StatisticsPage   : Ptr64 Uint8B
+0x6018 PackageProcessorSet : _KAFFINITY_EX
+0x60c0 SharedReadyQueueMask : Uint8B
+0x60c8 SharedReadyQueue : Ptr64 _KSHARED_READY_QUEUE
+0x60d0 SharedQueueScanOwner : Uint4B
+0x60d4 ScanSiblingIndex : Uint4B
+0x60d8 CoreProcessorSet : Uint8B
+0x60e0 ScanSiblingMask  : Uint8B
+0x60e8 LLCMask          : Uint8B
+0x60f0 CacheProcessorMask : [5] Uint8B
+0x6118 ProcessorProfileControlArea : Ptr64 _PROCESSOR_PROFILE_CONTROL_AREA
+0x6120 ProfileEventIndexAddress : Ptr64 Void
+0x6128 PrcbPad94        : [11] Uint8B
+0x6180 SynchCounters    : _SYNCH_COUNTERS
+0x6238 PteBitCache      : Uint8B
+0x6240 PteBitOffset     : Uint4B
+0x6248 FsCounters       : _FILESYSTEM_DISK_COUNTERS
+0x6258 VendorString     : [13] UChar
+0x6265 PrcbPad100       : [3] UChar
+0x6268 FeatureBits      : Uint8B
+0x6270 PrcbPad110       : Uint4B
+0x6278 UpdateSignature  : _LARGE_INTEGER
+0x6280 Context          : Ptr64 _CONTEXT
+0x6288 ContextFlagsInit : Uint4B
+0x6290 ExtendedState    : Ptr64 _XSAVE_AREA
+0x6298 IsrStack         : Ptr64 Void
+0x62a0 EntropyTimingState : _KENTROPY_TIMING_STATE
+0x63f0 AbSelfIoBoostsList : _SINGLE_LIST_ENTRY
+0x63f8 AbPropagateBoostsList : _SINGLE_LIST_ENTRY
+0x6400 AbDpc            : _KDPC
+0x6440 IoIrpStackProfilerCurrent : _IOP_IRP_STACK_PROFILER
+0x6494 IoIrpStackProfilerPrevious : _IOP_IRP_STACK_PROFILER
+0x6500 LocalSharedReadyQueue : _KSHARED_READY_QUEUE
+0x6760 TimerExpirationTrace : [16] _KTIMER_EXPIRATION_TRACE
+0x6860 TimerExpirationTraceCount : Uint4B
+0x6868 ExSaPageArray    : Ptr64 Void
+0x6880 Mailbox          : Ptr64 _REQUEST_MAILBOX
+0x68c0 RequestMailbox   : [1] _REQUEST_MAILBOX
```

- **Seuil d'ajustement DPC (AdjustDpcThreshold)** Indicateur dynamique sur lequel Windows s'appuie pour contrôler la fréquence de traitement des appels de procédure différée (DPC, *Deferred Procedure Call*).

Lorsque l'activité DPC est élevée (nombre important de DPC en attente ou durées d'exécution prolongées), ce seuil est pour accélérer leur traitement revue à la baisse. À l'inverse, en période de faible activité, il peut être augmenté en vue d'améliorer les performances globales (minimisation des basculements de contexte). Voir service NtQuerySystemInformation (SystemInformationClass = SystemDpcInformation) ; variable noyau KiAdjustDpcThreshold.

- *BuildType* Voir constantes PRCB_BUILD_DEBUG et PRCB_BUILD_UNIPROCESSOR.
- CcCopyReadNoWait Total des copies asynchrones dans le cache. Voir routine CcCopyRead.
- CcCopyReadNoWaitMiss Total des copies asynchrones dans le cache n'ayant pu être satisfaites Voir routine CcCopyRead.
- CcCopyReadWait Total des copies synchrones dans le cache.
- CcCopyReadWaitMiss Total des copies synchrones dans le cache n'ayant pu être satisfaites .
- *Nombre de vidanges (CcDataFlushes)* Nombre de fois que des pages du cache ont été vidangés explicitement, pour cause d'écritures immédiates ou d'écrivain paresseux. Voir structure de données SYSTEM_PERFORMANCE_INFORMATION, attribut DataFlushes ; voir compteur de performances Cache: Vidange de données/s.
- *Nombre de pages vidangées (CcDataPages)* Nombre de pages vidangées explicitement, pour cause d'écritures immédiates ou d'écrivain paresseux. Voir structure de données SYSTEM_PERFORMANCE_INFORMATION, attribut DataPages ; Voir compteur de performances Vidange de données, page/s.
- *CcLazyWriteIos* Nombre de vidanges faites par l'écrivain paresseux. Voir routine CcFlushCache ; structure SYSTEM_PERFORMANCE_INFORMATION (attribut CcLazyWriteIos).
- *CcLazyWritePages* Nombre de pages écrites par l'écrivain paresseux. Voir routine CcFlushCache ; structure SYSTEM_PERFORMANCE_INFORMATION (attribut CcLazyWritePages).
- *CcFastReadNoWait* Nombre de lectures rapides asynchrones ayant été traitées en tant que requêtes Fast I/O.
- *CcFastReadWait* Nombre de lectures rapides synchrones ayant été traitées en tant que requêtes Fast I/O.
- *CcFastReadNotPossible* Nombre de lectures rapides n'ayant pu être satisfaites. Voir attribut CcFastReadNotPossible de structure SYSTEM_PERFORMANCE_INFORMATION ; routine FsRtlIncrementCcFastReadNotPossible.
- *Lectures MDL asynchrones (CcMdlReadNoWait)* Total des lectures MDL asynchrones dans le cache.
- *CcMdlReadNoWaitMiss* Total des lectures MDL asynchrones dans le cache n'ayant pu être satisfaites.
- *Lectures MDL synchrones (CcMdlReadWait)* Total des lectures MDL synchrones dans le cache.
- *CcMdlReadWaitMiss* Total des lectures MDL synchrones dans le cache n'ayant pu être satisfaites.
- *Contexte de thread (Context)* Pointeur vers le contexte matériel (spécifique à l'architecture processeur) du thread.
- *Disponibilité CPUID (CpuID)* Valeur booléenne indiquant la présence de l'instruction processeur CPUID.
- **Fabricant du processeur (CpuVendor)** Indique de quel fabricant est issu un processeur, par exemple Intel, AMD, etc. Une telle information est utilisée en interne pour optimiser certaines décisions dépendantes du constructeur, telles que le contournement de bogues ou l'interprétation spécifique de valeurs inhérentes au matériel.
- **Modèle de processeur (CpuType)** Valeur numérique indiquant la classe générale à laquelle se rattache le processeur ; information dont se sert le noyau pour ajuster ses optimisations ou appliquer des comportements

spécifiques, par exemple contourner des bogues matériels connus. Un tel attribut est souvent combiné avec d'autres, comme CpuStep ou VendorString, pour affiner l'identification.

- *Thread courant (CurrentThread)* Pointeur vers le bloc KTHREAD du thread que ce processeur est en train d'exécuter.
- *CpuID* Indique la disponibilité de l'instruction CPUID, laquelle interroge le processeur pour obtenir des informations sur les fonctionnalités prises en charge.
- **Veille approfondie (DeepSleep)** Détermine si le processeur est autorisé à entrer dans des états de veille très profonds (ACPI C6 et supérieurs). Le système peut de la sorte réduire significativement la consommation d'énergie en période d'inactivité, au prix d'une latence accrue au réveil. Windows ajuste dynamiquement cette autorisation en fonction la stratégie d'alimentation active, et des contraintes de performance qui y sont associées.
- **File des prêts différés (DeferredReadyListHead)** Liste chaînée utilisée pour mémoriser temporairement des threads prêts à s'exécuter, mais dont l'insertion dans la file d'attente de prêts principale (attribut `DispatcherReadyListHead`) a été remise à plus tard afin d'améliorer les performances, évitant de la sorte de coûteuses opérations de synchronisation.
- **IRQL sauvegardé (DebuggerSavedIRQL)** Valeur de l'IRQL du processeur au moment d'un effondrement du système. Cette information est enregistrée en premier lieu afin de faciliter une analyse post mortem, notamment en fournissant des indices sur le contexte d'exécution tel qu'il se présentait lors de la défaillance. Voir routine KeBugCheck.
- *File DPC (DpcData)* File contenant les DPC en attente d'exécution.
- *Pile DPC (DpcStack)* Tampon mémoire utilisé comme moyen de stockage temporaire au cours de l'exécution d'un DPC.
- *Files des prêts (DispatcherReadyListHead)* Tableau de 32 entrées de liste qui regroupe les têtes de liste des files d'attente des prêts.
- *Taux de demande DPC (DpcRequestRate)* Nombre de DPC qui ont été ajoutés à la file DPC du processeur depuis le dernier tick horloge. Voir service NtQuerySystemInformation (SystemInformationClass = SystemProcessorStatistics).
- **Exécution DPC immédiate (DpcRoutineActive)** Valeur booléenne indiquant si le processeur est actuellement en train de traiter un appel de procédure différée (DPC), généralement à un niveau d'interruption élevé (`DISPATCH_LEVEL` ou supérieur), synonyme de courte durée et de forte contrainte temporelle. Une telle information permet à diverses sous-parties de Windows, pilotes et autres codes système, d'adapter leur comportement en fonction du contexte d'exécution, par exemple en évitant les synchronisations bloquantes ou les attentes prolongées. Voir routine KeIsExecutingDpc.
- **Exécution DPC différée (DpcThreadActive)** Valeur booléenne indiquant si le thread dédié au traitement des DPC différés est actuellement actif sur le processeur, offrant une gestion plus souple et moins urgente des tâches. Contrairement aux DPC traités immédiatement après une interruption, certains DPC sont planifiés pour une exécution différée à un niveau d'interruption plus bas (`PASSIVE_LEVEL` ou `DISPATCH_LEVEL`) via un thread spécifique. Ce champ permet de savoir si un tel traitement est mis en oeuvre.
- *DpcThreadRequested* Voir routine KeInsertQueueDpc.
- **Temps DPC (DpcTime)** Temps total passé par le processeur à exécuter des appels de procédure différée (DPC).
- **Masque de fonctionnalités (FeatureBits**) Masque binaire représentant les fonctionnalités spéciales dont dispose le processeur, telles que les jeux d'instructions MMX ou SSE, les capacités de virtualisation (Intel VT-x, AMD-V), ou encore les extensions de sécurité (par exemple, AES-NI). Voir fonction IsProcessorFeaturePresent ; service NtQuerySystemInformation (SystemInformationClass = SystemProcessorInformation) ; variable noyau

KeFeatureBits ; attribut ProcessorFeatures de structure KUSER_SHARED_DATA, attribut ProcessorFeatureBits de structure SYSTEM_PROCESSOR_INFORMATION.

- **Groupe de processeur (Group)** Dans un système multiprocesseur, désigne le groupe auquel un processeur est affecté. Les groupes de processeurs permettent de mieux organiser et gérer les ressources processeurs, notamment pour dans une telle configuration optimiser la gestion de la charge de travail et les stratégies d'ordonnancement.

- *IdleHalt* Valeur booléenne qui, du moment qu'elle est armée, indique que le processeur peut passer en état de veille (aucun thread n'a été sélectionné pour l'exécution).

- **Thread inactif (IdleThread)** Pointeur vers le bloc KTHREAD sous-jacent au thread inactif, responsable de l'exécution des tâches d'arrière-plan tandis que le système ne dispose d'aucun autre thread paré pour l'exécution.

- **Compteur d'interruptions (InterruptCount)** Nombre de fois qu'un processeur a répondu a une interruption depuis le démarrage du système. Voir fonction `NtQuerySystemInformationEx` ; structure `SYSTEM_PROCESSOR_PERFORMANCE_INFORMATION`.

- **Objet interruption (InterruptObject)** Pointeur vers l'objet interruption auquel un processeur est associé. Voir routine KiGetInterruptObjectAddress.

- *Compteur d'E/S lecture (IoReadOperationCount)* Cette valeur est actualisée (comprendre incrémentée) chaque fois que le service système NtReadFile a été appelé.

- *IoWriteOperationCount* Cette valeur est actualisée (comprendre incrémentée) chaque fois que le service système NtWriteFile a été appelé.

- **Cumul des changements de contexte (KeContextSwitches)** Nombre total de commutations de contexte effectuées par un processeur depuis le démarrage. Ce compteur est actualisé chaque fois qu'un changement d'exécution entre deux threads a lieu sur le processeur concerné, qu'il soit de manière volontaire (par exemple, entrée dans un état d'attente) ou préemptive (expiration de quantum, arrivée d'un thread de priorité supérieure, etc.).

- **Cumul des appels système (KeSystemCalls)** Nombre d'appels système traités par un processeur depuis le démarrage. Ce compteur est actualisé à chaque transition explicite du mode utilisateur vers le mode noyau, déclenchée par une instruction d'appel système telle que `SYSENTER` ou `SYSCALL`.

- **Fréquence du processeur (MHz)** Fréquence en mégahertz du processeur, telle qu'estimée au démarrage du système sur la base de mesures internes (comme le compteur TSC) ou d'informations fournies par le micro-logiciel, les registres spécialisés (CPUID) ou de contrôle (MSR). Ce chiffre reflète la fréquence nominale attendue du processeur, mais pas nécessairement sa fréquence instantanée, celle-ci étant susceptible de varier selon la charge ou les mécanismes d'ajustement de la fréquence d'horloge, comme EIST (Enhanced Intel SpeedStep Technology). Voir valeur \~MHz sous la clé HKLM\HARDWARE\DESCRIPTION \System\CentralProcessor\0.

- *Version majeure (MajorVersion)* Numéro interne utilisé en tant en tant qu'information de version, ici un numéro de version majeur, dont la valeur est calquée directement sur celle de la constante PRCB_MAJOR_VERSION, définie à 1.

- **Profondeur de file DPC (MaximumDpcQueueDepth)** Nombre maximal d'éléments que le file DPC d'un processeur est susceptible de contenir. Une telle limite est en interne employée dans l'optique de réguler l'accumulation des DPC en attente d'exécution, afin d'éviter une surcharge du processeur par des routines différées. Voir service NtQuerySystemInformation (SystemInformationClass = SystemDpcInformation) ; variable noyau KiMaximumDpcQueueDepth.

- **Seuil minimal d'activité DPC (MinimumDpcRate)** Voir service NtQuerySystemInformation (SystemInformationClass = SystemDpcInformation) ; variable noyau KiMinimumDpcRate.

- *Version mineure (MinorVersion)* Numéro interne utilisé en tant en tant qu'information de version, ici un numéro de version mineur, dont la valeur est calquée directement sur celle de la constante PRCB_MINOR_VERSION, définie à 1.

- **Compteur des E/S en cache (MmCacheIoCount)** Comptabilise les opérations d'E/S, telles que les lectures et écritures, satisfaites à partir du cache du système de fichiers, excluant ainsi les accès au disque. Voir structure SYSTEM_PERFORMANCE_INFORMATION (attribut CacheIoCount)

- **Compteur des lectures en cache (MmCacheReadCount)** Comptabilise le nombre d'E/S lecture satisfaites à partir du cache du système de fichiers. Voir structure SYSTEM_PERFORMANCE_INFORMATION (attribut CacheReadCount)

- **Compteur des transitions en cache (MmCacheTransitionCount)** Comptabilise le nombre d'E/S lecture satisfaites à partir du cache du système de fichiers, mais ayant exigé une transition d'état de page en mémoire (et, partant, une mise à jour des métadonnées concernées) avant l'accès effectif aux données. Voir structure SYSTEM_PERFORMANCE_INFORMATION (attribut CacheTransitionCount).

- **Compteur de copies sur écriture (MmCopyOnWriteCount)** Nombre de fois que le noyau a effectué une copie d'une page partagée suite à une tentative d'écriture.

- **Compteur de pages de zéros (MmDemandZeroCount)** Nombre de demandes de page à zéro que le processeur a satisfait. Voir service NtQuerySystemInformation (SystemInformationClass = SystemPerformanceInformation) ; structure SYSTEM_PERFORMANCE_INFORMATION (attribut DemandZeroFaults).

- **Compteur de pages modifiées (MmDirtyPagesWriteCount)** Nombre total de pages modifiées, issues notamment du fichier d'échange ou de fichiers (mappés) en mémoire, qui ont été écrites - et ce faisant sauvegardées - sur disque depuis un processeur.

- **Compteur d'E/S de pages modifiées (MmDirtyWriteIoCount)** Nombre total d'opérations d'E/S initiées en vue de remiser les pages mémoire ayant fait l'objet de modification depuis leur dernière sauvegarde sur disque.

- **Compteur de fautes de page (MmPageFaultCount)** Nombre total de fautes de page qu'un processeur a traité jusqu'à présent, toutes natures confondues.

- **Valeur MXCSR (MxCsr)** Duplicata de la valeur du registre matériel MXSCR, qui contrôle les opérations SIMD.

- **Prochain thread (NextThread)** Pointeur vers le bloc KTHREAD sous-jacent au thread en faveur duquel devrait avoir lieu le prochain basculement de contexte.

- **État processeur (ProcessorState)** Valeurs des registres généraux (encapsulées dans une structure `CONTEXT`) et spéciaux (structure `KSPECIAL_REGISTERS`) du processeur, dont l'ensemble est sauvegardé lors de transitions système critiques, telles que l'hibernation, certains arrêts contrôlés ou passages en veille profonds. À la reprise, l'état sauvegardé assure une restauration fidèle de l'exécution du processeur.

- *Résumé des prêts (ReadySummary)* Masque binaire des niveaux de priorité pour lesquels il existe un ou plusieurs threads prêt.

- *Autorisation des DPC threadés (ThreadDpcEnable)* Indique si oui ou non le processeur honore les DPC threadés.

- **Séquence d'identifiant du fabricant (VendorString)** Séquence de 12 caractères ASCII renvoyée par le processeur suite à l'exécution de l'instruction `CPUID`, permettant d'identifier le fabricant du processeur, par exemple Intel (GenuineIntel) ou AMD (AuthenticAMD). Windows emploie cette information pour initialiser certains champs internes ou opter pour des optimisations spécifiques. Voir routine KiGetCpuVendor.

- *Informations WHEA (WheaInfo)* Informations concernant Windows Hardware Error Architecture (WHEA).

Vous pouvez voir le contenu du bloc de contrôle de chaque processeur par le biais de la commande !pcr du débogueur noyau. Sans paramètres pour l'accompagner, la commande montre le PCR du premier processeur du système (numéroté 0).

```
lkd> !pcr
KPCR for Processor 0 at fffff803703ce000:
    Major 1 Minor 1
 NtTib.ExceptionList: fffff803721e7000
     NtTib.StackBase: fffff803721e8070
    NtTib.StackLimit: 000000000302f318
  NtTib.SubSystemTib: fffff803703ce000
       NtTib.Version: 00000000703ce180
   NtTib.UserPointer: fffff803703ce7f0
       NtTib.SelfTib: 000000df35234000

             SelfPcr: 0000000000000000
                Prcb: fffff803703ce180
                Irql: 0000000000000000
                 IRR: 0000000000000000
                 IDR: 0000000000000000
       InterruptMode: 0000000000000000
                 IDT: 0000000000000000
                 GDT: 0000000000000000
                 TSS: 0000000000000000

       CurrentThread: ffffe185701b2740
          NextThread: 0000000000000000
          IdleThread: fffff80370448940
```

Variables noyau

Le tableau suivant énumère quelques unes des variables globales système établies conjointement aux threads.

Tableau 8.27. Variable noyau associées aux threads

Variable	Description
PspCreateThreadNotifyRoutine	Tableau de pointeurs vers des routines à appeler lors de la création/suppression de thread.
PspCreateThreadNotifyRoutineCount	Nombre de routines de notification de thread déclarées.
PsThreadType	Attributs génériques des objets de type thread.

Compteurs de performance des threads

La liste qui suit énumère les compteurs associés aux threads.

- *Processus: Priorité de base* Priorité de base actuelle du processus, laquelle sert de priorité de départ pour les threads créées dans ce processus.
- *Thread: % Temps privilégié* Pourcentage de temps qu'un thread a passé en mode noyau.
- *Thread: % Temps processeur* Pourcentage de temps processeur qu'un thread a consommé ; égal à la somme de % Temps privilégié et % Temps utilisateur.
- *Thread: % Temps utilisateur* Pourcentage de temps qu'un thread a passé en mode utilisateur.
- *Thread: Changements de contexte/s* Nombre de basculements de contexte qui ont eu lieu en faveur d'un thread.
- *Thread: Temps total écoulé* Temps processeur (en secondes) qu'un thread a consommé.
- *Thread: No du processus* ID du processus dans lequel un thread s'exécute.
- *Thread: No du thread* ID du thread.

- *Thread: Priorité de base* Priorité de base actuelle du thread.
- *Thread: Priorité actuelle* Priorité dynamique actuelle du thread.
- *Thread: État du thread* Contient une valeur, comprise entre 0 et 7, indiquant l'état actuel du thread.
- *Thread: Raison d'attente du thread* Contient une valeur, comprise entre 0 et 19, indiquant la raison pour laquelle un thread se trouve en état d'attente.

Restrictions concernant les threads des processus protégées

Le tableau suivant monte les restrictions imposées par des threads protégés.

Droit d'accès	Utilité
THREAD_ALL_ACCESS	Empêche l'accès complet au thread protégé.
THREAD_DIRECT_IMPERSONATION	Empêche un processus d'emprunter l'identité du thread protégé.
THREAD_GET_CONTEXT	Empêche l'accès au contexte processeur (registres) et au contexte mémoire (piles) du thread protégé.
THREAD_QUERY_INFORMATION	Empêche un processus d'obtenir des informations concernant un thread protégé.
THREAD_SET_CONTEXT	Empêche la modification du contexte processeur (registres) et du contexte mémoire (piles) du thread protégé.
THREAD_SET_INFORMATION	Empêche un processus de modifier les informations concernant un thread protégé.
THREAD_SET_TOKEN	Empêche un thread de redéfinir le jeton de sécurité d'un thread protégé.
THREAD_TERMINATE	Empêche de mettre fin au thread protégé.

Thread courant

Le thread en cours d'exécution sur un processeur est dit *courant*.

Identifiant de thread

A l'instar d'un processus, tout thread se voit lors de sa naissance attribuer un code unique, appelé identifiant de thread (TID, Thread Identifier), qui le distingue de tous ses semblables.

Le système génère un ID pour un nouveau thread durant la création du bloc ETHEAD idoine, laquelle a lieu dans la fonction interne PspCreateThread, sollicitée depuis le service système NtCreateThread et la routine PsCreateSystemThread.

Compte tenu des principes qui entourent la rétention des objets, l'ID d'un thread peut perdurer au delà de son exécution. En l'occurence, tant qu'une référence s'applique à un thread, l'ID concerné est valide et garanti unique. A titre informatif, notez qu'un ID de thread ne peut avoir une valeur nulle.

Un thread peut questionner l'identité d'un autre par le biais de la fonction GetThreadId, qui retourne l'ID du thread donné en paramètre. Vu que l'opération concerne un contexte d'exécution différent de l'actuel, il est dans cette configuration nécessaire d'exécuter un service système, ici NtQueryInformationThread, qui exige les droits THREAD_QUERY_INFORMATION ou THREAD_QUERY_LIMITED_INFORMATION.

Tableau 8.28. Fonctions Windows concernant les identifiants de thread

Fonction	Description	Autres références
GetCurrentThreadId	Retourne l'ID du thread courant	TEB Cid
GetThreadId	Retourne l'ID du thread spécifié	NtQueryInformationThread

Tableau 8.29. Routines concernant les identifiants de thread

Routine	Description	Autres références

PsGetCurrentThreadId	Retourne l'ID du thread courant	ETHREAD Cid.UniqueThread
PsGetThreadId	Retourne l'ID du thread spécifié	ETHREAD Cid.UniqueThread

Terminaison de threads

Conditionné le long de l'existence par des facteurs de nature soit intrinsèque (issus des instructions principales du thread) soit extrinsèque (provenant d'autres threads du processus ou de processus externes), tout thread doit à un moment ou un autre prendre fin. La terminaison d'un thread survient en résultat de n'importe quel des phénomènes suivants :

- **Fin de routine principale** Un thread prend fin implicitement après que sa routine principale ai rendu la main, ce qui à bien des égards est la méthode la plus commode pour terminer un thread. Un processus peut déterminer le moment où s'est achevé une routine principale en utilisant l'une des fonctions Windows d'attente. Il peut également questionner la façon dont elle s'est achevé, ceci au moyen de la fonction GetExitCodeThread.
- **Exceptions non traitées** Une exceptions non traitée entraine dans la plupart des cas la mise en fin de vie du processus duquel l'exception est issue. En la matière, le système d'exploitation implémente un gestionnaire d'execption conçu pour ne laisser aucun code se poursuivre en une situation d'erreur, ce qui donnerait des résultats imprévisibles. Le comportement classique de ce gestionnaire est, via l'affichage d'une boite de dialogue, de donner le choix à l'utilisateur de déboguer ou de terminer l'application fautive.
- **Fin de thread** Il est possible d'appeler explicitement la fonction ExitThread dans une routine de thread pour mettre fin au traitement sans attendre la fin de la fonction principale. ExitThread termine normalement un thread et notifie toutes les DLL associées avec une valeur indiquant que le thread courant se détache, puis désaloue la pile du thread courant pour mettre fin au thread courant. Le processus d'application se termine si le thread courant est le thread principal. Comme ExitThread agit sur le thread courant, il n'est pas nécessaire de spécifier un handle de thread. Une méthode alternative consiste à arrêter un thread spécifié sans exécuter de code de nettoyage, chose réalisée dans la fonction TerminateThread.
- **Fin de processus** Lorsqu'un processus se termine, le système - entre autre opération - met fin à l'ensemble des threads auxquels ce processus fournit les ressources (lesquelles ressources sont pour l'occasion aussi démantelées). Comme avec les threads, l'API Windows offre deux méthodes pour supprimer un processus, l'une avec nettoyage et déchargement de DLL, ExitProcess, et l'autre sans, TerminateProcess.

Quand un thread est supprimé, les ressources dont il seul détenteur sont libérées, son code de fin est défini à une valeur adéquate et, pour finir, l'objet thread sous-jacent est mis à l'état signalé. Si le thread est le dernier actif dans le processus, ce dernier est terminé.

Annulation de threads

L'annulation d'un thread est le procédé consistant à mettre prématurément un terme à l'exécution d'un thread. Dans des circonstances ordinaires, un thread se termine soit en sortie de sa fonction principale soit en appelant la fonction ExitThread, laquelle met fin à l'appelant. Il est en plus de cela possible pour un thread de solliciter le système en vue de terminer un autre thread, cela par le biais de la fonction TerminateThread.

TerminateThread attend en entrée deux paramètres : le handle du thread à interrompre et un code de sortie, récupérable ultérieurement à l'aide de la fonction GetExitCodeThread. En interne, TerminateThread se base sur le service système NtTerminateThread, lequel s'appuie lui-même sur PspTerminateThreadByPointer.

L'une des caractéristiques les plus distinctives de TerminateThread est de déboucher sur un arrêt brutal du thread cible. De ce fait, un certain nombre d'opérations de nettoyage communément effectuées lors de la suppression de threads n'ont dans ce contexte pas lieu, par exemple l'envoi de messages de notification aux DLL de sous-système. Les éventuelles conséquences de l'emploi de TerminateThread incluent la non-libération des ressources allouées, le laisser à l'abandon des sections critiques et des mutexs appartenant au thread interrompu, et d'autres. Pour toutes ces raisons, TerminateThread ne devrait jamais faire partie du cheminement normal d'une application.

Création de threads

La liste qui suit décrit les grandes étapes de la création d'un thread, incarnée en l'occurence par la fonction la fonction Windows CreateThread.

1. Création d'une pile mode utilisateur pour le thread dans l'espace d'adressage du processus dans lequel il s'exécute.

2. Initialisation du contexte matériel du thread.

3. Appel du service système NtCreateThread.

4. Notification au sous-système Windows de l'existence d'un nouveau thread.

5. Génération de l'ID du thread.

La fonction CreateThread, employée pour donner lieu à un nouveau thread, attend plusieurs paramètres qui contrôlent la manière dont le système crée le thread et les instructions que le thread exécute.

- Le premier argument correspond aux attributs dont l'appelant désire doter le nouveau thread en matière de sécurité et d'héritage, à savoir à quel profil de sécurité l'associer, et si un processus fils peut hériter du handle retourné. Si ce pointeur est null, aucun héritage n'est possible, et la politique de sécurité associée correspond à celle définie par défaut pour le processus appelant.

- Le second argument indique la taille de la pile mode utilisateur pour le thread dans l'espace d'adressage du processus. Si cette valeur est nulle, la taille associée est celle définie par défaut pour le programme.

- Le troisième argument est un pointeur représentant la fonction principale du nouveau thread. Cette fonction, qui définit généralement les principales instructions de traitement pour le thread, est invoquée dès la création du thread et reçoit en paramètre le pointeur passé en quatrième argument de CreateThread. Il s'agit d'un pointeur car le type de la variable est inconnu au moment de la création du thread.

- Le cinquième paramètre permet d'indiquer le mode d'exécution du thread. A moins que l'appelant n'ait créé le thread avec le flag CREATE_SUSPENDED, le thread peut immédiatement être ordonnancé pour l'exécution. Autrement, il ne pourra l'être qu'à condition d'être sujet de la fonction ResumeThread, laquelle permet de reprendre l'exécution d'un thread suspendu.

- Le sixième et dernier paramètre est un pointeur vers une variable appelée à accueillir éventuellement l'ID du thread nouvellement créé. Cet ID n'existe que pendant la durée de vie du thread, vu que les ID des threads (comme ceux des processus) sont réutilisés.

Toute création de threads peut pour un certain nombre de raisons subir un revers, auquel cas la valeur retournée par la fonction CreateThread est « null". La fonction GetLastError peut être utilisé pour recueillir des information sur ce manquement aux résultats attendus. Notez que, au sens des ressources mobilisables, chaque nouveau thread consomme une part non négligeable de mémoire, incluant une certaine quantité de celle non paginée. Par conséquent, si le système affronte une situation de ressources faibles, il est très probable que créer un thread se solde par un échec. D'usage, on considère que toute application devrait savoir gérer de tels scénarios, et inclure à cet effet des chemins alternatifs dans le code.

Création de threads système . La naissance de threads fonctionnant uniquement en mode noyau (threads système) est le fait de la routine PsCreateSystemThread, laquelle crée un thread distinct exécutant du code chargé dans l'espace système. Composants et pilotes de périphériques sollicitent cette fonction soit lors de l'initialisation, soit quand des requêtes commencent à alimenter les mécanismes de réponse à des d'E/S (routine de diffusion du pilote par exemple).

Par défaut un thread système est la propriété du processus système initial. Un tel thread continue de fonctionner jusqu'à ce que le système soit cible d'une procédure d'extinction, ou jusqu'à ce que le thread demande sa propre fin avec PsTerminateSystemThread.

Un pilote peut s'il le désire créer un thread système dans n'importe quel processus. Les routines de pilotes s'exécutant dans un contexte différent du processus système initial doivent toutefois veiller à ne pas exposer des ressources au code mode utilisateur, et à cet effet restreindre l'utilisation du handle retournée par PsCreateSystemThread aux seuls processus s'exécutant en mode noyau. Si cela n'est pas fait, le handle reste accessible aux threads du processus dans lequel le pilote est en cours d'exécution.

Sécurité des threads

La tableau qui suit énumère les droits d'accès concernant les threads.

Tableau 8.30. Droits d'accès concernant les threads

Constante	Valeur	Description
THREAD_GET_CONTEXT	0x0008	Requis pour obtenir les valeurs de contexte d'un thread. Voir service NtGetContextThread.
THREAD_ALL_ACCESS	-	Cumule l'ensemble des droits d'accès génériques et des droits d'accès spécifiques aux threads.
THREAD_SET_INFORMATION	0x0020	Requis pour définir certaines informations relatives à un thread.
THREAD_SET_CONTEXT	0x0010	Voir services NtQueueApcThread et NtSetContextThread.
THREAD_ALERT	0x0004	Requis pour alerter un thread. Voir service NtAlertThread.
THREAD_IMPERSONATE	0x0100	Requis pour emprunter l'identité d'un thread. Voir service NtImpersonateThread.
THREAD_QUERY_INFORMATION	0x0040	Requis pour obtenir des informations sur un thread. Voir service NtQueryInformationToken.
THREAD_SET_THREAD_TOKEN	0x0080	Requis pour associer un jeton d'emprunt d'identité à un thread. Controlé dans NtSetInformationThread. Voir fonction SetThreadToken.
THREAD_SUSPEND_RESUME	0x0002	Requis pour suspendre un thread ou reprendre l'exécution d'un thread suspendu. Voir services NtAlertResumeThread, NtResumeThread et NtSuspendThread.
THREAD_TERMINATE	0x0001	Requis pour mettre fin à un thread.- Controlé par NtTerminateThread.

THREADINFOCLASS

Tableau 8.31. THREADINFOCLASS

Nom	Valeur	Type de données associé
ThreadBasicInformation	00	THREAD_BASIC_INFORMATION
ThreadTimes	01	KERNEL_USER_TIMES
ThreadPriority	02	KPRIORITY
ThreadBasePriority	03	LONG
ThreadAffinityMask	04	KAFFINITY
ThreadImpersonationToken	05	HANDLE
ThreadDescriptorTableEntry	06	DESCRIPTOR_TABLE_ENTRY
ThreadEnableAlignmentFaultFixup	07	BOOLEAN
ThreadEventPair	08	

Nom	Valeur	Type de données associé
ThreadQuerySetWin32StartAddress	9	PVOID
ThreadZeroTlsCell	10	
ThreadPerformanceCount	11	LARGE_INTEGER
ThreadAmILastThread	12	ULONG
ThreadIdealProcessor	13	ULONG
ThreadPriorityBoost	14	ULONG
ThreadSetTlsArrayAddress	15	
ThreadIsIoPending	16	ULONG
ThreadHideFromDebugger	17	
ThreadBreakOnTermination	18	ULONG
ThreadSwitchLegacyState	19	
ThreadIsTerminated	20	ULONG
ThreadLastSystemCall	21	THREAD_LAST_SYSCALL_INFORMATION
ThreadIoPriority	22	
ThreadCycleTime	23	THREAD_CYCLE_TIME_INFORMATION
ThreadPagePriority	24	ULONG
ThreadActualBasePriority	25	
ThreadTebInformation	26	THREAD_TEB_INFORMATION
ThreadCSwitchMon	27	
ThreadCSwitchPmu	28	
ThreadWow64Context	29	WOW64_CONTEXT
ThreadGroupInformation	30	GROUP_AFFINITY
ThreadUmsInformation	31	THREAD_UMS_INFORMATION
ThreadCounterProfiling	32	
ThreadIdealProcessorEx	33	PROCESSOR_NUMBER
ThreadCpuAccountingInformation	34	
ThreadSuspendCount	35	
ThreadActualGroupAffinity	41	
ThreadDynamicCodePolicyInfo	42	
ThreadExplicitCaseSensitivity	43	
ThreadWorkOnBehalfTicket	44	
ThreadSubsystemInformation	45	SUBSYSTEM_INFORMATION_TYPE
ThreadDbgkWerReportActive	46	
ThreadAttachContainer	47	
ThreadManageWritesToExecutableMemory	48	MANAGE_WRITES_TO_EXECUTABLE_MEMORY
ThreadPowerThrottlingState	49	THREAD_POWER_THROTTLING_STATE

THREAD_BASIC_INFORMATION.

```
typedef struct _THREAD_BASIC_INFORMATION
{
    NTSTATUS ExitStatus;
    PTEB TebBaseAddress;
    CLIENT_ID ClientId;
    ULONG_PTR AffinityMask;
    KPRIORITY Priority;
    LONG BasePriority;
} THREAD_BASIC_INFORMATION, *PTHREAD_BASIC_INFORMATION;
```

- **ID Client (*ClientId*)** Voir fonctions GetProcessIdOfThread et GetThreadId.
- **Code de fin (*ExitStatus*)** Voir fonctions GetExitCodeThread.

- **Priorité d'ordonnancement (*Priority*)** Voir fonction GetThreadPriority.

Tableau 8.32. Valeurs initiales des champs de la structure THREAD_BASIC_INFORMATION

Champ	Valeur initiale
AffinityMask	KTHREAD Affinity
BasePriority	KTHREAD BasePriority
ClientId	ETHREAD Cid
ExitStatus	ETHREAD ExitStatus
Priority	KTHREAD Priority
TebBaseAddress	KTHREAD Teb

Tableau 8.33. Valeurs initiales des champs pour la classe ThreadTimes

Champ	**Valeur initiale**
CreateTime	ETHREAD CreateTime
ExitTime	ETHREAD ExitTime
KernelTime	KTHREAD KernelTime
UserTime	KTHREAD UserTime

THREAD_LAST_SYSCALL_INFORMATION.

```
typedef struct _THREAD_LAST_SYSCALL_INFORMATION
{
    PVOID FirstArgument;
    USHORT SystemCallNumber;
} THREAD_LAST_SYSCALL_INFORMATION, *PTHREAD_LAST_SYSCALL_INFORMATION;
```

Heures de thread

Tableau 8.34. Eléments auxiliaires sous-jacents à l'heure de fin d'un thread

DLL	**Élément**	**Attribut**
Kernel32.dll	ExitThread	aucun particulier
	GetThreadTimes	lpExitTime + LPFILETIME
	TerminateThread	aucun particulier
Ntdll.dll	KERNEL_USER_TIMES	ExitTime
	NtQueryInformationThread	ThreadInformationClass = ThreadTimes
	NtTerminateThread	aucun particulier
Ntoskrnl.exe	ETHREAD	ExitTime
	PspExitThread	aucun particulier

Tableau 8.35. Eléments auxiliaires sous-jacents à l'heure de création d'un thread

DLL	**Élément**	**Attribut**
Kernel32.dll	CreateThread	aucun particulier
	CreateRemoteThread	aucun particulier
	GetThreadTimes	lpCreationTime
Ntdll.dll	KERNEL_USER_TIMES	CreateTime
	NtCreateThread	aucun particulier
	NtQueryInformationThread	ThreadInformationClass = ThreadTimes
Ntoskrnl.exe	ETHREAD	CreateTime
	PspCreateThread	aucun particulier

Traitement multithread

Une large majorité des logiciels conçus pour les configurations matérielles actuelles comportent plusieurs threads, et donnent ainsi lieu à différents chemins d'exécution. Un navigateur Web peut par exemple héberger

un thread dédié à la présentation (affichage des images et du texte), et un autre dévolu à la lecture des données en provenance du réseau. Un traitement de texte peut avoir un thread occupé par le rendu de l'interface utilisateur, un autre par la gestion des saisies de l'utilisateur, et un troisième thread pour effectuer les corrections orthographiques et grammaticales en arrière-plan.

Avantages du traitement multithread. Les applications multithread améliorent sensiblement les performances notamment pour les raisons que voici :

- *Réactivité* La solution multithread permet à un programme de poursuivre son exécution même si l'une de ses parties est bloquée ou accomplie une opération de longue durée.
- *Économie* La création d'un processus consomme de façon intensive du temps et des ressources système. Comme les threads partagent les ressources du processus auquel ils appartiennent, il est dès lors moins onéreux (donc plus optimal) de créer et gérer des threads que des processus.

Notifications de niveau thread

En plus des routines appelées lors de la création/suppression de processus et lors le chargement d'image exécutable, l'exécutif Windows gère aussi un tel système de suivi pour la création et le démantèlement des threads.

Les pilotes peuvent déclarer leurs propres routines de notification de thread via l'interface PsSetCreateThreadNotifyRoutine. Une fonction de rappel installée avec PsSetCreateThreadNotifyRoutine à pour contexte d'exécution l'ID du nouveau thread, l'ID du processus auquel le thread appartient et, finalement, un attribut qui, actif ou non, permet de déterminer si l'opération ayant déclenchée la routine est une création ou une suppression de thread.

Windows donne la main à la routine de notification de thread au niveau IRQL PASSIVE_LEVEL. Quand un thread est créé, la routine s'exécute dans le contexte du thread créateur. Quand un thread est supprimé, elle s'exécute dans le contexte de ce thread après qu'il ai pris fin.

Un thread supprime une routine de notification par le biais de PsRemoveCreateThreadNotifyRoutine. Si la routine cible est fonctionnelle tandis que PsRemoveCreateThreadNotifyRoutine est appelée, cette dernière attend la fin des opérations en cours, ce afin de ne pas court circuiter le flux d'exécution.

En interne, les alertes relatives aux threads sont gérés de manière tout à fait similaire à celles concernant les processus, à savoir à l'aide d'un tableau de pointeurs sur fonction (variable noyau PspCreateThreadNotifyRoutine), et d'un nombre de routines déclarées (variable noyau PspCreateThreadNotifyRoutine).

Threads système

Les threads système ont tous les attributs des threads ordinaires mode utilisateur (contexte, priorité, affinité, etc.), à ceci près qu'ils fonctionnent uniquement en mode noyau, et exécutent du code résidant exclusivement dans l'espace système, que ce soit en corrélation avec le centre opérationnel de Windows (Ntoskrnl.exe), les sphères exécutives (Ntoskrnl.exe, encore), ou les pilotes. De threads threads n'ont en plus de cela pas d'espace d'adressage de processus utilisateur.

Windows emploie des threads système en diverses et nombreuses circonstances, par exemple pour gérer la diffusion des E/S (gestionnaire d'E/S), l'élagage des working sets (balance set manager), l'écriture des pages modifiées (gestionnaire de cache), l'ordonnancement des threads (noyau), les réponses aux requêtes réseau (pilote de serveur de fichiers Srv). Pour d'avantage de détails sur tel ou tel thread système particulier, reportez-vous au chapitre dévolu au composant concerné.

Par défaut, les threads système sont placés sous l'égide du processus système initial (baptisé System), mais un pilote peut créer un thread système dans n'importe quel processus. Ainsi, le pilote apparenté au sous-système Windows (Win32k.sys) crée des threads système dans le processus d'exécution client-serveur (Csrss.exe).

Ordonnancement des threads

Les sections qui suivent expliquent en détail les structures de données et algorithmes par lesquels Windows régit l'ordonnancement des threads.

Tableau 8.36. Éléments auxiliaires sous-jacents à l'ordonnancement

Élément	Type	Lieu	Description
DispatcherReadyListHead	LIST_ENTRY [32]	KPRCB	Têtes de liste des 32 files d'attente des prêts
ReadySummary	ULONG	KPRCB	Masque binaire des niveaux de priorité pour lesquels il existe un ou plusieurs threads prêt.

États d'un thread

Tout thread au sein du modèle d'ordonnancement de Windows voit son cycle de vie s'articuler autour de diverses étapes, appelées en interne états d'exécution. La liste qui suit énumère les différents états possibles pour un thread, et explique en détail la signification se rapportant à chacun.

- *Initialisé* (*Initialized*) État interne utilisé par le système d'exploitation afin de souligner le succès des opérations menées en vue de la création du thread, y compris l'allocation des ressources utiles à son exécution ultérieure. C'est seulement au stade ou un thread est mis à l'état Initialisé qu'il peut être l'objet de décisions de la part de la stratégie générale d'ordonnancement, laquelle traite d'ailleurs avec tous les autres états hormis celui-ci.

- *Prêt* (*Ready*) Un thread dans l'état Prêt attend d'être exécuté. Les choix de l'ordonnanceur en ce qui concerne le prochain thread à exécuter s'orientent uniquement sur la base de threads dans cet état.

- *Prêt différé* État employé pour les threads qui ont été sélectionnés pour exécution sur un processeur spécifique, mais qui n'ont pas encore été ordonnancés. En interne, cette configuration permet au noyau de minimiser la durée de détention du verrou global posé sur la base de données d'ordonnancement.

- *Se tenant prêt* (*Standby*) Le thread a été choisi pour s'exécuter au prochain tour sur un certain processeur, avec comme conséquence d'être le bénéficiaire futur le plus probable de l'utilisation du processeur. Rien n'étant encore joué à ce stade, les circonstances peuvent néanmoins encore aller contre ce schéma, par exemple si un thread de plus de grande importance fait irruption. De ce fait, notez qu'un thread peut être préempté hors de l'état actif avant même d'avoir été exécuté. Pour chaque processeur de la machine, il ne peut se trouver qu'un seul thread dans cet état.

- *En exécution* (*Running*) Une fois prise la décision de quel thread exécuter, et à quel endroit (quel processeur), le dispatcher du noyau effectue un basculement de contexte en faveur de ce thread, qui entre à cette occasion dans l'état En exécution. L'exécution du thread dure jusqu'à ce que son quantum expire, jusqu'à ce qu'il soit préempté par un thread de priorité supérieure, jusqu'à ce qu'il se termine, jusqu'à ce qu'il cède sa place ou jusqu'à ce qu'il entre (volontairement ou non) dans l'état d'attente. Dans un système multi processeur, plusieurs threads peuvent se trouver simultanément en cours d'exécution

- *En attente* (*Waiting*) Sous l'effet de facteurs exogènes défavorables, le thread n'entre plus en considération pour l'exécution. La transition vers cet état a lieu de plusieurs façons : le thread peut attendre volontairement un objet avec lequel se synchroniser, le système d'exploitation peut attendre pour le compte du thread, par exemple pour l'achèvement d'une opération d'E/S, ou un sous-système contraindre le thread à se mettre en pause. Quand l'attente se termine, s'il n'y a aucun autre thread de priorité supérieure ou égale dans la liste des threads prêts du dispatcher, le thread peut reprendre son exécution. Autrement, il repasse dans l'état Prêt. Les threads bloqués en attente d'une ressource matérielle ou logique temporairement indisponible sont placés dans la file d'attente de la primitive de synchronisation associée à la ressource.

- *En transition* (*Transition*) Le thread ne dispose momentanément pas de toutes les données indispensables à son fonctionnement, une partie de celles-ci s'étant vu paginées hors de la mémoire. Une fois ces éléments rechargés en mémoire, par exemple la pile mode noyau, le thread entrera dans l'état Prêt.

- *Terminé* (*Terminated*) Quand un thread a fini son exécution, il entre dans l'état Terminé, signe qu'il ne réclamera plus de temps processeur. Notez que cela ne signifie pas pour autant que toutes les ressources allouées à ce thread vont être supprimées. Le gestionnaire d'objet ne fait disparaitre toute donnée interne consacrée à un objet que lorsque la dernière référence vers cet objet est supprimée.

Les transitions entre états s'effectuent au gré de deux facteurs majeurs : d'une part les décisions d'ordonnancement prises par Windows, d'autre part les attentes sur synchronisation.

Dans l'éventualité où aucun thread n'est éligible pour l'exécution, le système passe alors la main à un thread spécial exécutant la fonction KiIdleLoop, laquelle se charge essentiellement de basculer le processeur dans un mode basse consommation en attendant que les circonstances se prêtent de nouveau à mener à bien un thread.

Le noyau regroupe les valeurs internes admissibles en ce qui concerne les états de thread au sein d'une variable d'énumération de type KTHREAD_STATE, dont le tableau que voici montre les éléments.

Tableau 8.37. KTHREAD_STATE

Indice	État d'ordonnancement
0	Initialized
1	Ready
2	Running
3	Standby
4	Terminated
5	Waiting
6	Transition
7	DeferredReady
8	GateWait

Fonctions Windows d'ordonnancement

La liste qui suit énumère les fonctions de l'API Windows ayant traits à l'ordonnancement des threads.

- *Get/SetPriorityClass* Retourne ou modifie la classe de priorités (priorité de base) d'un processus.
- *Get/SetProcessAffinityMask* Retourne ou modifie le masque d'affinité d'un processus.
- *Get/SetThreadPriority* Retourne ou modifie la priorité d'un thread (relative à la priorité de base du processus).
- *SetThreadAffinityMask* Définit le masque d'affinité d'un thread pour un certain ensemble de processeurs.
- *SetThreadIdealProcessor* Définit un processeur privilégié pour pour un thread.
- *Sleep* Suspend un thread pendant une durée spécifiée.
- *Suspend/ResumeThread* Suspend un thread ou reprend l'exécution d'un thread suspendu.

Contexte d'exécution

Comme tout autre système d'exploitation basé sur un modèle d'ordonnancement préemptif, Windows conserve en permanence la main sur les tâches exécutées par le processeur, et est à cet égard pourvu de capacités à, outre exécuter ou stopper une tâche planifiée en cours (via l'horloge système et des phénomènes d'interruptions), représenter l'état du processeur quand il fait fonctionner une tâche, chose que l'on appelle communément un contexte (ou *contexte d'exécution*) de thread.

Un contexte d'exécution désigne l'ensemble minimal de données à sauvegarder pour permettre une interruption de la tâche à un moment donné, et une reprise de cette exécution au point où elle a été interrompue. Lorsque le dispatcher déloge un thread du processeur qu'il occupe, il enregistre un certain nombre de données à son sujet,

parmi lesquelles les registres du processeur sur lequel le thread est exécuté, les zones de mémoire utilisées par lui, ou encore les registres de configuration tels que définies pour gérer ce thread en particulier. Quand les conditions se présentent et que le thread est élu pour l'exécution, son contexte est restauré, il peut de nouveau faire aller de l'avant.

Le contexte d'un thread dépend de l'architecture du processeur. Le noyau et la couche d'abstraction matérielle s'en servent pour réaliser des opérations spécifiques à la machine.

On trouve au contexte de thread deux utilisations distinctes, mais liés entre elles : l'une est de l'ordre de l'étude et de l'examen, l'autre du contrôle (sous-entendu de l'exécution).

Pour visualiser quelles données constituent le contexte d'un thread, utilisez la commande *dt* du débogueur noyau. Voici à quoi ressemble la sortie de la commande :

```
lkd> dt nt!_CONTEXT
   +0x000 P1Home           : Uint8B
   +0x008 P2Home           : Uint8B
   +0x010 P3Home           : Uint8B
   +0x018 P4Home           : Uint8B
   +0x020 P5Home           : Uint8B
   +0x028 P6Home           : Uint8B
   +0x030 ContextFlags     : Uint4B
   +0x034 MxCsr            : Uint4B
   +0x038 SegCs            : Uint2B
   +0x03a SegDs            : Uint2B
   +0x03c SegEs            : Uint2B
   +0x03e SegFs            : Uint2B
   +0x040 SegGs            : Uint2B
   +0x042 SegSs            : Uint2B
   +0x044 EFlags           : Uint4B
   +0x048 Dr0              : Uint8B
   +0x050 Dr1              : Uint8B
   +0x058 Dr2              : Uint8B
   +0x060 Dr3              : Uint8B
   +0x068 Dr6              : Uint8B
   +0x070 Dr7              : Uint8B
   +0x078 Rax              : Uint8B
   +0x080 Rcx              : Uint8B
   +0x088 Rdx              : Uint8B
   +0x090 Rbx              : Uint8B
   +0x098 Rsp              : Uint8B
   +0x0a0 Rbp              : Uint8B
   +0x0a8 Rsi              : Uint8B
   +0x0b0 Rdi              : Uint8B
   +0x0b8 R8               : Uint8B
   +0x0c0 R9               : Uint8B
   +0x0c8 R10              : Uint8B
   +0x0d0 R11              : Uint8B
   +0x0d8 R12              : Uint8B
   +0x0e0 R13              : Uint8B
   +0x0e8 R14              : Uint8B
   +0x0f0 R15              : Uint8B
   +0x0f8 Rip              : Uint8B
   +0x100 FltSave          : _XSAVE_FORMAT
   +0x100 Header           : [2] _M128A
   +0x120 Legacy           : [8] _M128A
   +0x1a0 Xmm0             : _M128A
   +0x1b0 Xmm1             : _M128A
   +0x1c0 Xmm2             : _M128A
   +0x1d0 Xmm3             : _M128A
   +0x1e0 Xmm4             : _M128A
   +0x1f0 Xmm5             : _M128A
   +0x200 Xmm6             : _M128A
   +0x210 Xmm7             : _M128A
```

```
+0x220 Xmm8             : _M128A
+0x230 Xmm9             : _M128A
+0x240 Xmm10            : _M128A
+0x250 Xmm11            : _M128A
+0x260 Xmm12            : _M128A
+0x270 Xmm13            : _M128A
+0x280 Xmm14            : _M128A
+0x290 Xmm15            : _M128A
+0x300 VectorRegister   : [26] _M128A
+0x4a0 VectorControl    : Uint8B
+0x4a8 DebugControl     : Uint8B
+0x4b0 LastBranchToRip  : Uint8B
+0x4b8 LastBranchFromRip : Uint8B
+0x4c0 LastExceptionToRip : Uint8B
+0x4c8 LastExceptionFromRip : Uint8B
```

Tableau 8.38. Opérations concernant le contexte d'exécution

Opération	Fonction	Service	Routine
Obtenir les valeurs des registres processeur d'un thread	GetThreadContext	NtGetContextThread	PsGetContextThread
Modifier les valeurs des registres processeur d'un thread	SetThreadContext	NtSetContextThread	PsSetContextThread

Commutation de contexte

Laisser à chaque thread l'occasion d'occuper momentanément un processeur nécessite une sauvegarde de l'état volatile de la machine associé au thread en cours d'exécution et un chargement de l'état sauvegardé du thread nouvellement élu. La procédure incarnant et concrétisant ces aspects est dite *commutation de contexte*.

En général, un basculement de contexte exige la mémorisation et le déploiement ultérieur des informations que voici :

- Pointeur d'instruction
- Pointeurs de pile utilisateur et de pile noyau
- Pointeur vers l'espace d'adressage du processus dans lequel le thread s'exécute (répertoire des tables de pages du processus)

Lorsqu'une commutation de contexte a lieu, le noyau commence par mémoriser les informations se rapportant au thread auquel il demande de laisser sa place en les plaçant sur la pile mode noyau courante. Une fois cela fait, il actualise le pointeur de pile puis le mémorise dans le bloc KTHREAD. Le pointeur de pile noyau est ensuite configuré de manière à correspondre à la pile noyau du thread sélectionné pour l'exécution, après quoi le contexte sauvegardé dudit thread est chargé. Dans l'éventualité où le thread se trouve dans un processus différent, il y a chargement de son répertoire des tables de pages dans un registre spécial du processeur (en l'occurence CR3).

Les commutations de contexte, ainsi que par extension les supports qui les véhiculent (registres machine et procédure de basculement par elle-même) dépendent étroitement de l'architecture du processeur. De ce fait, la vitesse d'une commutation de contexte varie d'une machine à une autre en fonction de la vitesse de la mémoire, du nombre de registres dont il faut assurer la préservation et, enfin, de l'existence d'instructions spéciales optimisées à cet effet.

Classes de priorité

Déterminant la façon dont le système planifie l'exécution du thread par rapport à celle de tous les autres threads du processus et d'autres processus, les niveaux de priorités sont dans Windows assignés de deux points de vue différents : d'un côté celui du noyau, d'un autre côté celui du sous-système Windows proprement dit. Pour simplifier cela, et exposer une vue réduite du grand nombre de priorités gérées en interne (32), l'API Windows

rend visible 6 classes de priorité (voir liste ci-dessous). Plus la priorité d'un processus est haute, plus Windows donne aux threads appartenant à ce processus une chance d'être exécuté.

- *Inactive* Les threads de processus appartenant à la classe Inactive (IDLE_PRIORITY_CLASS) s'exécutent seulement quand le système est au repos. Exemple : un écran de veille. Les threads du processus sont précomptés par les threads de n'importe quel processus s'executant dans classe de priorité plus élevée.

- *Inférieure à la normale* Classe intermédiaire (BELOW_NORMAL_PRIORITY_CLASS) entre la Normale et celle Inactive.

- *Normale* Les processus de la classe Normale (NORMAL_PRIORITY_CLASS) n'ont exigences particulières concernant leur d'ordonnancement. Cette classe est utilisée par défaut.

- *Supérieure à la normale* Classe intermédiaire (ABOVE_NORMAL_PRIORITY_CLASS) entre la Normale et la Haute.

- *Haute* Les processus s'exécutant dans la classe Haute (HIGH_PRIORITY_CLASS) font des actions qui exigent un court temps de réponse, quelle que soit les conditions. Exemple : le gestionnaire des tâches, qui doit répondre quelle que soit la charge du système.

- *Haute* Les processus de la classe Haute (HIGH_PRIORITY_CLASS) effectuent des tâches critiques dans le temps qui, quelle que soit les conditions, doivent être exécutées rapidement. Exemple : le gestionnaire des tâches, qui, sollicité par l'utilisateur de la session interactive, doit répondre rapidement quelle que soit la charge du système.

- *Temps réel* Les processus de le classe Temps réel ont un aspect critique en regard des performances, et requièrent à cet égard la plus haute priorité́ possible. Les threads d'un tel processus préemptent les threads de tous les autres processus, y compris ceux appartenant au système d'exploitation.

Normalement, la propriété de base du processus (et, partant, celle du thread) est par défaut égale à la valeur médiane de chaque plage de priorité de processus : priorité niveau 24 pour la classe Temps réel, niveau 13 pour Haute, niveau 10 pour Supérieure à la normale, niveau 8 pour Normale, niveau 6 pour Inférieure à la normale et 4 pour Inactive.

Si la classe de priorité du processus change, sa valeur de la priorité de base change également. Cette modification change automatiquement la priorité actuelle de chaque thread associé à ce processus, car cette valeur est toujours calculée à partir d'une combinaison entre la priorité de base du processus et la priorité relative du thread.

Par défaut, un processus hérite sa priorité de base du processus créateur. Il est néanmoins possible de redéfinir ce comportement dans la fonction CreateProcess, laquelle accepte une valeur pour la priorité du processus, ou par un argument passé à la commande start. Par exemple, la commande start /realtime notepad lance le programme Bloc-note en assignant au processus d'application la classe Temps réel. (Comme déjà mentionné dans ce chapitre, placer un processus dans la classe Temps réel est une opération qui requiert le privilège Augmenter la priorité de planification.) Les fonctions SetPriorityClass et GetPriorityClass permettent respectivement de modifier et de lire la classe de priorités d'un processus.

Quelques processus système de Windows, tels le Gestionnaire de fenêtres du Bureau (Dwm.exe), le processus d'ouverture de session (Winlogon.exe) ou le processus de démarrage de Windows (Wininit.exe), ont dès le départ une classe de priorité plus élevée que la normale, ce qui tend à garantir que les threads de ces processus, en partant avec une priorité de base initiale standard supérieure à la valeur par défaut de la classe Normale (en l'occurence 8), soient les plus réactifs possibles.

Suspension et reprise de thread

Windows laisse les sous-systèmes environnementaux être responsable d'arrêts temporaires pour les tâches dont ils font la gestion, et décider ainsi de mettre en pause, ou au contraire, de reprendre tel ou tel thread.

Dans le premier cas, le système d'exploitation suspend les activités dudit thread, ce qui l'écarte d'office des choix possibles en matière d'ordonnancement. Dans le second, le thread retourne à sa configuration d'origine et redevient de la sorte éligible pour l'ordonnancement.

Chaque thread suspendu l'est en interne par le biais d'un compteur dont les évolutions varient au gré des scénarios de synchronisation. L'API Windows expose 2 fonctions ayant une incidence directe sur la valeur de ce compteur : SuspendThread, qui l'accroit, et ResumeThread, qui accomplit l'inverse et le fait en l'occurence diminuer, dans les deux cas d'une unité. Un thread suspendu deux fois doit donc être réveillé deux fois avant d'être exécuté (à quoi correspond un compteur ayant une valeur nulle).

Au niveau de la sécurité et des droits d'accès, soustraite ou redonner à un thread sa capacité d'être ordonnancée est une opération pour laquelle le requérant doit disposer de la permission THREAD_SUSPEND_RESUME.

Si, dans Windows, une part de responsabilité est laissée aux sous-systèmes d'environnement concernant suspension et reprise de threads, lesquelles activités échoient d'habitude au noyau, c'est en raison de la nécessité de telles aptitudes au sein d'application de type débogueur.

Affinité

Chaque thread (comme du reste chaque processus) a un masque d'affinité qui spécifie les processeurs sur lesquels le thread a le droit de fonctionner. Le masque d'affinité du thread est hérité du masque d'affinité du processus, lui-même hérité du processus créateur. Par défaut, tous les processus (et donc tous les thread) sont configurés avec un masque d'affinité correspondant à l'ensemble des processeurs actifs, ce qui laisse de la sorte le champ libre au système d'ordonnancer les threads sur tous les processeurs disponibles (en d'autres termes, à condition que les conditions s'y prêtent, n'importe quel thread peut occuper n'importe quel processeur).

Une application peut, pour améliorer les performances et/ou répartir la charge sur un ensemble spécifique de processeurs, procéder à différents réglages en ce qui concerne l'affinité. Les possibilités à cet égard sont multiples :

- Définir l'affinité d'un thread individuel dans un processus.
- Définir l'affinité de tous les threads apparentés à un même processus. Une large majorité des logiciels assurant une gestion des processus, dont le gestionnaire des tâches intégrés à Windows, offrent une interface graphique sur cette option. La commande start, permettant de lancer une application à partir de ligne de commande, prend en compte la notion d'affinité via le commutateur /affinity.
- Introduire un processus dans un job.
- Spécifier une configuration d'affinité dans l'entête de l'image.
- Activer le flag mono processeur d'une image.

Partant du principe de limiter le déplacement des threads, Windows ne considère pas l'affinité comme un motif suffisant pour la migration d'un thread d'un processeur vers un autre. Ainsi, considérez le scénario suivant :

- Le processeur 0 exécute un thread A de priorité 8 et dont les paramètres au niveau de l'affinité n'imposent aucune restriction particulière.
- Le processeur 1 exécute un thread B de priorité 4 qui, comme le thread A, peut fonctionner sur n'importe quel processeur.
- Un thread C de priorité 6 qui ne peut être exécuté que sur le processeur 0 devient prêt pour l'exécution.

On pourrait en la circonstance penser que Windows va transférer le thread A (priorité 8) du processeur 0 au processeur 1 (préemptant au passage le thread B, de priorité 4) pour exécuter le thread C. Il n'en est en fait rien ; thread C devra attendre que la place qu'il réclame (processeur 0) soit laissée vacante à la fin de l'exécution du thread A.

Tableau 8.39. Opérations sur l'affinité

Opération	Fonction	Service
Obtenir le masque d'affinité d'un processus	GetProcessAffinityMask	NtQueryInformationProcess (ProcessBasicInformation)
Modifier le masque d'affinité d'un processus	SetProcessAffinityMask	NtSetInformationProcess (ProcessAffinityMask)
Modifier le masque d'affinité d'un thread	SetThreadAffinityMask	NtSetInformationThread (ThreadAffinityMask)

Tableau 8.40. Interfaces noyau concernant l'affinité

Routine	Description	Autres références
KeSetSystemAffinityThread	Modifie le masque d'affinité du thread actuel	KTHREAD Affinity, SystemAffinityActive

Visualisation et modification de l'affinité de processus. Pour modifier l'affinité d'un processus, procédez comme suit :

1. Ouvrez l'invite de commandes (cmd.exe).
2. Exécutez le gestionnaire des tâches (taskmgr.exe), accédez à l'onglet Détails, puis repérez le processus cmd.exe dans la liste.
3. Cliquez avec le bouton droit de la souris sur le processus, puis choisissez l'option Définir l'affinité. Une liste de processeurs devrait alors apparaître.
4. Sélectionnez à l'aide des cases à cocher un sous-ensemble des processeurs disponibles, puis cliquez sur OK. Les threads du processus sont désormais restreints à s'exécuter uniquement sur les processeurs pour lesquels vous venez d'opter.
5. Exécutez maintenant le bloc-notes (notepad.exe) depuis l'invite de commandes (la même que nous avons utilisé dans les étapes précédentes).
6. Retournez dans le gestionnaire des tâches, puis repérez le processus du bloc-notes. Faites un clic-droit dessus, puis sélectionnez Définir l'affinité. Vous devriez à ce moment voir la même liste de processeurs que celle choisie en amont en ce qui concerne le processus de l'invite de commandes, preuve en est que les processus héritent de l'affinité de leurs parents.

A titre informatif, notez que les modifications apportées à l'affinité d'un processus ne vont pas au delà de l'exécution dudit processus. Utilisez l'option /affinity de la commande start pour mémoriser les paramètres d'affinité à appliquer automatiquement au démarrage d'une application.

Affinité système

Les pilotes Windows s'exécutent généralement dans le contexte du thread appelant ou d'un thread arbitraire (choisi ou non), et se voient par conséquent soumis aux mêmes obligations en ce qui concerne les processeurs qu'ils ont le droit d'occuper. Dans l'éventualité où ces exigences se heurteraient aux circonstances optimales pour le fonctionnement des pilotes (notamment pour tout ce qui a trait au traitement des interruptions et autres travaux organisés en file d'attente), Windows offre un mécanisme permettant de contourner temporairement les paramètres d'affinité qui émanent de l'exécution en mode utilisateur, incarné en l'occurence par des routines comme KeSetSystemAffinityThread(Ex) et KeRevertToUserAffinityThread(Ex.

Tableau 8.41. Routines concernant l'affinité système

Routine	Description
KeRevertToUserAffinityThreadEx	Restaure l'affinité de processeur du thread appelant à sa valeur d'origine.
KeRevertToUserGroupAffinityThread	Restaure l'affinité de groupe du thread appelant à sa valeur d'origine.
KeSetSystemGroupAffinityThread	Modifie le numéro de groupe et le masque d'affinité du thread appelant.
KeSetSystemAffinityThread(Ex)	Définit l'affinité système du thread appelant.

Thread inactif

Lorsqu'aucun autre scénario d'ordonnancement ne peut avoir lieu, Windows planifie pour l'exécution un thread dit inactif, dont la principale caractéristique est d'endosser auprès du système le rôle d'un thread toujours prêt à s'exécuter. Dans un système multi processeur, tous, s'ils n'ont d'autres opérations à faire aller de l'avant, sont occupés par un thread inactif (aucune tâche ne pourrait être immédiatement éligible sur un CPU, ou bien toutes les tâches pourraient déjà avoir été distribuées sur un CPU).

Chaque thread inactif fait fonctionner le même service système, KiIdleLoop, exécuté au niveau DPC/dispatch, qui boucle en quête d'opérations à effectuer tels les livraisons de DPC ou la recherche de threads à dispatcher.

Pour optimiser les performances, Windows gère les threads inactifs comme des cas particuliers, et ne prend en l'occurence pas en compte la priorité desdits threads. De ce fait, la boucle inactive n'est exécutée que s'il n'existe réellement pas d'autre séquence d'instructions qui puisse être mis en mouvement.

Accélérations de priorité

Afin de prendre en compte des scénarios d'ordonnancement potentiellement injustes, Windows peut accéler (augmenter) la priorité courante d'un thread. Cette façon de procéder se justifie en vertu de plusieurs cas, que voici :

- Lors de la complétude d'opérations d'E/S
- Après attente d'événements ou de sémaphores
- Après que des threads du processus de premier plan ont fini une opération d'attente
- Quand des threads d'interface utilisateur se réveillent pour cause d'activités dans les fenêtres
- Quand un thread prêt pour l'exécution n'a pas été exécuté depuis un certain temps

Tableau 8.42. Accélérations de priorité

Fonction	Service	Routine
GetProcessPriorityBoost	NtQueryInformationProcess (ProcessPriorityBoost)	
GetThreadPriorityBoost	NtQueryInformationThread (ThreadPriorityBoost)	KeSetDisableBoostThread
SetProcessPriorityBoost	NtSetInformationProcess (ProcessPriorityBoost)	
SetThreadPriorityBoost	NtSetInformationThread (ThreadPriorityBoost)	

Élément	Lieu	Autres références
DisableBoost	KPROCESS	Get/SetProcessPriorityBoost
DisableBoost	KTHREAD	Get/SetThreadPriorityBoost

Accélération de priorité pour les tâches multimédia (MMCSS)

Un autre cas de figure, peut-être le plus atypique de tous, où Windows admet des accélérations de priorité concerne la lecture de contenus multimédia, par nature très gourmands en termes de ressources machine et qui occupent un rôle prépondérant pour ce qui est de la fiabilité perçue du système d'exploitation. De ce fait, Windows intègre un composant spécial appelé MultiMedia Class Scheduler Service (MMCSS), lequel garantit aux programmes multimédia des performances optimisées.

MMCSS fonctionne selon une perspective d'organisation des tâches, et emploie des informations stockées dans le registre pour identifier lesquelles sont prises en charge et déterminer quels ajustements seraient opportuns pour les threads effectuant ces tâches.

MMCSS ajuste la priorité des threads en fonction la catégorie à laquelle ils appartiennent.

Tableau 8.43. Catégories d'ordonnancement

Catégorie	Priorité	Description
High	23-26	Threads s'exécutant à une priorité plus élevée que tout autre, à l'exception des threads système critiques.
Medium	16-22	Threads faisant partie de l'application qui se trouve au premier plan.
Low	8-15	Threads qui ne font partie d'aucun des catégories susmentionnées.
Exhausted	1-7	Threads ayant épuisé leur part d'utilisation du ressources, et qui ne continueront dès lors à s'exécuter que si aucun autre thread de priorité supérieure n'est prêt à le faire.

Sur le plan structurel, MMCSS est implémenté sous forme d'un service en mode utilisateur s'exécutant dans un processus hôte partagé (svchost).

Les paramètres de configuration à l'usage de MMCSS sont regroupés dans le registre sous HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile. Parmi les informations enregistrés à cet endroit, observez au premier plan la valeur SystemResponsiveness vous permettant d'ajuster la quantité d'utilisation du processeur que MMCSS garantit aux threads de faible priorité.

```
C:\>reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile"

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile
    NetworkThrottlingIndex    REG_DWORD    0xa
    SystemResponsiveness    REG_DWORD    0x14

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile\Tasks

C:\>reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile\Tasks"

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile\Tasks\Audio
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile\Tasks\Capture
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile\Tasks\DisplayPostProcessing
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile\Tasks\Distribution
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile\Tasks\Games
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile\Tasks\Playback
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile\Tasks\Pro Audio
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile\Tasks\Window Manager

C:\>reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile\Tasks\Audio"

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile\Tasks\Audio
    Affinity    REG_DWORD    0x0
    Background Only    REG_SZ    True
    Clock Rate    REG_DWORD    0x2710
    GPU Priority    REG_DWORD    0x8
    Priority    REG_DWORD    0x6
    Scheduling Category    REG_SZ    Medium
    SFIO Priority    REG_SZ    Normal
```

Contrairement aux autres scénarios qui conduisent Windows à parfaire la stratégie d'ordonnancement de quelques threads savamment élus, lesquels s'appuient sur des décisions étant du ressort exclusif du noyau, les accélérations de priorité concernant des programmes multimédia sont gérés en premier lieu par MMCSS, qui relaie au noyau la demande de prioriser tel ou tel thread.

Scénarios d'ordonnancement

Windows, lequel implémente un multi tache préemptif axé sur les priorités de thread, doit un moment donné faire le choix de quelle unité de code exécuter. En pratique, cette décision est motivée par plusieurs facteurs, plusieurs *scénarios* dans lesquels le système d'exploitation doit déterminer, entre autres choses, quand faire céder sa place au thread en cours d'execution, à quel autre thread faire profiter d'une commutation de contexte,

dans quelle mesure augmenter ou diminuer les niveaux de priorité, et finalement, à combien se monte la facture en terme d'unité de quantum.

La liste qui suit énumère les différents motifs en vue desquels Windows doit examiner l'éventualité d'une commutation de contexte. (Notez que les situations présentées ne le sont ici que dans les grandes lignes, les sections suivantes donnant de plus amples détails sur chacune des éventualités.)

- Mise en sommeil suite à une requête d'E/S non satisfaite (basculement volontaire)
- Fin du laps de temps imparti pour l'exécution (fin de quantum)
- Modification des conditions auxquelles est subordonnée la stratégie générale d'ordonnancement du système, avec comme possible effet l'éligibilité nouvelle d'un thread de priorité plus élevé que l'actuel (préemption)
- Cessation par achèvement (fin)

Basculement volontaire

Un thread peut volontairement laisser sa place et entrer dans un état d'attente, soit parce qu'il attend un objet avec lequel synchroniser son exécution, ou que le système d'exploitation attend pour son compte (par exemple dans le contexte d'un appel de procédure asynchrone).

Préemption

Les politiques d'ordonnancement en vigueur dans le noyau de Windows donnent inconditionnellement faveur au thread exécutable ayant la plus haute priorité. Aussi, toute tâche en cours est interrompu lorsqu'une tâche de priorité supérieure devient prête pour l'exécution. Cela peut se produire pour deux raisons : soit l'attente d'un thread de plus forte priorité a pris fin, soit la priorité d'un thread est augmentée ou diminuée. Quand un thread est préempté, il est placé en tête de la file des prêts associée à la priorité avec laquelle il fonctionnait, de sorte à pouvoir s'exécuter à nouveau dès que le thread préemptant aura cédé sa place.

Fin de quantum

Lorsque le quantum du thread en cours d'execution s'est écoulé, Windows doit déterminer si ledit thread peut continuer à fonctionner ou s'il faut en ordonnancer un différent. Si aucun autre thread de priorité supérieure ou égale n'est prêt pour l'exécution, le thread courant peut continuer d'occuper le processeur. S'il y a d'autres threads dans la file des prêts de même priorité, Windows sélectionne le thread suivant dans cette file et met à la fin de la file le thread précédemment exécuté.

Fin

Quand un thread termine son exécution (quelles qu'aient été les raisons de cet arrêt), il passe de l'état En exécution à l'état Terminé. Si aucune référence établie ne le concerne (handles ouverts), le thread est supprimé de la liste des threads du processus et les structures de données associées sont supprimées de la mémoire.

Généralités sur l'ordonnancement dans Windows

Windows implémente un système d'ordonnancement préemptif axé sur la priorité de threads individuels. Chaque thread dans l'environnement dispose d'une priorité (sa priorité d'exécution), le qualifiant de la sorte auprès de la politique générale d'exécuition du système. Cette priorité est représentée au moyen d'une plage de valeurs, de 0 à 31, inclusivement ; des nombres plus élevés indiquent des threads de priorité supérieure. Le thread exécutable présentant la priorité la plus élevée est toujours exécuté en premier.

Chaque thread sélectionné est autorisé à s'exécuter pendant une durée appelée quantum. Un quantum définit la quantité maximale de temps processeur qu'un thread peut épuiser avant que le noyau n'interrompe le thread afin de déterminer s'il existe un autre thread exécutable présentant la même priorité ou une priorité plus élevée. Dans la mesure où l'algorithme de planification est préemptif, un thread peut être interrompu avant la fin du

quantum si un autre thread de priorité plus élevée est prêt à s'exécuter. Les valeurs du quantum peuvent varier selon quelle version du système est installé, du type de processeur sur lequel Windows est lancé, des paramètres de performance et, finalement, de la configuration logicielle en cours (par exemple, si le thread fait partie d'un processus d'application multimédia, ou encore selon le statut premier plan / arrière-plan du processus).

Fait intéressant à noter, il n'existe pas pour servir d'abri au code ordonnancement dans Windows de couche logicielle aux contours bien délimités (en tout cas pas au sens usuel du terme, entendu au sens de l'architecture du système), à savoir qu'il n'existe pas de module autonome ou de routine « ordonnanceur unique » : le code, disséminé dans tout le noyau, est sollicité chaque fois qu'une question liée à l'ordonnancement doit être élucidée, exécuté par un thread ou un DPC (appel de procédure différé).

Si, comme la plupart des systèmes d'exploitation modernes, Windows utilise les notions de threads et de processus pour exécuter du code, il fait son ordonnancement seulement au niveau du thread. Cette approche est logique si vous considérez que les processus, du point de vue des mécanismes bas niveau en oeuvre dans le dispatcheur, ne sont pas réellement executés ; ils fournissent des ressources et un contexte dans lequel leurs threads sont exécutés mais, stricto sensu, eux ne le sont pas. Aussi, pour tout thread, on ne tient pas compte du processus auquel il appartient, ce dans l'optique de répartir équitablement le temps processeur entre tous les threads de chaque processus. Par exemple, pour deux processus, si le processus A contient 2 threads exécutables, le processus B un seul, et que les 3 threads ont tous la même priorité, chaque thread reçoit théoriquement un tiers du temps processeur.

Quantum

La planification des threads Windows repose sur un algorithme à quantité temporelle finie (tranche de temps), lequel assujetti tout thread à une durée maximale pendant laquelle occuper le processeur. Ainsi, quand un thread a été sélectionné pour l'execution, il est exécuté pendant une quantité de temps dite quantum.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\PriorityControl
    Win32PrioritySeparation    REG_DWORD    0x2
```

- *Court ou long* Une valeur égale à 1 signifie long, alors que 2 signifie court. Une valeur égale à 0 ou 3 indique que l'on utilise la valeur par défaut (long pour les systèmes serveur, court pour les autres).
- *Variable ou fixe* Une valeur égale à 1 implique la variation du quantum du processus de premier plan, alors que 2 signifie que le quantum ne change pas. Une valeur égale à 0 ou 3 indique que l'on utilise la valeur par défaut (fixe pour les systèmes serveur, variable pour les autres).
- *Augmentation de quantum pour premier plan* Le tableau suivant énumère les valeurs possibles de PspForegroundQuantum.

Tableau 8.44. Valeurs de quantum

	Court			Long		
Variable	6	12	18	12	24	36
Fixe	18	18	18	36	36	36

La longueur de l'intervalle d'horloge varie selon la plateforme matérielle. Ainsi, l'intervalle pour la plupart des machines mono processeur est d'environ 10 millisecondes, et 15 pour les machines multi processeur. Pour connaitre l'intervalle d'horloge réel, examinez le contenu de la variable globale système KeMaximumIncrement.

Accroissement de quantum

Windows distingue le processus en avant-plan actuellement sélectionné à l'écran, des processus en arrière-plan, qui ne sont pas sélectionnés. Quand une fenêtre d'application passe au premier plan sur une station ou un poste client, tous les threads du processus de premier plan (processus propriétaire de la fenêtre ayant le focus) voient leur quantum d'ordonnancement augmenter d'un certain facteur. Visée de cette approche : fournir via le don de temps processeur plus important que la moyenne de bonnes performances aux programmes interactifs, ce qui laisse à l'utilisateur une meilleure impression d'ensemble de la réactivité du système.

Par défaut, quand une instance de programme en cours est concernée par l'accroissement de quantum (toutes ne le sont pas, voir paragraphe ci-après), l'augmentation alloue au processus à l'avant-plan trois fois plus de temps pour s'exécuter - autrement dit, le quantum des threads du processus de premier plan est triplé. Notez que cet ajustement est évidement temporaire, à savoir qu'une fois ôté le focus d'une fenêtre d'un processus, c'est au nouveau processus de premier plan dont Windows fait bénéficier la révision du quantum.

Les règles d'ordonnancement spécialement conçues pour le processus de premier plan sont appliquées seulement sur des systèmes réglés sur Programmes dans les options de performances, et si ce processus a une priorité supérieure à Inactive.

Processeur idéal

Sur les systèmes multi processeurs, il revient à chaque thread un *processeur idéal*, lequel est celui privilégié pour l'exécution du thread. Lors de l'affectation d'un thread à un processeur, Windows tend à donner toujours la préférence d'abord à son processeur idéal, maximisant de la sorte les chances qu'un thread à de fonctionner sur le processeur pour lequel il a affiché sa prédilection.

Le choix du processeur idéal se fait lors de la création du thread à l'aide d'un attribut enregistré dans le bloc noyau du processus (KPROCESS ThreadSeed). La valeur dudit attribut est incrémentée chaque fois que le système se prépare à donner lieu à un thread, de façon que le processeur idéal de chaque nouveau thread soit différent. Par exemple, le premier thread du premier processus du système se voit assigner comme processeur idéal le numéro 0 ; le second thread le numéro 1. Pour le processus suivant, le premier thread aura comme processeur idéal le numéro 1, le second le numéro 2, et ainsi de suite. Windows procède ainsi de sorte à répartir équitablement les threads de chaque processus sur tous les processeurs.

Sur les systèmes hyperthread, le processeur idéal est le premier processeur logique disponible du processeur physique suivant. Par exemple, sur une machine biprocesseur hyperthread (soit quatre processeurs logiques), si le processeur idéal du premier thread est le processeur logique 0 (lequel se trouve sur le processeur physique 0), le second thread se verra assigné le processeur logique 2 (processeur physique 1), le troisième thread le processeur logique 1 (processeur logique 0), le quatrième thread le processeur logique 3 (processeur physique 1), et ainsi de suite. Les threads sont de cette façon répartis équitablement entre les processeurs physiques.

Tableau 8.45. Interfaces concernant le processeur idéal

Fonction	Service
SetThreadIdealProcessor(Ex)	NtSetInformationThread(ThreadIdealProcessor)

Base de données du dispatcher

Les choix stratégiques effectuées par Windows en matière d'ordonnancement dérivent d'un ensemble de structures de données appelées collectivement *base de données du dispatcher*, laquelle assure le suivi des threads en attente d'exécution.

Les files d'attente des prêts du dispatcher (DispatcherReadyListHead) contiennent les threads qui attendent d'être ordonnancés pour l'exécution, et sont par conséquent dans l'état Prêt. Il existe autant de files que de niveaux de priorités gérés, à savoir 32.

Pour améliorer la gestion des files des prêts, et de la sorte accélérer la sélection du thread a exécuter, le noyau gère un masque de 32 bits dit résumé des prêts (ReadySummary). Chaque bit armé au sein dudit masque rend compte de la présence d'au moins un thread prêt pour ce niveau de priorité. (Le bit 0 représente la priorité 0, le bit 1 la priorité 1, et ainsi de suite.)

Windows synchronise les accès à la base de données du dispatcher différemment selon que le système est mono ou multi processeur. Sur les systèmes mono processeur, une telle forme de protection est assurée via élévation de l'IRQL au niveau SYNCH_LEVEL, minimisant ainsi le risque qu'un thread puisse interrompre la selection des threads. Sur les systèmes multi processeur, procéder ainsi (augmenter l'IRQL) n'offre pas de garantie suffisante

pour isoler la base de données du dispatcher des opérations qui émanent des autres processeurs. Pour en apprendre plus sur la façon dont le noyau s'adapte à ces contraintes, reportez-vous à la section Systèmes multi processeur donnée plus loin.

Systèmes multi processeur

Dans un environnement mono processeur, où l'ensemble des activités liées à de la préemption est localisé à un seul endroit, l'ordonnancement sous Windows est relativement simple à appréhender : les algorithmes de sélection de threads exécutables misent irrémédiablement sur le thread de plus haute priorité, de sorte que ce soit toujours la tâche à la criticité la plus forte qui soit exécutée. Là où l'ordinateur à plus d'un seul processeur, la situation est plus délicate, l'architecture de ces systèmes induisant pour le système d'exploitation l'adoption d'un modèle de multitraitement, avec en filigrane la prise en compte dans la gestion de l'ordonnancement de la multiplicité des processeurs.

Systèmes NUMA

Le noyau gère les données de noeud via une structure KNODE. La variable noyau *KeNodeBlock* est un tableau de pointeurs vers les structures KNODE de chaque noeud.

```
lkd> dtx nt!_KNODE poi(nt!KeNodeBlock)
(*((nt!_KNODE *)0xfffff8016458a240))                [Type: _KNODE]
    [+0x000] IdleNonParkedCpuSet : 0x0 [Type: unsigned __int64]
    [+0x008] IdleSmtSet       : 0x0 [Type: unsigned __int64]
    [+0x010] IdleCpuSet       : 0x0 [Type: unsigned __int64]
    [+0x040] DeepIdleSet      : 0x0 [Type: unsigned __int64]
    [+0x048] IdleConstrainedSet : 0x0 [Type: unsigned __int64]
    [+0x050] NonParkedSet     : 0x3 [Type: unsigned __int64]
    [+0x058] NonIsrTargetedSet : 0x0 [Type: unsigned __int64]
    [+0x060] ParkLock         : 0 [Type: long]
    [+0x064] ThreadSeed       : 0x0 [Type: unsigned short]
    [+0x066] ProcessSeed      : 0x0 [Type: unsigned short]
    [+0x080] SiblingMask      : 0x1 [Type: unsigned long]
    [+0x088] Affinity         [Type: _GROUP_AFFINITY]
    [+0x088] AffinityFill     [Type: unsigned char [10]]
    [+0x092] NodeNumber       : 0x0 [Type: unsigned short]
    [+0x094] PrimaryNodeNumber : 0x0 [Type: unsigned short]
    [+0x096] Spare0           : 0x0 [Type: unsigned short]
    [+0x098] SharedReadyQueueMask : 0x3 [Type: unsigned __int64]
    [+0x0a0] StrideMask       : 0x1 [Type: unsigned __int64]
    [+0x0a8] ProximityId      : 0x0 [Type: unsigned long]
    [+0x0ac] Lowest           : 0x0 [Type: unsigned long]
    [+0x0b0] Highest          : 0x1 [Type: unsigned long]
    [+0x0b4] MaximumProcessors : 0x20 [Type: unsigned char]
    [+0x0b5] Flags            [Type: _flags]
    [+0x0b6] Spare10          : 0x0 [Type: unsigned char]
    [+0x0b8] HeteroSets       [Type: _KHETERO_PROCESSOR_SET [5]]
    [+0x130] PpmConfiguredQosSets [Type: unsigned __int64 [4]]
    [+0x150] LLCLeaders       : 0x1 [Type: unsigned __int64]
```

Vous pouvez examiner les informations qui se perpétuent concernant chaque noeud d'un système NUMA à l'aide de la commande !numa du débogueur noyau.

Systèmes hyperthread

Dans la perspective du système d'exploitation, chaque processeur unique doté de la technologie hyperthread se comporte comme deux processeurs logiques, ouvrant ainsi aux algorithmes d'ordonnancement la possibilité de planifier et affecter des ressources à deux threads la fois.

Windows gère les machines multi processeur hyperthread de telle sorte à répartir équitablement les threads entre les processeurs physiques, ne faisant intervenir les processeurs logiques d'un même coeur que si les

circonstances l'y force. De ce fait, s'il existe un processeur physique libre, le noyau opte pour un processeur logique de ce processeur.

Pour examiner les données avec lesquelles traite Windows relativement à la gestion de l'ordonnancement dans les système hyperthread, utilisez la commande !smt du débogueur noyau.

Base de données du dispatcher multi processeur

Sur les systèmes multi processeur, la base de données du dispatcher assure en plus du suivi des threads en attente d'exécution l'état des processeurs du système. En interne, cette prise en charge se traduit par deux attributs.

- Le masque des processeurs actifs (KeActiveProcessors), dans lequel chaque bit armé représente un processeur utilisable. (Il peut y avoir moins de processeurs utilisables que de processeurs présents, selon les modalités de la licence d'utilisation de la version de Windows exécutée sur la machine.)
- Le résumé des inactifs (KiIdleSummary), dans lequel chaque bit armé représente un processeur inactif. Comme expliqué dans la section "Base de données du dispatcher" vue en amont, sur les systèmes mono processeur, la base de données du dispatcher est verrouillée via augmentation de l'IRQL. Sur les systèmes multi processeur, où les accès concurrents peuvent émaner de n'importe quel processeur, cette protection ne suffit pas. -

Algorithmes d'ordonnancement multi processeur

La stratégie générale d'ordonnancement dans Windows tend à exploiter au maximum l'ensemble des processeurs disponibles. De ce fait, quand un thread devient prêt, Windows l'affecte en priorité à un processeur inactif. Dans l'éventualité où plusieurs processeurs sont inactifs, la préférence est donnée d'abord au processeur idéal du thread, puis au processeur précédent du thread, puis au processeur courant (autrement dit le processeur exécutant le code d'ordonnancement).

Une fois qu'un processeur a été élu comme moteur d'exécution pour un thread, le système actualise l'état de ce thread à l'état Se tenant prêt (Standby), et le bloc de contrôle (KPRCB) du processeur est modifié de façon à pointer vers ce thread.

Niveaux de priorité de thread

Chaque thread Windows a en interne deux priorités : la valeur courante et la valeur de base. Les choix d'ordonnancement s'appuient la priorité courante.

La priorité de base initiale d'un thread est héritée de la priorité de base du processus auquel il appartient. Il est possible de modifier la priorité de base d'un thread à l'aide de la routine KeSetBasePriorityThread.

Tableau 8.46. Eléments auxiliaires sous-jacents aux priorités de thread

Élément	Lieu	Autres références
BasePriority	THREAD_BASIC_-INFORMATION	Fonctions : GetThreadPriority. Services : NtQueryInformationThread (ThreadInformationClass = ThreadBasicInformation).
BasePriority	KTHREAD	Fonctions : SetThreadPriority. Services NtSetInformationThread (ThreadInformationClass = ThreadBasePriority). Routines : KeGetBasePriorityThread, KeSetBasePriorityThread.
Priority	KTHREAD	Routines : KeQueryPriorityThread, KeSetPriorityThread
Priority	THREAD_BASIC_-INFORMATION	Services : NtQueryInformationThread (ThreadInformationClass = ThreadBasicInformation).

Groupes de processeur

Les versions actuelles de Windows gèrent plus de 64 processeurs logiques sur un seul ordinateur (contre 32 initialement, puis 64 par la suite, consécutivement à l'émergence des premiers systèmes et plateformes 64

bits). En interne, une telle prise en charge dérive de la notion de_groupe de processeurs_, à quoi se ramène un ensemble statique de processeurs logiques (64 maximum) traités comme une entité de planification unique.

Par défaut, une application (un processus et ses threads) est limitée à un seul groupe, ce qui signifie qu'un thread peut être planifié sur l'un des 64 processeurs (au maximum) qui font partie de son groupe actuel. Lorsqu'un processus est créé, Windows lui assigne un groupe de processeurs de manière cyclique (stratégie round-robin), de sorte que tous les processus soient répartis uniformément entre les différents groupes, cherchant ainsi un équilibrage de charge optimal. Tous les threads d'un processus appartiennent au même groupe que le processus lui-même.

Une application qui nécessite l'utilisation de plusieurs groupes pour s'exécuter sur plus de 64 processeurs doit déterminer explicitement où exécuter ses threads, et définir les affinités de processeur qu'implique une telle configuration. Le flag `INHERIT_PARENT_AFFINITY`, utilisé lors de la création d'un processus (paramètre dwCreationFlags dans l'appel à CreateProcess), permet en la circonstance de s'assurer que ce dernier hérite du groupe de processeurs de son parent (plutôt que de laisser le système déterminer ce groupe automatiquement). Autre solution, plus souple : l'attribut `PROC_THREAD_ATTRIBUTE_GROUP_AFFINITY` de la function CreateRemoteThreadEx, utilisable pour créer des threads avec des affinités de groupe de processeurs spécifiques.

Les groupes de processeurs associés à un processus peuvent être consultés à l'aide de la fonction GetProcessGroupAffinity. Le contrôle du groupe de processeurs d'un thread spécifique est possible avec la fonction SetThreadGroupAffinity.

DLL (*Dynamic Link Library*)

Une large majorité des fonctions rendues visibles aux programmes Windows le sont par l'intermédiaire de bibliothèques de liens dynamiques (DLL, Dynamic Link Libraries). Chaque DLL est constituée d'un ensemble de sous-routines, liées entre elles en tant qu'image binaire, que les applications utilisatrices peuvent charger statiquement ou dynamiquement.

Au sens le plus strict, une DLL est un fichier qui regroupe, en vue de les partager, diverses ressources (fonctions compilées, variables globales, chaînes de caractères, etc.), facilitant de la sorte la mise en commun du code et des données nécessaires à la réalisation de tâches données. Une DLL contient une table des fonctions exportées qui contient :

- les noms symboliques des fonctions
- éventuellement, des entiers appelés nombres ordinaux associés à ces fonction
- les adresses des fonctions dans la DLL

Quand l'application cliente charge la DLL, elle sollicite les fonctions que rend visibles cette dernière par le biais soit des noms symboliques soit des ordinaux. Le processus de liaison dynamique relie les appels du client aux adresses où résident les fonctions au sein de la DLL.

Un fichier exécutable peut être lié à une DLL (ou la charger) soit implicitement soit explicitement. Avec la liaison implicite (parfois qualifiée de chargement statique ou liaison dynamique au moment du chargement), l'exécutable qui utilise la DLL est lié à une bibliothèque d'importation, laquelle contient uniquement le code de chargement de la DLL et d'implémentation d'appels aux fonctions de la DLL.

Lors du chargement d'un programme contenant des références implicites, Windows s'appuie sur les informations que renferme le fichier exécutable du programme pour rechercher les DLL requises. S'il ne les trouve pas, le système interrompt le processus et affiche une boîte de dialogue indiquant l'erreur. Autrement, le système mappe les modules de la DLL dans l'espace d'adressage du processus. Au delà de cette étape, l'exécutable client appelle les fonctions exportées de la DLL comme si ces fonctions étaient contenues dans l'exécutable.

Avec la liaison explicite (parfois désignée par chargement dynamique ou liaison dynamique au moment de l'exécution), les applications doivent effectuer un appel de fonction pour charger explicitement la DLL au moment de l'exécution. Dans cette configuration, une application doit : (1) appeler LoadLibrary pour charger la DLL et ainsi obtenir au retour de ladite fonction un handle de module ; (2) appeler GetProcAddress de sorte à obtenir un pointeur pour chaque fonction que l'application souhaite employer. (Comme les applications sollicitent les fonctions des DLL par l'intermédiaire d'un pointeur, le compilateur ne génère pas de références externes, et il n'est de ce fait pas nécessaire d'établir une liaison avec une bibliothèque d'importation.) ; (3) appeler FreeLibrary quand elle en a fini avec la DLL, cela afin de libérer toutes les ressources qui doivent l'être.

La plupart des application, dans la mesure où il s'agit là de la méthode la plus commode à adopter, emploie la liaison implicite. Diverses raisons courantes, cependant, peuvent légitimer un recours à la liaison explicite, parmi lesquelles celles présentées ci-dessous.

- L'application ne découvre le nom d'une DLL qu'elle doit charger qu'au moment de l'exécution, cela par exemple au travers d'un fichier de configuration, d'un commutateur spécial sur la ligne de commande, de la présence ou de l'état d'une valeur de registre, etc.

- Un processus utilisant la liaison implicite est interrompu par le système d'exploitation si la DLL est introuvable au moment du démarrage du processus. Un processus utilisant la liaison explicite n'est pas interrompu dans ce cas, lui conférant ainsi la possibilité de remédier à un tel dysfonctionnement. Par exemple, le processus peut signaler l'erreur à l'utilisateur et lui demander de spécifier un autre chemin d'accès pour la DLL.

- Un processus qui utilise la liaison implicite est également interrompu si l'une des DLL auxquelles il est lié possède une fonction DllMain qui échoue. Un processus utilisant la liaison explicite n'est pas soumis à de pareilles contraintes.

- Le grand nombre de DLL utilisées par un logiciel nécessite une gestion adaptée de leur chargement en mémoire. Pour accélérer le démarrage, une application peut être liée de manière implicite aux DLL requises immédiatement après le chargement, puis attendre d'être liée explicitement aux autres DLL au fur et à mesure des besoins.

Un exécutable peut utiliser la même DLL en ayant recours à l'une ou l'autre méthode de liaison. De plus, ces mécanismes ne s'excluent pas mutuellement dans la mesure où un exécutable peut être lié de manière implicite à une DLL et un autre exécutable venir s'y attacher de manière explicite.

Que la méthode de liaison d'une application cliente à une DLL soit implicite ou explicite, Windows emploie les emplacements suivants (selon l'ordre indiqué) dans le but de localiser une DLL :

1. Le répertoire hébergeant le module exécutable sous-jacent à l'application.

2. Le répertoire actif.

3. Le répertoire système de Windows, tel qu'obtenu via la fonction GetSystemDirectory (normalement C:\Windows\System).

4. Le répertoire Windows, tel qu'obtenu via la fonction GetWindowsDirectory (normalement C:\Windows).

5. Les répertoires désignés dans la variable d'environnement PATH.

DLL et gestion de la mémoire. Windows actualise un compteur de références pour chaque DLL sur laquelle s'appuie un processus. Lorsqu'un thread du processus charge la DLL, ce compteur est incrémenté d'une unité. Lorsque le processus se termine ou lorsque le nombre de références tombe à zéro (liaison dynamique au moment de l'exécution uniquement), la DLL est supprimée de l'espace d'adressage virtuel du processus.

Fonctions

La liste qui suit énumère les principales fonctions de l'API Windows qui se trouvent en conjonction avec des DLL.

- *AddDllDirectory* Ajoute une entrée au chemin d'accès utilisé par Windows pour localiser une DLL.
- *DisableThreadLibraryCalls* Empêche les notifications de chargement et déchargement de se produire pour une DLL spécifique.
- *DllGetVersion* Retourne des informations de version concernant une DLL.
- *GetModuleFileName* Retourne le chemin complet d'une image binaire.
- *GetProcAddress* Retourne l'adresse d'une fonction ou variable exportée depuis une DLL.
- *LoadLibrary* Charge un module dans l'espace d'adressage virtuel du processus appelant.
- *SetDllDirectory*

DLL connues

Quelques-unes des bibliothèques intégrés à Windows, collectivement appelées DLL connues, bénéficient d'un traitement spécial, et en l'occurence se différencient des autres par le fait que le système d'exploitation les recherche toujours au même emplacement, court-circuitant de la sorte les procédures normales prévues pour la localisation de tels modules.

Au démarrage du système, Windows lit la liste des DLL connues à l'aide des valeurs définies sous la clé HKLM \CurrentControlSet\Control\Session Manager\KnownDLLs, et alimente à partir des informations recueillies alors le répertoire \KnownDLLs, quant à lui défini dans l'espace de noms global du gestionnaire d'objet.

```
lkd> !object \KnownDLLs
Object: ffffb9894d998480  Type: (ffffa7049be74a60) Directory
    ObjectHeader: ffffb9894d998450 (new version)
    HandleCount: 100  PointerCount: 3257689
    Directory Object: ffffb9894c8092a0  Name: KnownDlls

    Hash Address          Type                    Name
    ---- -------          ----                    ----
     00  ffffb9894daa6cd0 Section                 kernel32.dll
     01  ffffb9894daf3850 Section                 kernel.appcore.dll
         ffffb9894daa8650 Section                 windows.storage.dll
     02  ffffb9894daa8710 Section                 ucrtbase.dll
         ffffb9894da63490 Section                 SHLWAPI.dll
         ffffb9894da63910 Section                 WS2_32.dll
         ffffb9894da62710 Section                 MSCTF.dll
     03  ffffb9894daa99d0 Section                 kernelbase.dll
     04  ffffb9894daa6490 Section                 wow64.dll
     05  ffffb9894daa7ed0 Section                 msvcp_win.dll
         ffffb9894da62650 Section                 gdiplus.dll
         ffffb9894da20a70 SymbolicLink            KnownDllPath
     06  ffffb9894daf3790 Section                 MSASN1.dll
         ffffb9894da63550 Section                 user32.dll
     08  ffffb9894daa8110 Section                 bcrypt.dll
     10  ffffb9894daa7990 Section                 COMCTL32.dll
     11  ffffb9894daa7210 Section                 cfgmgr32.dll
     12  ffffb9894daa63d0 Section                 combase.dll
         ffffb9894da0d310 Section                 IMM32.dll
     13  ffffb9894daa7810 Section                 rpcrt4.dll
     14  ffffb9894daa9310 Section                 bcryptPrimitives.dll
         ffffb9894daa8290 Section                 ntdll.dll
         ffffb9894da62350 Section                 coml2.dll
     16  ffffb9894daa8590 Section                 win32u.dll
     17  ffffb9894da60e50 Section                 wow64cpu.dll
     18  ffffb9894da630d0 Section                 COMDLG32.dll
     19  ffffb9894daa8c50 Section                 gdi32full.dll
         ffffb9894da60850 Section                 IMAGEHLP.dll
```

```
    20  ffffb9894da0dc10 Section                  SHELL32.dll
    21  ffffb9894daa60d0 Section                  sechost.dll
    22  ffffb9894daa75d0 Section                  WINTRUST.dll
    24  ffffb9894da61b10 Section                  NORMALIZ.dll
    25  ffffb9894da62a10 Section                  difxapi.dll
    26  ffffb9894daf30d0 Section                  profapi.dll
        ffffb9894da61510 Section                  Setupapi.dll
    27  ffffb9894daf33d0 Section                  CRYPT32.dll
    28  ffffb9894daa6f10 Section                  MSVCRT.dll
        ffffb9894da0dcd0 Section                  gdi32.dll
    29  ffffb9894da62590 Section                  wow64win.dll
        ffffb9894da60cd0 Section                  advapi32.dll
    30  ffffb9894da60b50 Section                  PSAPI.DLL
        ffffb9894da606d0 Section                  NSI.dll
    31  ffffb9894daf3b50 Section                  UMPDC.dll
        ffffb9894daa72d0 Section                  WLDAP32.dll
        ffffb9894da62890 Section                  OLEAUT32.dll
    33  ffffb9894da0db50 Section                  SHCORE.dll
    34  ffffb9894da63190 Section                  ole32.dll
    35  ffffb9894da60c10 Section                  clbcatq.dll
    36  ffffb9894daf4210 Section                  powrprof.dll
        ffffb9894daa96d0 Section                  cryptsp.dll
```

Visualisation des DLL chargées

Pour obtenir des informations en ce qui concerne les bibliothèques (DLL) en cours d'utilisation sur la station de travail locale, employez l'utilitaire ListDLLs. Vous pourrez de la sorte voir quelles DLL sont actuellement chargées, depuis quels emplacements du système de fichiers, lesquelles ont été mises à jour/déplacées, etc. Une autre fonctionnalité particulièrement utile de l'application ListDLLs tient au fait qu'elle indique si la version d'une DLL chargée en mémoire diffère de celle présente sur le disque.

ListDLLs requiert des droits administratifs, y compris le privilege Déboguer des programmes (SeDebugPrivilege), seulement pour répertorier les DLL contenues dans des processus exécutés en tant qu'un utilisateur différent ou à un niveau d'intégrité supérieur. De telles exigences ne sont pas demandées pour les processus s'exécutant sous le même utilisateur et au même niveau d'intégrité ou à un niveau inférieur.

ListDLLs fonctionne en ligne de commande et honore la syntaxe suivante :

```
listdlls [-r] [processname | PID | -d dllname]
```

Procédez par exemple comme suit afin de voir quelles DLL héberge le processus Bloc-Notes.

```
C:\>Listdlls.exe notepad

Notepad.exe pid: 10660
Command line: "C:\Program Files\WindowsApps\Microsoft.WindowsNotepad_11.2312.18.0_x64__8wekyb3d8bbwe\Notepad\Notepad.exe"

Base                Size      Path
0x0000000072210000  0x139000  C:\Program Files\WindowsApps\Microsoft.WindowsNotepad_11.2312.18.0_x64__8wekyb3d8bbwe\Notepad
\Notepad.exe
0x00000000be570000  0x217000  C:\WINDOWS\SYSTEM32\ntdll.dll
0x00000000bd5f0000  0xc4000   C:\WINDOWS\System32\KERNEL32.DLL
0x00000000bbcb0000  0x3a6000  C:\WINDOWS\System32\KERNELBASE.dll
0x00000000bcf70000  0x5e000   C:\WINDOWS\System32\SHLWAPI.dll
0x00000000bd750000  0xa7000   C:\WINDOWS\System32\msvcrt.dll
0x00000000be230000  0x1ae000  C:\WINDOWS\System32\USER32.dll
0x00000000bb8b0000  0x26000   C:\WINDOWS\System32\win32u.dll
0x00000000bddb0000  0x29000   C:\WINDOWS\System32\GDI32.dll
...
```

En ajoutant l'option -d à la ligne de commande de listdlls.exe, suivie du nom complet ou partiel d'une DLL, vous obtenez la liste des processus ayant chargé ledit module. Dans un tel scénario, ListDLLs génère une liste des

processus actifs et procède pour chacun d'entre eux à un inventaire des DLL employées. Si le nom spécifié lors de la recherche apparait n'importe où dans le chemin complet d'une DLL chargée, ListDLLs fait alors apparaitre les informations appropriées. Par exemple, pour rechercher tous les processus qui ont chargé kernel32.dll, exécutez la commande listdlls -d kernel32.

Point d'entrée DllMain

Comme les programmes, chaque bibliothèque sous Windows (DLL) encapsule une fonction spéciale, en l'occurrence nommée `DllMain`, à laquelle le système passe le relais chaque fois qu'un processus ou un thread charge ou décharge ledit module, créant ainsi la possibilité pour l'appelant de mener à bien diverses tâches, par exemple initialiser ou démanteler une ressource.

La signature de la fonction DllMain se présente comme suit :

```
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved);
```

Le premier paramètre correspond à l'adresse où réside la DLL dans l'espace d'adressage virtuel du processus actuel. Le second indique la raison pour laquelle la fonction DllMain a été sollicité (voir plus loin). Enfin, le troisième et dernier paramètre apporte des informations complémentaires sur les circonstances ayant entouré l'opération.

Sur le plan pratique, la fonction DllMain est conditionnelle de l'une ou l'autre des circonstances que voici :

- Un processus charge une première fois la DLL. Les charges ultérieures, par exemple lors de multiples liaisons à la demande (LoadLibrary), n'entraînent pas d'appel supplémentaire à DllMain.
- Un processus décharge la DLL. Cela peut être dû au fait que le processus prend fin ou que le nombre d'instances de chargement de la DLL - tel que mis à jour par la fonction FreeLibrary - atteint zéro. En cas de terminaison brutale du processus (TerminateProcess), la fonction DllMain ne s'exécute pas pour effectuer les opérations de nettoyage normalement attendues.
- Un processus crée un thread. Dans ce cas, le système appelle la fonction point d'entrée de l'ensemble des DLL actuellement attachées au processus. L'appel est effectué dans le contexte du nouveau thread.
- Un thread d'un processus ayant chargé la DLL se termine.

Dans l'éventualité où une DLL n'exige pas d'être informée des événements liés à la création ou à la destruction des threads dans un processus donné, une optimisation utile est envisageable au travers de la fonction DisableThreadLibraryCalls.

Communication inter processus

Une large majorité des fonctions prises en charge par Windows le sont par l'intermédiaire de processus coopérants (voire dans quelques cas, distants), qui agissent en interaction les uns avec les autres et oeuvrent ainsi ensemble pour la réalisation d'un algorithme. A titre d'exemple, dans le cadre d'une opération a priori aussi sommaire que l'ouverture d'un fichier, échangent entre eux, entre autres, le noyau, le gestionnaire de cache, le gestionnaire de mémoire, le sous-système d'E/S, et le pilote de système de fichiers, tous se réclamant à divers stades des services. Le sous-système d'environnement standard (Csrss.exe), la gestion de la sécurité (Lsass.exe), le contrôle des services (Services.exe), les files d'impression (Spoolsv.exe), les systèmes de fichiers en réseau, et bien d'autres caractéristiques sont implémentés selon ce modèle, dans laquelle une entité est collaboratrice d'une autre. Les raisons d'entretenir une telle approche sont multiples : (1) partage d'informations, (2) réduction de l'empreinte mémoire, (3) accélération des traitements, (4) modularité, (5) facilité d'utilisation, (6) séparation des privilèges. En marge de cette complicité, le système d'exploitation doit par conséquent fournir différentes techniques pour l'échange de données entre unités fonctionnelles d'exécution. Collectivement appelés les communications inter-processus (IPC, Inter Process Communication), les activités permises par ces mécanismes tendent à faciliter la division des tâches, soit à l'échelle de l'environnement local (plusieurs processus/threads), soit à l'échelle d'environnements connectés (entre les ordinateurs d'un réseau).

Canaux de transmission (*pipes*)

Les canaux de transmission, ou tubes (*pipes*), constituent un mécanisme standard de communication entre processus, issu du monde Unix.

Tubes anonymes et tubes nommés

Les tubes gérés dans la perspective Windows sont de deux sortes, selon qu'ils soient ou non nommés.

Un tube anonyme permet de chainer deux processus, via une zone mémoire, de telle sorte que les données en sortie de l'un constituent les données en entrée de l'autre. La communication y est uni unidirectionnelle, et sert généralement au dialogue entre un processus parent et ses fils, dans la mesure où eux seuls connaissent son existence. Une fois le processus faisant appel au tube terminé, le tube anonyme est automatiquement détruit.

Un tube nommé, matérialisé parmi les ressources globales du système d'exploitation, permet de faire communiquer des processus sans lien de parenté entre eux. La communication dans le cadre de tubes nommés peut être unidirectionnel ou bidirectionnel.

Windows enracine les canaux de transmissions, nommés ou non, dans le système de fichiers. C'est pourquoi hormis leur création (CreatePipe et CreateNamedPipe), les opérations en ce qui les concerne sont prises en charge non par un jeu fonctions spécialisées, mais par les fonctions usuelles d'E/S fichier, incluant par exemple *CreateFile*, *WriteFile*, *ReadFile*, et consorts.

Opérations sur des tubes nommés

Windows stocke les tubes nommés dans un système de fichiers ah-hoc hébergé en mémoire, via le pilote NPFS.

FILE_PIPE_INFORMATION.

```
typedef struct _FILE_PIPE_INFORMATION {
  ULONG ReadMode;
  ULONG CompletionMode;
} FILE_PIPE_INFORMATION, *PFILE_PIPE_INFORMATION;
```

FILE_PIPE_LOCAL_INFORMATION.

```
typedef struct _FILE_PIPE_LOCAL_INFORMATION {
  ULONG NamedPipeType;
  ULONG NamedPipeConfiguration;
  ULONG MaximumInstances;
  ULONG CurrentInstances;
  ULONG InboundQuota;
  ULONG ReadDataAvailable;
  ULONG OutboundQuota;
  ULONG WriteQuotaAvailable;
  ULONG NamedPipeState;
  ULONG NamedPipeEnd;
} FILE_PIPE_LOCAL_INFORMATION, *PFILE_PIPE_LOCAL_INFORMATION;
```

Passage de messages

Une forme de communication inter processus particulièrement prisée des applications basées sur des fenêtres consiste à employer des messages.

Un thread peut envoyer un message à un autre thread ou à une fenêtre en faisant appel à l'une ou l'autre des fonctions suivantes : GetMessage, PeekMessage, PostQuitMessage, PostThreadMessage, RegisterWindowMessage, SendMessageCallback, SendNotifyMessage.

La différence entre un *postage* et un *envoi* est que les routine de postages sont asynchrones. Elles rendent aussitôt la main et peuvent de ce fait être rappelées avant même l'issue de la requête à laquelle elles donnent

lieu (autrement dit avant que l'opération correspondante se soit achevée). Les routines d'envoi, quant à elles, sont synchrones, et bloquent en l'occurence le thread appelant jusqu'à ce que le message ait été reçu et traité.

Tableau 8.47. Codes de message Windows

0x0000	WM_NULL
0x0001	WM_CREATE
0x0002	WM_DESTROY
0x0003	WM_MOVE
0x0005	WM_SIZE
0x0006	WM_ACTIVATE
0x0007	WM_SETFOCUS
0x0008	WM_KILLFOCUS
0x000A	WM_ENABLE
0x000B	WM_SETREDRAW
0x000C	WM_SETTEXT
0x000D	WM_GETTEXT
0x000E	WM_GETTEXTLENGTH
0x000F	WM_PAINT
0x0010	WM_CLOSE
0x0011	WM_QUERYENDSESSION
0x0013	WM_QUERYOPEN
0x0016	WM_ENDSESSION
0x0012	WM_QUIT
0x0014	WM_ERASEBKGND
0x0015	WM_SYSCOLORCHANGE
0x0018	WM_SHOWWINDOW
0x001A	WM_WININICHANGE
0x001A	WM_SETTINGCHANGE
0x001B	WM_DEVMODECHANGE
0x001C	WM_ACTIVATEAPP
0x001D	WM_FONTCHANGE
0x001E	WM_TIMECHANGE
0x001F	WM_CANCELMODE
0x0020	WM_SETCURSOR
0x0021	WM_MOUSEACTIVATE
0x0022	WM_CHILDACTIVATE
0x0023	WM_QUEUESYNC
0x0024	WM_GETMINMAXINFO
0x0026	WM_PAINTICON
0x0027	WM_ICONERASEBKGND
0x0028	WM_NEXTDLGCTL
0x002A	WM_SPOOLERSTATUS
0x002B	WM_DRAWITEM
0x002C	WM_MEASUREITEM
0x002D	WM_DELETEITEM
0x002E	WM_VKEYTOITEM
0x002F	WM_CHARTOITEM
0x0030	WM_SETFONT
0x0031	WM_GETFONT

0x0032	WM_SETHOTKEY
0x0033	WM_GETHOTKEY
0x0037	WM_QUERYDRAGICON
0x0039	WM_COMPAREITEM
0x003D	WM_GETOBJECT
0x0041	WM_COMPACTING
0x0044	WM_COMMNOTIFY
0x0046	WM_WINDOWPOSCHANGING
0x0047	WM_WINDOWPOSCHANGED
0x0048	WM_POWER
0x004A	WM_COPYDATA
0x004B	WM_CANCELJOURNAL
0x004E	WM_NOTIFY
0x0050	WM_INPUTLANGCHANGEREQUEST
0x0051	WM_INPUTLANGCHANGE
0x0052	WM_TCARD
0x0053	WM_HELP
0x0054	WM_USERCHANGED
0x0055	WM_NOTIFYFORMAT
0x007B	WM_CONTEXTMENU
0x007C	WM_STYLECHANGING
0x007D	WM_STYLECHANGED
0x007E	WM_DISPLAYCHANGE
0x007F	WM_GETICON
0x0080	WM_SETICON
0x0081	WM_NCCREATE
0x0082	WM_NCDESTROY
0x0083	WM_NCCALCSIZE
0x0084	WM_NCHITTEST
0x0085	WM_NCPAINT
0x0086	WM_NCACTIVATE
0x0087	WM_GETDLGCODE
0x0088	WM_SYNCPAINT
0x00A0	WM_NCMOUSEMOVE
0x00A1	WM_NCLBUTTONDOWN
0x00A2	WM_NCLBUTTONUP
0x00A3	WM_NCLBUTTONDBLCLK
0x00A4	WM_NCRBUTTONDOWN
0x00A5	WM_NCRBUTTONUP
0x00A6	WM_NCRBUTTONDBLCLK
0x00A7	WM_NCMBUTTONDOWN
0x00A8	WM_NCMBUTTONUP
0x00A9	WM_NCMBUTTONDBLCLK
0x00AB	WM_NCXBUTTONDOWN
0x00AC	WM_NCXBUTTONUP
0x00AD	WM_NCXBUTTONDBLCLK
0x00FF	WM_INPUT
0x0100	WM_KEYFIRST

0x0100	WM_KEYDOWN
0x0101	WM_KEYUP
0x0102	WM_CHAR
0x0103	WM_DEADCHAR
0x0104	WM_SYSKEYDOWN
0x0105	WM_SYSKEYUP
0x0106	WM_SYSCHAR
0x0107	WM_SYSDEADCHAR
0x0109	WM_KEYLAST
0x0109	WM_UNICHAR
0x010D	WM_IME_STARTCOMPOSITION
0x010E	WM_IME_ENDCOMPOSITION
0x010F	WM_IME_COMPOSITION
0x010F	WM_IME_KEYLAST
0x0110	WM_INITDIALOG
0x0111	WM_COMMAND
0x0112	WM_SYSCOMMAND
0x0113	WM_TIMER
0x0114	WM_HSCROLL
0x0115	WM_VSCROLL
0x0116	WM_INITMENU
0x0117	WM_INITMENUPOPUP
0x011F	WM_MENUSELECT
0x0120	WM_MENUCHAR
0x0121	WM_ENTERIDLE
0x0122	WM_MENURBUTTONUP
0x0123	WM_MENUDRAG
0x0124	WM_MENUGETOBJECT
0x0125	WM_UNINITMENUPOPUP
0x0126	WM_MENUCOMMAND
0x0127	WM_CHANGEUISTATE
0x0128	WM_UPDATEUISTATE
0x0129	WM_QUERYUISTATE
0x0132	WM_CTLCOLORMSGBOX
0x0133	WM_CTLCOLOREDIT
0x0134	WM_CTLCOLORLISTBOX
0x0135	WM_CTLCOLORBTN
0x0136	WM_CTLCOLORDLG
0x0137	WM_CTLCOLORSCROLLBAR
0x0138	WM_CTLCOLORSTATIC
0x0200	WM_MOUSEFIRST
0x0200	WM_MOUSEMOVE
0x0201	WM_LBUTTONDOWN
0x0202	WM_LBUTTONUP
0x0203	WM_LBUTTONDBLCLK
0x0204	WM_RBUTTONDOWN
0x0205	WM_RBUTTONUP
0x0206	WM_RBUTTONDBLCLK

0x0207	WM_MBUTTONDOWN
0x0208	WM_MBUTTONUP
0x0209	WM_MBUTTONDBLCLK
0x0209	WM_MOUSELAST(95)
0x020A	WM_MOUSEWHEEL
0x020A	WM_MOUSELAST(NT4,98)
0x020B	WM_XBUTTONDOWN
0x020C	WM_XBUTTONUP
0x020D	WM_XBUTTONDBLCLK
0x020D	WM_MOUSELAST(2K,XP,2k3)
0x0210	WM_PARENTNOTIFY
0x0211	WM_ENTERMENULOOP
0x0212	WM_EXITMENULOOP
0x0213	WM_NEXTMENU
0x0214	WM_SIZING
0x0215	WM_CAPTURECHANGED
0x0216	WM_MOVING
0x0218	WM_POWERBROADCAST
0x0219	WM_DEVICECHANGE
0x0220	WM_MDICREATE
0x0221	WM_MDIDESTROY
0x0222	WM_MDIACTIVATE
0x0223	WM_MDIRESTORE
0x0224	WM_MDINEXT
0x0225	WM_MDIMAXIMIZE
0x0226	WM_MDITILE
0x0227	WM_MDICASCADE
0x0228	WM_MDIICONARRANGE
0x0229	WM_MDIGETACTIVE
0x0230	WM_MDISETMENU
0x0231	WM_ENTERSIZEMOVE
0x0232	WM_EXITSIZEMOVE
0x0233	WM_DROPFILES
0x0234	WM_MDIREFRESHMENU
0x0281	WM_IME_SETCONTEXT
0x0282	WM_IME_NOTIFY
0x0283	WM_IME_CONTROL
0x0284	WM_IME_COMPOSITIONFULL
0x0285	WM_IME_SELECT
0x0286	WM_IME_CHAR
0x0288	WM_IME_REQUEST
0x0290	WM_IME_KEYDOWN
0x0291	WM_IME_KEYUP
0x02A1	WM_MOUSEHOVER
0x02A3	WM_MOUSELEAVE
0x02A0	WM_NCMOUSEHOVER
0x02A2	WM_NCMOUSELEAVE
0x02B1	WM_WTSSESSION_CHANGE

0x02C0	WM_TABLET_FIRST
0x02DF	WM_TABLET_LAST
0x0300	WM_CUT
0x0301	WM_COPY
0x0302	WM_PASTE
0x0303	WM_CLEAR
0x0304	WM_UNDO
0x0305	WM_RENDERFORMAT
0x0306	WM_RENDERALLFORMATS
0x0307	WM_DESTROYCLIPBOARD
0x0308	WM_DRAWCLIPBOARD
0x0309	WM_PAINTCLIPBOARD
0x030A	WM_VSCROLLCLIPBOARD
0x030B	WM_SIZECLIPBOARD
0x030C	WM_ASKCBFORMATNAME
0x030D	WM_CHANGECBCHAIN
0x030E	WM_HSCROLLCLIPBOARD
0x030F	WM_QUERYNEWPALETTE
0x0310	WM_PALETTEISCHANGING
0x0311	WM_PALETTECHANGED
0x0312	WM_HOTKEY
0x0317	WM_PRINT
0x0318	WM_PRINTCLIENT
0x0319	WM_APPCOMMAND
0x031A	WM_THEMECHANGED
0x0358	WM_HANDHELDFIRST
0x035F	WM_HANDHELDLAST
0x0360	WM_AFXFIRST
0x037F	WM_AFXLAST
0x0380	WM_PENWINFIRST
0x038F	WM_PENWINLAST
0x0400	WM_USER
0x8000	WM_APP

Chaque thread graphique Windows dispose de sa propre file d'attente d'entrée par laquelle il reçoit et identifie des messages. Une telle façon de procéder est plus fiable que le partage d'une même file d'attente, dans la mesure où une application bloquée n'influence pas négativement les entrées des autres applications.

Lorsque la fenêtre de niveau supérieur d'une application montre des signes de faiblesse en ce qui concerne la gestion des messages (cela se traduisant en interne par le fait qu'elle ne soit pas parvenu à interroger, avec GetMessage ou PeekMessage par exemple, les événements de sa propre file d'attente pendant un laps de temps supérieur à 5 secondes - délai mis en avant au sein de la fonction IsHungAppWindow), le système masque ladite fenêtre et la remplace par une autre, alors qualifiée de fantôme et pourvue des mêmes caractéristiques (emplacement, taille, profondeur, attributs visuels). Cela laisse à l'utilisateur l'occasion de déplacer, redimensionner voire même fermer l'application. (Seules ces actions sont disponibles.)

WM_COPYDATA

Un processus peut expédier des données à un autre processus en appelant la fonction SendMessage pour envoyer un message de type `WM_COPYDATA` avec une structure de données COPYDATASTRUCT contenant la longueur et l'adresse des données à transférer. Comme chaque processus a son propre espace d'adressage

privé, le système effectue en pareil cas une copie des données sortantes dans un bloc mémoire nouvellement alloué et donne l'adresse virtuelle dudit bloc au processus destinataire. La structure COPYDATASTRUCT se présente comme suit :

```
typedef struct _COPYDATASTRUCT {
  ULONG_PTR dwData;
  DWORD cbData;
  PVOID lpData;
} COPYDATASTRUCT, *PCOPYDATASTRUCT;
```

Fibres

En contrepoint de l'ordonnancement à base de priorités intégré à Windows, les fibres permettent à une application de planifier ses propres unités fonctionnelles et, ce faisant, choisir l'ordre et le moment où elles s'exécutent. Elles constituent de la sorte une forme de multitâche coopératif - par contraste avec celui piloté par le système, quant à lui préemptif.

Essentiellement proche, comme notion, des threads, les fibres se distinguent de ces derniers par le fait d'être gérées entièrement au niveau applicatif. En référence à ce phénomène, on parle quelquefois à propos des fibres de threads "allégés", dans le sens où elles sont invisibles au noyau, implémentées en l'occurrence en totalité dans le mode utilisateur.

Le système crée une fibre suite à l'appel de deux fonctions : ConvertThreadToFiber, laquelle convertit un thread en une fibre exécutée, et CreateFiber, elle utilisée afin de donner lieu à une fibre à partir d'une autre déjà existante. (Chaque fibre peut avoir son propre jeu de fibres.)

Contrairement à un thread, une fibre n'est pas éligible pour l'exécution dès la naissance, mais doit pour le devenir être manuellement sélectionnée via à un appel à la fonction SwitchToFiber. La nouvelle fibre est exécutée jusqu'à ce qu'elle se termine ou jusqu'à ce qu'elle sollicite SwitchToFiber pour sélectionner une autre fibre.

Dans la perspective du système d'exploitation, chaque fibre emprunte l'identité et partage les ressources du thread l'exécutant. Les fibres ne présentent toutefois pas les mêmes attributs que des threads, cela dans la mesure où seules quelques informations d'état sont préservées lors du passage d'un thread en fibre. Cela inclut la pile, un sous-ensemble des registres processeur et les données fournies à la création.

Les fibres ne sont pas planifiées de façon préemptive. Lorsqu'un thread qui fait fonctionner une fibre est préempté, sa fibre actuelle l'est également, mais reste sélectionnée. Elle reprendra son exécution en même temps que le fil d'exécution qui l'accompagne.

Débogage

Fonction	Description
FatalExit	Transfère le contrôle de l'exécution au débogueur.
FlushInstructionCache	Vide le cache d'instructions d'un processus
GetThreadContext	Retourne les valeurs des registres processeur d'un thread
GetThreadSelectorEntry	Retourne l'entrée de table de descripteurs associée à un thread (ne s'applique qu'aux systèmes x86).
IsDebuggerPresent	Détermine si le processus appelant est sous contrôle d'un débogueur.
OutputDebugString	Envoie une chaîne au débogueur pour l'affichage.
ReadProcessMemory	Accède à la mémoire d'un processus
SetThreadContext	Modifie les valeurs des registres processeur d'un thread
WaitForDebugEvent	Attendre sur un événement de débogage
WriteProcessMemory	Ecrit dans la mémoire d'un processus

Groupe de processus (Objets job)

Outre processus et threads, Windows met entre les mains des développeurs et administrateurs système un modèle additionnel d'organisation des ressources, les *travaux*, qui permettent de gérer et manipuler comme un tout un groupe de processus.

Chaque objet job est un objet noyau nommable, sécurisable et partageable qui contrôle les attributs des processus lui étant associés. Toutes les actions et opérations effectuées sur un objet job affectent tous les processus du job. Les exemples incluent l'application de limites telles que le minimum et maximum de working set, l'affinité avec un processeur, ou encore la capacité de mettre fin à tous les processus du job. L'objet job contient aussi des données basiques de comptabilisation sur tous les processus associés au job, y compris ceux terminés.

La liste qui suit dresse un bref portrait des fonctions Windows utilisables en matière de gestion des objets job :

- *CreateJobObject* Crée un job. L'objet résultant peut être nommé ou non.
- *OpenJobObject* Ouvre par son nom un objet job.
- *AssignProcessToJobObject* Ajoute un processus à un job.
- *TerminateJobObject* Met fin à l'ensemble des processus dans un job.
- *SetInformationJobObject* Définit des limites.
- *QueryInformationJobObject* Donne des informations sur le job.

Les jobs ont pour fonction principale l'application aux processus de certaines limites dans l'utilisation des ressources, par exemple la quantité de temps processeur qui peut être consommée ou la quantité d'espace d'adressage qui peut être réservée. Voici quelques-uns des limites qu'il est possible de spécifier pour un job.

- *Nombre maximal de processus actifs* Définit le nombre maximal de processus susceptibles d'exister simultanément dans le job.
- *Limite globale de temps processeur mode utilisateur* Limite la quantité maximale de temps processeur mode utilisateur que les processus du job peuvent consommer (cela comprend les processus qui ont terminé). Une fois cette limite atteinte, par défaut tous les processus du job prennent automatiquement fin et plus aucun nouveau processus ne peut intégrer le job.
- *Limite par processus de temps processeur mode utilisateur* Permet à chaque processus qui compose un job de ne consommer qu'un montant maximal fixé de temps processeur mode utilisateur. Quand le maximum est atteint, le processus se termine.
- *Affinité processeur du job* Définit le masque d'affinité de processeur pour chaque processus du job. L'affinité ainsi spécifiée doit être un sous-ensemble du masque d'affinité du système (voir fonction GetProcessAffinityMask). Les processus ne peuvent pas modifier leur d'affinité. Les threads, eux, restent libres de le faire pourvu que l'affinité soit un sous-ensemble de l'affinité du job.
- *Limite de mémoire virtuelle réservée au processus et au job* Définit la quantité maximale d'espace d'adressage qui peut être réservée soit par un processus individuel, soit par l'ensemble des processus d'un job.
- *Classe d'ordonnancement du job* Définit la longueur de la tranche de temps (ou quantum) des threads des processus du job.
- *Limite d'interface utilisateur*

Constante	Valeur	Signification
JOB_OBJECT_UILIMIT_DESKTOP	0x00000040	Empêche les processus de créer des bureaux et de changer de bureau via, respectivement, les fonctions CreateDesktop et SwitchDesktop.

Constante	Valeur	Signification
JOB_OBJECT_UILIMIT_DISPLAYSETTINGS	0x00000010	Empêche les processus de modifier les paramètres d'affichage via la fonction Windows SystemParametersInfo.
JOB_OBJECT_UILIMIT_EXITWINDOWS	0x00000080	Empêche les processus d'initier une procédure de déconnexion de session via les fonctions Windows ExitWindows et ExitWindowsEx.
JOB_OBJECT_UILIMIT_GLOBALATOMS	0x00000020	Empêche les processus d'accéder aux atomes globaux.
JOB_OBJECT_UILIMIT_HANDLES	0x00000001	Empêche les processus d'ouvrir des handles vers des fenêtres possédées par des threads extérieurs au job.
JOB_OBJECT_UILIMIT_READCLIPBOARD	0x00000002	Empêche les processus de lire dans le presse-papiers.
JOB_OBJECT_UILIMIT_-SYSTEMPARAMETERS	0x00000008	Empêche les processus de modifier les paramètres système d'interface utilisateur via la fonction Windows SystemParametersInfo.
JOB_OBJECT_UILIMIT_WRITECLIPBOARD	0x00000004	Empêche les processus d'écrire dans le presse-papiers.

Visualisation du format de l'objet job. Utilisez la commande *dt* du débogueur noyau pour visualiser les données internes maintenues dans un objet job. Ce qui suit présente un exemple de résultat de la commande :

```
lkd> dt nt!_EJOB
   +0x000 Event            : _KEVENT
   +0x018 JobLinks         : _LIST_ENTRY
   +0x028 ProcessListHead  : _LIST_ENTRY
   +0x038 JobLock          : _ERESOURCE
   +0x0a0 TotalUserTime    : _LARGE_INTEGER
   +0x0a8 TotalKernelTime  : _LARGE_INTEGER
   +0x0b0 TotalCycleTime   : _LARGE_INTEGER
   +0x0b8 ThisPeriodTotalUserTime : _LARGE_INTEGER
   +0x0c0 ThisPeriodTotalKernelTime : _LARGE_INTEGER
   +0x0c8 TotalContextSwitches : Uint8B
   +0x0d0 TotalPageFaultCount : Uint4B
   +0x0d4 TotalProcesses   : Uint4B
   +0x0d8 ActiveProcesses  : Uint4B
   +0x0dc TotalTerminatedProcesses : Uint4B
   +0x0e0 PerProcessUserTimeLimit : _LARGE_INTEGER
   +0x0e8 PerJobUserTimeLimit : _LARGE_INTEGER
   +0x0f0 MinimumWorkingSetSize : Uint8B
   +0x0f8 MaximumWorkingSetSize : Uint8B
   +0x100 LimitFlags       : Uint4B
   +0x104 ActiveProcessLimit : Uint4B
   +0x108 Affinity         : _KAFFINITY_EX
   +0x1b0 AccessState      : Ptr64 _JOB_ACCESS_STATE
   +0x1b8 AccessStateQuotaReference : Ptr64 Void
   +0x1c0 UIRestrictionsClass : Uint4B
   +0x1c4 EndOfJobTimeAction : Uint4B
   +0x1c8 CompletionPort   : Ptr64 Void
   +0x1d0 CompletionKey    : Ptr64 Void
   +0x1d8 CompletionCount  : Uint8B
   +0x1e0 SessionId        : Uint4B
   +0x1e4 SchedulingClass  : Uint4B
   +0x1e8 ReadOperationCount : Uint8B
   +0x1f0 WriteOperationCount : Uint8B
   +0x1f8 OtherOperationCount : Uint8B
   +0x200 ReadTransferCount : Uint8B
   +0x208 WriteTransferCount : Uint8B
   +0x210 OtherTransferCount : Uint8B
   +0x218 DiskIoInfo       : _PROCESS_DISK_COUNTERS
```

```
+0x240 ProcessMemoryLimit : Uint8B
+0x248 JobMemoryLimit   : Uint8B
+0x250 JobTotalMemoryLimit : Uint8B
+0x258 PeakProcessMemoryUsed : Uint8B
+0x260 PeakJobMemoryUsed : Uint8B
+0x268 EffectiveAffinity : _KAFFINITY_EX
+0x310 EffectivePerProcessUserTimeLimit : _LARGE_INTEGER
+0x318 EffectiveMinimumWorkingSetSize : Uint8B
+0x320 EffectiveMaximumWorkingSetSize : Uint8B
+0x328 EffectiveProcessMemoryLimit : Uint8B
+0x330 EffectiveProcessMemoryLimitJob : Ptr64 _EJOB
+0x338 EffectivePerProcessUserTimeLimitJob : Ptr64 _EJOB
+0x340 EffectiveNetIoRateLimitJob : Ptr64 _EJOB
+0x348 EffectiveHeapAttributionJob : Ptr64 _EJOB
+0x350 EffectiveLimitFlags : Uint4B
+0x354 EffectiveSchedulingClass : Uint4B
+0x358 EffectiveFreezeCount : Uint4B
+0x35c EffectiveBackgroundCount : Uint4B
+0x360 EffectiveSwapCount : Uint4B
+0x364 EffectiveNotificationLimitCount : Uint4B
+0x368 EffectivePriorityClass : UChar
+0x369 PriorityClass    : UChar
+0x36a NestingDepth     : UChar
+0x36b Reserved1        : [1] UChar
+0x36c CompletionFilter : Uint4B
+0x370 WakeChannel      : _WNF_STATE_NAME
+0x370 WakeInfo         : _PS_WAKE_INFORMATION
+0x3a8 WakeFilter       : _JOBOBJECT_WAKE_FILTER
+0x3b0 LowEdgeLatchFilter : Uint4B
+0x3b4 OwnedHighEdgeFilters : Uint4B
+0x3b8 NotificationLink : Ptr64 _EJOB
+0x3c0 CurrentJobMemoryUsed : Uint8B
+0x3c8 NotificationInfo : Ptr64 _JOB_NOTIFICATION_INFORMATION
+0x3d0 NotificationInfoQuotaReference : Ptr64 Void
+0x3d8 NotificationPacket : Ptr64 _IO_MINI_COMPLETION_PACKET_USER
+0x3e0 CpuRateControl   : Ptr64 _JOB_CPU_RATE_CONTROL
+0x3e8 EffectiveSchedulingGroup : Ptr64 Void
+0x3f0 ReadyTime        : Uint8B
+0x3f8 MemoryLimitsLock : _EX_PUSH_LOCK
+0x400 SiblingJobLinks  : _LIST_ENTRY
+0x410 ChildJobListHead : _LIST_ENTRY
+0x420 ParentJob        : Ptr64 _EJOB
+0x428 ParentSilo       : Ptr64 _EJOB
+0x430 RootJob          : Ptr64 _EJOB
+0x438 IteratorListHead : _LIST_ENTRY
+0x448 AncestorCount    : Uint8B
+0x450 Ancestors        : Ptr64 Ptr64 _EJOB
+0x450 SessionObject    : Ptr64 Void
+0x458 TimerListLock    : Uint8B
+0x460 TimerListHead    : _LIST_ENTRY
+0x470 Accounting       : _EPROCESS_VALUES
+0x4c8 ShadowActiveProcessCount : Uint4B
+0x4cc ActiveAuxiliaryProcessCount : Uint4B
+0x4d0 SequenceNumber   : Uint4B
+0x4d4 JobId            : Uint4B
+0x4d8 ContainerId      : _GUID
+0x4e8 ServerSiloGlobals : Ptr64 _ESERVERSILO_GLOBALS
+0x4f0 PropertySet      : _PS_PROPERTY_SET
+0x508 Storage          : Ptr64 _PSP_STORAGE
+0x510 NetRateControl   : Ptr64 _JOB_NET_RATE_CONTROL
+0x518 JobFlags         : Uint4B
+0x518 CloseDone        : Pos 0, 1 Bit
+0x518 MultiGroup       : Pos 1, 1 Bit
+0x518 OutstandingNotification : Pos 2, 1 Bit
+0x518 NotificationInProgress : Pos 3, 1 Bit
```

```
+0x518 UILimits         : Pos 4, 1 Bit
+0x518 CpuRateControlActive : Pos 5, 1 Bit
+0x518 OwnCpuRateControl : Pos 6, 1 Bit
+0x518 Terminating      : Pos 7, 1 Bit
+0x518 WorkingSetLock   : Pos 8, 1 Bit
+0x518 JobFrozen        : Pos 9, 1 Bit
+0x518 Background       : Pos 10, 1 Bit
+0x518 WakeNotificationAllocated : Pos 11, 1 Bit
+0x518 WakeNotificationEnabled : Pos 12, 1 Bit
+0x518 WakeNotificationPending : Pos 13, 1 Bit
+0x518 LimitNotificationRequired : Pos 14, 1 Bit
+0x518 ZeroCountNotificationRequired : Pos 15, 1 Bit
+0x518 CycleTimeNotificationRequired : Pos 16, 1 Bit
+0x518 CycleTimeNotificationPending : Pos 17, 1 Bit
+0x518 TimersVirtualized : Pos 18, 1 Bit
+0x518 JobSwapped       : Pos 19, 1 Bit
+0x518 ViolationDetected : Pos 20, 1 Bit
+0x518 EmptyJobNotified : Pos 21, 1 Bit
+0x518 NoSystemCharge   : Pos 22, 1 Bit
+0x518 DropNoWakeCharges : Pos 23, 1 Bit
+0x518 NoWakeChargePolicyDecided : Pos 24, 1 Bit
+0x518 NetRateControlActive : Pos 25, 1 Bit
+0x518 OwnNetRateControl : Pos 26, 1 Bit
+0x518 IoRateControlActive : Pos 27, 1 Bit
+0x518 OwnIoRateControl : Pos 28, 1 Bit
+0x518 DisallowNewProcesses : Pos 29, 1 Bit
+0x518 Silo             : Pos 30, 1 Bit
+0x518 Spare            : Pos 31, 1 Bit
+0x51c EffectiveHighEdgeFilters : Uint4B
+0x520 EnergyValues     : Ptr64 _PROCESS_ENERGY_VALUES
+0x528 SharedCommitCharge : Uint8B
+0x530 WakeRoot         : Ptr64 _EJOB
+0x538 DiskIoAttributionUserRefCount : Uint4B
+0x53c DiskIoAttributionRefCount : Uint4B
+0x540 DiskIoAttributionContext : Ptr64 Void
+0x540 DiskIoAttributionOwnerJob : Ptr64 _EJOB
+0x548 IoRateControlHeader : _JOB_RATE_CONTROL_HEADER
+0x570 GlobalIoControl  : _PS_IO_CONTROL_ENTRY
+0x5a8 IoControlStateLock : Int4B
+0x5b0 VolumeIoControlTree : _RTL_RB_TREE
+0x5c0 IoRateOverQuotaHistory : Uint8B
+0x5c8 IoRateCurrentGeneration : Uint4B
+0x5cc IoRateLastQueryGeneration : Uint4B
+0x5d0 IoRateGenerationLength : Uint4B
+0x5d4 IoRateOverQuotaNotifySequenceId : Uint4B
+0x5d8 IoControlLock    : _EX_PUSH_LOCK
+0x5e0 SiloHardReferenceCount : Uint8B
+0x5e8 RundownWorkItem  : _WORK_QUEUE_ITEM
```

- *Compteur de processus (ActiveProcesses)* Spécifie le nombre total de processus actuellement associés au job. Voir fonction QueryInformationJobObject (JobObjectInfoClass = JobObjectBasicAccountingInformation) ; structure JOBOBJECT_BASIC_ACCOUNTING_INFORMATION.

- *Évenement (Event)* Objet sur lequel s'appuie le job en vue d'offrir une sémantique de synchronisation signalé/ non signalé.

- *Limite d'utilisation de la mémoire (JobMemoryLimit)* Voir structure JOBOBJECT_EXTENDED_LIMIT_INFORMATION.

- **Verrou de niveau job (JobLock)** Garanti l'accès synchronisé aux données sensibles de l'objet, notamment lorsqu'elles sont modifiées par plusieurs threads ou processus en même temps.

- *Maximum de working set (MaximumWorkingSetSize)* Spécifie la taille minimale jusqu'à laquelle l'ensemble de travail de chaque processus du job peut diminuer. Voir fonction QueryInformationJobObject (JobObjectInfoClass = JobObjectBasicLimitInformation) ; structure JOBOBJECT_BASIC_LIMIT_INFORMATION.

- *Minimum de working set (MinimumWorkingSetSize)* Spécifie la taille maximale jusqu'à laquelle l'ensemble de travail de chaque processus du job peut grandir. Voir fonction QueryInformationJobObject (JobObjectInfoClass = JobObjectBasicLimitInformation) ; structure JOBOBJECT_BASIC_LIMIT_INFORMATION.

- *PerProcessUserTimeLimit* Spécifie la limitation de la durée d'exécution en mode utilisateur par processus. Le système vérifie périodiquement chaque processus associé au job afin de déterminer si celui-ci a accumulé plus de durée en mode utilisateur que ne le permet la limitation. Si tel est le cas, le processus est interrompu.

- *Limite d'utilisation de la mémoire par processus (ProcessMemoryLimit)* Spécifie le montant maximal de mémoire que chaque processus peut utiliser.

- *ID de session (SessionId)* ID de la session à laquelle appartient le job.

- *Nombre de défauts de page (TotalPageFaultCount)* Nombre total de défauts de page que les processus au sein du job ont accumulés.

- *Compteur de processus terminés (TotalTerminatedProcesses)* Nombre total de processus qui se sont terminés du fait des restrictions que leur imposait l'objet job. Voir fonction QueryInformationJobObject (JobObjectInfoClass = JobObjectBasicAccountingInformation) ; service NtQueryInformationJobObject (JobInformationClass = JobObjectBasicAccountingInformation) ; structure JOBOBJECT_BASIC_ACCOUNTING_INFORMATION.

- *Temps noyau total (TotalKernelTime)* Spécifie la durée totale du temps d'exécution en mode noyau pour tous les processus associés ou l'ayant été au job.

- *Temps utilisateur total (TotalUserTime)* Indique le temps accumulé passé en mode utilisateur par les processus que le job renferme.

- *UIRestrictionsClass* Voir structure JOBOBJECT_BASIC_UI_RESTRICTIONS ; routine PsSetJobUIRestrictionsClass.

Tableau 8.48. Autorisations spécifiques aux jobs

Constante	Valeur	Description
JOB_OBJECT_ASSIGN_PROCESS	0x0001	Requis pour ajouter un processus à un job.- Controlé par NtAssignProcessToJobObject.
JOB_OBJECT_SET_ATTRIBUTES	0x0002	Requis pour modifier les différentes restrictions qu'impose un objet job.- Controlé par NtSetInformationJobObject.
JOB_OBJECT_QUERY	0x0004	Requis pour demander des informations relatives à un job. Controlé dans NtQueryInformationJobObject et NtIsProcessInJob.
JOB_OBJECT_TERMINATE	0x0008	Requis pour mettre fin à un job et, partant, aux processus qu'il renferme. Controlé par NtTerminateJobObject.
JOB_OBJECT_SET_SECURITY_ATTRIBUTES	0x0010	Obsolète.

Tableau 8.49. JOBOBJECTINFOCLASS

Nom	Valeur	Type de données associé
JobObjectBasicAccountingInformation	01	JOBOBJECT_BASIC_ACCOUNTING_INFORMATION
JobObjectBasicLimitInformation	02	JOBOBJECT_BASIC_LIMIT_INFORMATION
JobObjectBasicProcessIdList	03	JOBOBJECT_BASIC_PROCESS_ID_LIST

Nom	Valeur	Type de données associé
JobObjectBasicUIRestrictions	04	JOBOBJECT_BASIC_UI_RESTRICTIONS
JobObjectSecurityLimitInformation	05	JOBOBJECT_SECURITY_LIMIT_INFORMATION
JobObjectEndOfJobTimeInformation	06	JOBOBJECT_END_OF_JOB_TIME_INFORMATION
JobObjectAssociateCompletionPortInformation	07	JOBOBJECT_ASSOCIATE_COMPLETION_PORT
JobObjectBasicAndIoAccountingInformation	08	JOBOBJECT_BASIC_AND_IO_ACCOUNTING_INFORMATION
JobObjectExtendedLimitInformation	09	JOBOBJECT_EXTENDED_LIMIT_INFORMATION
JobObjectJobSetInformation	10	JOBOBJECT_JOBSET_INFORMATION
JobObjectGroupInformation	11	USHORT
JobObjectNotificationLimitInformation	12	JOBOBJECT_NOTIFICATION_LIMIT_INFORMATION
JobObjectLimitViolationInformation	13	JOBOBJECT_LIMIT_VIOLATION_INFORMATION
JobObjectGroupInformationEx	14	GROUP_AFFINITY (ARRAY)
JobObjectCpuRateControlInformation	15	JOBOBJECT_CPU_RATE_CONTROL_INFORMATION
JobObjectCompletionFilter	16	
JobObjectCompletionCounter	17	
JobObjectFreezeInformation	18	JOBOBJECT_FREEZE_INFORMATION
JobObjectExtendedAccountingInformation	19	JOBOBJECT_EXTENDED_ACCOUNTING_INFORMATION
JobObjectWakeInformation	20	JOBOBJECT_WAKE_INFORMATION
JobObjectBackgroundInformation	21	
JobObjectSchedulingRankBiasInformation	22	
JobObjectTimerVirtualizationInformation	23	
JobObjectCycleTimeNotification	24	
JobObjectClearEvent	25	
JobObjectInterferenceInformation	26	JOBOBJECT_INTERFERENCE_INFORMATION
JobObjectClearPeakJobMemoryUsed	27	
JobObjectMemoryUsageInformation	28	JOBOBJECT_MEMORY_USAGE_INFORMATION
JobObjectSharedCommit	29	
JobObjectContainerId	30	
JobObjectIoRateControlInformation	31	
JobObjectNetRateControlInformation	32	JOBOBJECT_NET_RATE_CONTROL_INFORMATION
JobObjectNotificationLimitInformation2	33	JOBOBJECT_NOTIFICATION_LIMIT_INFORMATION_2
JobObjectLimitViolationInformation2	34	JOBOBJECT_LIMIT_VIOLATION_INFORMATION_2
JobObjectCreateSilo	35	
JobObjectSiloBasicInformation	36	SILOOBJECT_BASIC_INFORMATION
JobObjectSiloRootDirectory	37	SILOOBJECT_ROOT_DIRECTORY
JobObjectServerSiloBasicInformation	38	SERVERSILO_BASIC_INFORMATION
JobObjectServerSiloUserSharedData	39	SILO_USER_SHARED_DATA
JobObjectServerSiloInitialize	40	
JobObjectServerSiloRunningState	41	
JobObjectIoAttribution	42	
JobObjectMemoryPartitionInformation	43	
JobObjectContainerTelemetryId	44	
JobObjectSiloSystemRoot	45	
JobObjectEnergyTrackingState	46	JOBOBJECT_ENERGY_TRACKING_STATE
JobObjectThreadImpersonationInformation	47	

JOBOBJECT_BASIC_ACCOUNTING_INFORMATION.

```
typedef struct _JOBOBJECT_BASIC_ACCOUNTING_INFORMATION {
  LARGE_INTEGER TotalUserTime;
```

```
  LARGE_INTEGER TotalKernelTime;
  LARGE_INTEGER ThisPeriodTotalUserTime;
  LARGE_INTEGER ThisPeriodTotalKernelTime;
  ULONG TotalPageFaultCount;
  ULONG TotalProcesses;
  ULONG ActiveProcesses;
  ULONG TotalTerminatedProcesses;
} JOBOBJECT_BASIC_ACCOUNTING_INFORMATION, PJOBOBJECT_BASIC_ACCOUNTING_INFORMATION;
```

JOBOBJECT_BASIC_AND_IO_ACCOUNTING_INFORMATION .

```
typedef struct _JOBOBJECT_BASIC_AND_IO_ACCOUNTING_INFORMATION {
  JOBOBJECT_BASIC_ACCOUNTING_INFORMATION BasicInfo;
  IO_COUNTERS IoInfo;
} JOBOBJECT_BASIC_AND_IO_ACCOUNTING_INFORMATION, *PJOBOBJECT_BASIC_AND_IO_ACCOUNTING_INFORMATION;
```

JOBOBJECT_BASIC_LIMIT_INFORMATION.

```
typedef struct _JOBOBJECT_BASIC_LIMIT_INFORMATION {
    LARGE_INTEGER PerProcessUserTimeLimit;
    LARGE_INTEGER PerJobUserTimeLimit;
    ULONG LimitFlags;
    SIZE_T MinimumWorkingSetSize;
    SIZE_T MaximumWorkingSetSize;
    ULONG ActiveProcessLimit;
    ULONG_PTR Affinity;
    ULONG PriorityClass;
    ULONG SchedulingClass;
} JOBOBJECT_BASIC_LIMIT_INFORMATION, *PJOBOBJECT_BASIC_LIMIT_INFORMATION;
```

JOBOBJECT_EXTENDED_LIMIT_INFORMATION.

```
typedef struct _JOBOBJECT_EXTENDED_LIMIT_INFORMATION {
    JOBOBJECT_BASIC_LIMIT_INFORMATION BasicLimitInformation;
    IO_COUNTERS IoInfo;
    SIZE_T ProcessMemoryLimit;
    SIZE_T JobMemoryLimit;
    SIZE_T PeakProcessMemoryUsed;
    SIZE_T PeakJobMemoryUsed;
} JOBOBJECT_EXTENDED_LIMIT_INFORMATION, *PJOBOBJECT_EXTENDED_LIMIT_INFORMATION;
```

JOBOBJECT_BASIC_UI_RESTRICTIONS.

```
typedef struct _JOBOBJECT_BASIC_UI_RESTRICTIONS {
    ULONG UIRestrictionsClass;
} JOBOBJECT_BASIC_UI_RESTRICTIONS, *PJOBOBJECT_BASIC_UI_RESTRICTIONS;
```

JOBOBJECT_BASIC_PROCESS_ID_LIST.

```
typedef struct _JOBOBJECT_BASIC_PROCESS_ID_LIST {
  DWORD     NumberOfAssignedProcesses;
  DWORD     NumberOfProcessIdsInList;
  ULONG_PTR ProcessIdList[1];
} JOBOBJECT_BASIC_PROCESS_ID_LIST, *PJOBOBJECT_BASIC_PROCESS_ID_LIST;
```

Tableau 8.50. Codes d'état concernant les jobs

Constante / Valeur	Fonction
STATUS_PROCESS_NOT_IN_JOB 0x00000123	NtIsProcessInJob
STATUS_PROCESS_IN_JOB 0x00000124	NtIsProcessInJob

Exécuter un processus sous un compte suppléant donne implicitement lieu à un job sans nom. Pour vérifier que le système se comporte bel et bien ainsi, procédez aux opérations suivantes.

1. Depuis l'invite de commandes, utilisez la commande runas afin donner lieu à un processus exécutant une autre invite de commandes (Cmd.exe).

2. Depuis l'invite de commandes de commandes nouvellement affichée, exécutez le bloc-notes (Notepad.exe).

3. Exécutez maintenant le débogueur noyau. Une fois cela fait, repérez le processus qui exécute la commande Cmd.exe, trouvez l'adresse de l'adresse job et enfin, affichez l'objet proprement dit via la commande !job. Ce qui montre un extrait du résultat de ces commandes.

```
lkd> !process 0 0 cmd.exe
PROCESS ffffd387fb595580
    SessionId: 1  Cid: 0d80    Peb: 4ebc6b6000  ParentCid: 0d90
    DirBase: 75b80002  ObjectTable: ffffe404dcbb7040  HandleCount:  48.
    Image: cmd.exe

PROCESS ffffd387fab10080
    SessionId: 1  Cid: 0540    Peb: 88e790a000  ParentCid: 0d98
    DirBase: 51f00002  ObjectTable: ffffe404df825940  HandleCount:  50.
    Image: cmd.exe

lkd> !process 0540
Searching for Process with Cid == 540
PROCESS ffffd387fab10080
    SessionId: 1  Cid: 0540    Peb: 88e790a000  ParentCid: 0d98
    DirBase: 51f00002  ObjectTable: ffffe404df825940  HandleCount:  50.
    Image: cmd.exe
    VadRoot ffffd387f919aa60 Vads 29 Clone 0 Private 201. Modified 0. Locked 8.
    DeviceMap ffffe404db247b80
    Token                             ffffe404e00b8aa0
    ElapsedTime                       00:01:32.836
    UserTime                          00:00:00.000
    KernelTime                        00:00:00.000
    QuotaPoolUsage[PagedPool]         30008
    QuotaPoolUsage[NonPagedPool]      4528
    Working Set Sizes (now,min,max)  (826, 50, 345) (3304KB, 200KB, 1380KB)
    PeakWorkingSetSize                1006
    VirtualSize                       2101297 Mb
    PeakVirtualSize                   2101306 Mb
    PageFaultCount                    1255
    MemoryPriority                    BACKGROUND
    BasePriority                      8
    CommitCharge                      550
    Job                               ffffd387f87fd830
    .
    .
    .

lkd> !job ffffd387f87fd830
Job at ffffd387f87fd830
  Basic Accounting Information
    TotalUserTime:             0x0
    TotalKernelTime:           0x0
    TotalCycleTime:            0x0
    ThisPeriodTotalUserTime:   0x0
    ThisPeriodTotalKernelTime: 0x0
    TotalPageFaultCount:       0x0
    TotalProcesses:            0x3
    ActiveProcesses:           0x3
    FreezeCount:               0
    BackgroundCount:           0
    TotalTerminatedProcesses:  0x0
    PeakJobMemoryUsed:         0xd23
    PeakProcessMemoryUsed:     0x612
  Job Flags
```

```
Limit Information (LimitFlags: 0x0)
Limit Information (EffectiveLimitFlags: 0x0)
```

Fonctions	Services	Routines
AssignProcessToJobObject	NtAssignProcessToJobObject	
CreateJobObject	NtCreateJobObject	ObCreateObject
IsProcessInJob	NtIsProcessInJob	
OpenJobObject		
QueryInformationJobObject	NtQueryInformationJobObject	ObReferenceObjectByHandle
SetInformationJobObject	NtSetInformationJobObject	
TerminateJobObject	NtTerminateJobObject	

Threads auxiliaires utilisateur

Services

NtCreateWorkerFactory
NtQueryInformationWorkerFactory
NtReleaseWorkerFactoryWorker
NtSetInformationWorkerFactory
NtShutdownWorkerFactory
NtWaitForWorkViaWorkerFactory
NtWorkerFactoryWorkerReady

Structures

WORKER_FACTORY_BASIC_INFORMATION

Tableau 8.51. API de pool de threads

Fonctions	Services (TpXxx)
TrySubmitThreadPoolCallback	TpSimpleTryPost
CreateThreadpoolWork	TpAllocWork
CloseThreadpoolWork	TpReleaseWork
SubmitThreadpoolWork	TpPostWork
CreateThreadpool	TpAllocPool
CloseThreadpool	TpReleasePool
SetThreadpoolThreadMaximum	TpSetPoolMaxThreads
SetThreadpoolThreadMinimum	TpSetPoolMinThreads
SetThreadpoolStackInformation	TpSetPoolStackInformation
QueryThreadpoolStackInformation	TpQueryPoolStackInformation
CreateThreadpoolCleanupGroup	TpAllocCleanupGroup
CloseThreadpoolCleanupGroup	TpReleaseCleanupGroup
SetThreadpoolCallbackCleanupGroup	TpSetCallbackCleanupGroup
CloseThreadpoolCleanupGroupMembers	TpReleaseCleanupGroupMembers
CreateThreadpoolTimer	TpAllocTimer
SetThreadpoolTimer	TpSetTimer
SetThreadpoolTimerEx	TpSetTimerEx
IsThreadpoolTimerSet	TpIsTimerSet
CreateThreadpoolWait	TpAllocWait
CreateThreadpoolIo	TpAllocIoCompletion

Tableau 8.52. WORKERFACTORYINFOCLASS

Nom	Valeur	Type de données associé
WorkerFactoryTimeout	00	LARGE_INTEGER

Nom	Valeur	Type de données associé
WorkerFactoryRetryTimeout	01	LARGE_INTEGER
WorkerFactoryIdleTimeout	02	LARGE_INTEGER
WorkerFactoryBindingCount	03	ULONG
WorkerFactoryThreadMinimum	04	ULONG
WorkerFactoryThreadMaximum	05	ULONG
WorkerFactoryPaused	06	ULONG
WorkerFactoryBasicInformation	07	WORKER_FACTORY_BASIC_INFORMATION
WorkerFactoryAdjustThreadGoal	08	
WorkerFactoryCallbackType	09	
WorkerFactoryStackInformation	10	
WorkerFactoryThreadBasePriority	11	ULONG
WorkerFactoryTimeoutWaiters	12	ULONG
WorkerFactoryFlags	13	ULONG
WorkerFactoryThreadSoftMaximum	14	ULONG
WorkerFactoryThreadCpuSets	15	
MaxWorkerFactoryInfoClass	16	

Paramètres de langue et d'Internationalisation

Fontions	Services	Routines
GetSystemDefaultLangID	NtQueryDefaultLocale	
GetSystemDefaultLCID	NtQueryDefaultLocale	
GetSystemPreferredUILanguages	RtlGetSystemPreferredUILanguages	
GetSystemDefaultUILanguage	NtQueryInstallUILanguage	
GetThreadLocale	NtQueryInformationThread	
GetUserDefaultLangID	NtQueryDefaultLocale	
GetUserDefaultLCID	NtQueryDefaultLocale	MmGetSessionLocaleId
GetUserDefaultUILanguage	NtQueryDefaultUILanguage	
GetUserGeoID	NtQueryValueKey	
IsValidLocale		
IsValidLocaleName	RtlIsValidLocaleName	
SetThreadLocale	NtSetInformationThread	
SetThreadUILanguage	RtlSetThreadPreferredUILanguages	
SetUserGeoID	NtSetValueKey	

Attributs	Structures	Autres références
CurrentLocale	TEB	GetThreadLocale, SetThreadLocale
LocaleId	MM_SESSION_SPACE	MmGetSessionLocaleId

Conclusion

Nous avons dans ce chapitre étudié la structure interne des processus, des threads et des *jobs*, vu comment tous naissait, mourrait, et ce qu'il se produisant dans le système en ces circonstances. Nous avons aussi examiné comment Windows décidait quels threads exécuter, sur quel processeur et pendant combien de temps.

Chapitre 9. IPC
Tubes

Les processus qui communiquent par l'intermédiaire d'un tube nommé doivent être liés au sein de la même généalogie. En effet, un tube anonyme, ou du moins les références (handles) dont il fait l'objet, fait partie des caractéristiques dont hérite les processus de leur créateur respectif. Les processus ayant en vue une coopération semblable, mais qui ne descendent pas du même père, ont pour seule solution de donner un nom au tube.

Le tableau qui suit énumère quelles fonctions de l'API Windows permettent la manipulation du tubes.

Tableau 9.1. API Windows concernant les tubes

Fonction	Description
CreatePipe	Crée un tube anonyme
CreateNamedPipe	Crée un tube anonyme
ConnectNamedPipe	-

Le mode de lecture auquel répond un tube détermine comment les données sont lues. Il existe deux modes de lecture : le mode octet et le mode message, l'un ou l'autre choisi par l'application serveur au moment de la mise en place d'un canal nommé (CreateNamedPipe).

- En mode octet (PIPE_READMODE_BYTE), les données sont lues à la manière d'un flux continu d'octets. Une opération d'E/S lecture ne peut dans cette optique être couronnée que si tous tous les octets disponibles au travers du tube sont sont lus, ou éventuellement si le nombre d'octets demandé est lu.

- Dans le mode message, les données sont extraites tels des blocs unitaires de taille fixe. Une opération de lecture prend fin avec succès seulement si l'ensemble du message est lu. Si le nombre d'octets à lire est inférieur à la taille du message en transit, la fonction récupère autant du message que possible et retourne un code d'erreur indiquant que la communication n'est pas complète (ERROR_MORE_DATA). Le reste du message peut être lu en réitérant une opération de lecture.

Il est possible de modifier à chaud le mode de lecture conditionnant un canal de transmission, ce au moyen de SetNamedPipeHandleState. Quand elle est appelée, la fonction consulte le paramètre lpMode exigé en paramètre, et s'il est présent, modifie en conséquence les attributs de l'objet fichier (NtSetInformationFile).

Atomes globaux

Tableau 9.2. Opération concernant les atomes globaux

Opération	Fonction	Service
Ajouter un atome à la table des atomes globaux	GlobalAddAtom	NtAddAtom
Rechercher un atome parmi la table des atomes globaux	GlobalFindAtom	NtFindAtom
Supprimer un atome de la table la table des atomes globaux	GlobalDeleteAtom	NtDeleteAtom
	GlobalGetAtomName	NtQueryInformationAtom

Tableau 9.3. ATOM_INFORMATION_CLASS

ID	Classe	Type de données associé
0x0000	AtomBasicInformation	ATOM_BASIC_INFORMATION
0x0001	AtomTableInformation	ATOM_TABLE_INFORMATION

ATOM_BASIC_INFORMATION.

```
typedef struct _ATOM_BASIC_INFORMATION {
    USHORT ReferenceCount;
```

```
    USHORT Flags;
    USHORT NameLength;
    WCHAR Name[1];
} ATOM_BASIC_INFORMATION, *PATOM_BASIC_INFORMATION;
```

Boites aux lettres (*Mailslots*)

Les boites aux lettes (*mailslots*) sont un mécanisme sans connexion de transmission de messages, généralement utilisé dans des environnement connectés pour, par exemple, trouver des composants réseau ou envoyer un message à une machine destinataire. Le fonctionnement de ces boites aux lettres s'inspire de celui de leurs homologues en matière de télécommunications (messagerie électronique) : chaque conteneur est un espace de mémoire réservé à un abonné, dans lequel sont conservés les messages qui lui sont destinés.

A l'instar d'un canal de communication, une boite aux lettres est implantée sous forme d'un fichier mémoire auquel que les applications peuvent accéder via les fonctions standards d'E/S fichier (CreateFile, WriteFile, ReadFile, CloseHandle). Les fichiers correspondants à des boites aux lettres sont temporaires, et disparaissent dès que le dernier descripteur ouvert sur une instance d'un tel type d'objet est refermé.

Chaque boite aux lettres est identifié par un descripteur d'instance, lequel résulte de la création de l'objet (CreateMailslot), et permet à un processus de lire les messages reçus. Seul un processus qui a créé ou hérité d'une boite aux lettres peut effectuer ce type d'opération. Une évidente mesure de sécurité supplémentaire est le fait que le système n'autorise pas la création distante de mailslots.

Tout processus qui connait le nom d'une boite aux lettes peut y déposer un message, et à ce titre en devenir client.

L'implémentation Windows des boites aux lettres diffuse les messages à l'aide de datagrammes. Orientés non connectés, ce type de paquet de données n'est pas fiable, au sens où le message envoyé peut être perdu avant que le destinataire le reçoive, et sans que l'émetteur ne sache rien de cette défaillance (pas d'accusé de réception).

Fonction	Service	Routine
CreateMailslot	NtCreateMailslotFile	IoCreateFile
GetMailslotInfo	NtQueryInformationFile	
SetMailslotInfo	NtSetInformationFile	

Structure de données ! Autres références

FILE_MAILSLOT_SET_INFORMATION

NtSetInformationFile

FILE_MAILSLOT_QUERY_INFORMATION

NtQueryInformationFile

Chapitre 10. Gestion de la mémoire
Principes de base de la mémoire

L'unité de mesure de la mémoire est l'octet. Lorsqu'une grandeur est exprimée en multiples de l'octet, les symboles Ko, Mo et Go représentent respectivement le kilooctet, le mégaoctet et le gigaoctet. Dans ce cas, et contrairement à la norme, on considère généralement qu'un kilooctet vaut 2^10 (1 024) octets (et non 1 000, ainsi que le préfixe kilo - multiplication par mille - le laisse entendre de prime abord), un mégaoctet vaut 2^20 octets, et ainsi de suite. A titre informatif, le tableau qui vient répertorie les différents multiples de l'octet.

Tableau 10.1. Multiples de l'octets

Unité de mesure	Capacité (octets)
Kilooctet	1024
Mégaoctet	1024^2
Gigaoctet	1024^3
Téraoctet	1024^4
Pétaoctet	1024^5
Exaoctet	1024^6
Zettaoctet	1024^7
Yottaoctet	1024^8

ROM

A l'origine, les mémoires mortes étaient des mémoires électroniques persistantes en lecture seule, dès lors uniquement consultables. Parallèlement à l'évolution des technologies, la définition du terme a pris de l'ampleur pour désormais inclure toute les formes de mémoire dont le contenu est défini initialement à la fabrication, mais qui peut peut aussi (plus ou moins facilement) faire l'objet de modifications. La plupart des systèmes actuels utilisent un type de ROM appelé EEPROM, une forme de mémoire flash.

La mémoire morte contient les instructions permettant le démarrage et les premiers contrôles du fonctionnement de l'ordinateur.

Remarquez que si l'on a généralement tendance a opposer les deux formes de mémoire (ce que nous avons nous même fait dans le plan de ce livre), la raison de l'antagonisme entre mémoire vive et mémoire morte provient essentiellement du caractère volatile de l'une, et non volatile (rémanente) de l'autre. Dans la perspective du système d'exploitation, en revanche, RAM et ROM sont simplement deux types différents de mémoire, qui plus est complémentaires, puisqu'une partie de l'espace d'adressage noyau (donc de la RAM) est allouée à plusieurs puces de mémoire ROM. En ce sens, la ROM est un sous ensemble de la RAM.

Gestionnaire de mémoire

Le composant de l'exécutif en charge des procédés sous-jacents à la gestion de de la mémoire, y compris l'espace d'adressage virtuel, l'allocation de la mémoire physique et la pagination, est le gestionnaire de mémoire.

La fonction première du gestionnaire de mémoire est l'implémentation de la mémoire virtuelle, à savoir un système de gestion mémoire qui offre à chaque processus un espace d'adressage privatif susceptible d'être plus grand que la mémoire physique disponible. La mise en oeuvre d'un tel projet étant fortement dépendante de dispositifs physiques, en premier lieu l'unité de gestion mémoire, partie intégrante de tous les microprocesseurs récents, le gestionnaire de mémoire suppose que l'architecture matérielle supporte un mécanisme de pagination et la translation virtuel vers physique.

Initiateur et administrateur du lien entre pages virtuelles et pages physiques, le gestionnaire de mémoire participe de cette manière au respect des règles de protection qui limitent l'accessibilité d'un thread donné aux pages appartenant à son propre espace d'adressage (sauf cas très particulier).

De par la proximité des schémas sous-jacents à la mémoire à ceux relatifs au cache système, le gestionnaire de mémoire virtuelle fournit une partie de la prise en charge sous-jacente au gestionnaire de cache.

Fonctionnalités clé du gestionnaire de mémoire

Le gestionnaire de mémoire a plusieurs responsabilités majeures :

- Configurer à bon escient les dispositifs matériels impliqués lors de la traduction des adresses virtuelles employées par les threads d'un processus en leurs correspondances au niveau physique.
- Mapper l'espace d'adressage virtuel d'un processus vers la mémoire physique, de sorte que quand un thread exécuté dans le contexte de ce processus lit ou écrit dans l'espace d'adresse virtuel, il y ait chargement de la page physique appropriée.
- Paginer une partie du contenu de la mémoire vers le disque quand elle devient surengagée, c'est à dire quand les threads exécutés essaient d'utiliser sollicitent plus de mémoire physique que l'espace réellement disponible, puis recharger le contenu en mémoire physique quand la nécessité s'en fait sentir. (La partie de l'espace d'adresse virtuel d'un processus qui réside en mémoire physique est dite *working set* ou *plage de travail*. Les working sets sont détaillées plus loin.)
- Assurer une protection renforcée contre les accès intempestifs ou non autorisés à l'espace d'adressage virtuel d'un processus individuel ou du système.

Le gestionnaire de mémoire se compose des éléments que voici :

- Un ensemble de services système chargés d'allouer, libérer et gérer la mémoire virtuelle ; la plupart d'entre eux étant exposés via l'API Windows.
- Un gestionnaire d'interception des traductions non valides chargé de résoudre les exceptions de gestion mémoire détectées par le matériel et de lire en mémoire physique les pages virtuelles pour le compte d'un processus.
- Le zero page thread (thread de page à zéro) remplit de zéros les pages de la liste des pages libres. Une telle précaution a pour but d'empêcher un processus utilisateur de récupérer des données ou de code ayant appartenu à l'espace d'adressage d'un précédent processus.
- Le process/stack swapper (échangeur processus/pile) effectue la permutation (inswapping/outswapping) des processus et des piles mode noyau du thread. Le code noyau d'ordonnancement réveille ce thread quand il est nécessaire de procéder à une opération de permutation.
- Le modified page writer (écrivain de pages modifiées) écrit dans les fichiers de pagination idoines les pages appartenant à la liste des pages modifiées.
- Le mapped page writer (écrivain de pages mappées) écrit sur le disque les pages modifiées se rapportant à des fichiers mappés.
- Le dereference segment thread (thread de segment de déréférencement) contrôle la taille du cache et du fichier de pagination.
- Le working set manager (gestionnaire de working set) pilote les stratégies générales de gestion mémoire, tel l'expansion et l'élagage de working set, le vieillissement des pages, etc.

Chacun de ces composants fait plus loin l'objet d'une étude détaillée.

Synchronisation interne

À l'instar des autres composants de l'exécutif Windows, le gestionnaire de mémoire est pleinement réentrant et autorise l'exécution simultanée sur les systèmes multi processeur. Il s'appuie pour ce faire sur un certain nombre de mécanismes de synchronisation, y compris spinlocks et pushlocks.

Les ressources globales envers lesquelles le gestionnaire de mémoire doit synchroniser les accès incluent la base de données PFN, les objets section, les fichiers de pagination et le working set système.

Les structures de données de gestion mémoire de niveau processus qui exigent de la synchronisation incluent l'espace d'adressage et le working set. Le tableau suivant répertorie les données sur lesquelles s'appuie le gestionnaire de mémoire quand il s'agit de synchronisation.

Tableau 10.2. Eléments auxiliaires sous-jacents à la synchronisation

Elément	Type	Lieu	Commentaires
PageFileCreationLock	EX_PUSH_LOCK	MI_PARTITION_MODWRITES	
AddressCreationLock	EX_PUSH_LOCK	EPROCESS	Voir routine MmInitializeProcessAddressSpace.
LoadLock	KMUTANT	MI_SYSTEM_IMAGE_STATE	Anciennement MmSystemLoadLock.- Synchronise le chargement et le déchargement des images système. Voir routine MmGetSystemRoutineAddress.
MmPfnLock	KSPIN_LOCK_QUEUE	KPRCB	
ExPageLockHandle		nt (variable noyau)	
ProcessLock	EX_PUSH_LOCK	EPROCESS	
WorkingSetMutex	EX_PUSH_LOCK	MMSUPPORT	
WorkingSetLock		EJOB	

Configuration du gestionnaire de mémoire

La liste qui suit recense les clés de registre corrélées directement à la gestion de la mémoire, stockés sous HKLM \System\CurrentControlSet\Control\Session Manager\Memory Management.

- *ClearPageFileAtShutdown* Contrôle la suppression automatique du fichier de pagination lors de l'arrêt ou du redémarrage du système.
- *SystemCacheLimit* Contrôle la taille virtuelle du cache. (Systèmes 32 bits uniquement.)
- *DisablePagingExecutive* Empêche que les pages de mémoire virtuelle apparentées à l'exécutif Windows soient remisées sur le disque.
- *HeapDeCommitFreeBlockThreshold* Détermine la façon dont le gestionnaire de tas se comporte lors d'opérations de libération de la mémoire.

Paramètres de démarrage

Afin de procurer une aide concernant les problèmes liés à la mémoire, Windows intègre parmi la configuration de démarrage plusieurs paramètres ayant une influence directe sur la façon dont les ressources sont effectivement exploitées.

L'utilitaire BCDEdit offre en vue d'interagir sur la quantité de mémoire reconnue par Windows deux méthodes qui, bien qu'elles aient une finalité identique, ne s'emploient pas de la même manière.

- Pour priver le système d'exploitation d'une certaine quantité de mémoire, employez l'option removememory.
- Pour définir une adresse physique à partir de laquelle la mémoire n'est plus prise en compte, employez l'option truncatememory.

Par exemple, pour limiter la mémoire d'un ordinateur doté de 2 Go de mémoire physique à maximum de 512 Mo de mémoire disponible, spécifiez auprès de l'option removememory une valeur de 1536 Mo (2048 - 1536 = 512 Mo). Pour parvenir à résultat similaire au moyen de l'option truncatememory, spécifiez la valeur 0x20000000 (ou 536870912, soit toujours 512 Mo).

Visualisation de l'utilisation de la mémoire

La commande !vm du débogueur noyau affiche des informations basiques concernant l'utilisation de la mémoire.

```
lkd> !vm 1
Page File: \??\C:\pagefile.sys
  Current:    655360 Kb  Free Space:    536064 Kb
  Minimum:    655360 Kb  Maximum:      9437184 Kb
Page File: \??\C:\swapfile.sys
  Current:     16384 Kb  Free Space:     16376 Kb
  Minimum:     16384 Kb  Maximum:      4683772 Kb
No Name for Paging File
  Current:  12559700 Kb  Free Space:  12021348 Kb
  Minimum:  12559700 Kb  Maximum:     12559700 Kb

Physical Memory:           780629 (    3122516 Kb)
Available Pages:           276499 (    1105996 Kb)
ResAvail Pages:            635621 (    2542484 Kb)
Locked IO Pages:                0 (          0 Kb)
Free System PTEs:      4294970779 (17179883116 Kb)
Modified Pages:              6646 (      26584 Kb)
Modified PF Pages:           6631 (      26524 Kb)
Modified No Write Pages:       29 (        116 Kb)
NonPagedPool Usage:           278 (       1112 Kb)
NonPagedPoolNx Usage:       31596 (     126384 Kb)
NonPagedPool Max:      4294967296 (17179869184 Kb)
PagedPool Usage:            69930 (     279720 Kb)
PagedPool Maximum:     4294967296 (17179869184 Kb)
Processor Commit:             508 (       2032 Kb)
Session Commit:              9589 (      38356 Kb)
Shared Commit:              66681 (     266724 Kb)
Special Pool:                   0 (          0 Kb)
Kernel Stacks:              10007 (      40028 Kb)
Pages For MDLs:              1389 (       5556 Kb)
Pages For AWE:                  0 (          0 Kb)
NonPagedPool Commit:        33949 (     135796 Kb)
PagedPool Commit:           69930 (     279720 Kb)
Driver Commit:              11736 (      46944 Kb)
Boot Commit:                 3267 (      13068 Kb)
PFN Array Commit:            9701 (      38804 Kb)
System PageTables:           1323 (       5292 Kb)
ProcessLockedFilePages:        10 (         40 Kb)
Pagefile Hash Pages:           60 (        240 Kb)
Sum System Commit:         218150 (     872600 Kb)
Total Private:             364444 (    1457776 Kb)
Misc/Transient Commit:       4118 (      16472 Kb)
Committed pages:           586712 (    2346848 Kb)
Commit limit:              944469 (    3777876 Kb)
```

- *Physical memory* Quantité de mémoire physique utilisable (RAM) installée sur l'ordinateur.
- *ResAvail Pages* Voir attribut de structure MI_VISIBLE_PARTITION ResidentAvailablePages.

Gestion du tas

#	Fonctions	Services
1	HeapCreate	RtlCreateHeap
2	HeapDestroy	RtlDestroyHeap
3	HeapAlloc	RtlAllocateHeap
4	HeapFree	RtlFreeHeap
5	HeapRealloc	RtlReAllocateHeap
6	HeapLock	RtlLockHeap

#	Fonctions	Services
7	HeapUnlock	RtlUnlockHeap
8	HeapWalk	RtlWalkHeap
9	HeapCompact	RtlCompactHeap
10	GetProcessHeap	RtlGetProcessHeap
11	GetProcessHeaps	RtlGetProcessHeaps
12	HeapValidate	RtlValidateHeap
13	HeapQueryInformation	RtlQueryHeapInformation
14	HeapSetInformation	RtlSetHeapInformation

1. Créer un tas pour le compte du processus appelant.
2. Supprimer un tas.
3. Allouer un bloc depuis un tas.
4. Démanteler un bloc précédemment alloué à partir d'un tas.
5. Modifier la taille d'une allocation existante (augmenter ou rétrécir un bloc).
6. Contrôler l'exclusion mutuelle requises pour les opérations concernant un tas (s'approprier la primitive de verrouillage associée).
7. Contrôler l'exclusion mutuelle requises pour les opérations concernant un tas (céder la primitive de verrouillage associée).
8. Énumérer les entrées et régions d'un tas.
9. Réorganiser l'espace non utilisé au sein d'un tas, cela dans l'optique d'en réduire la fragmentation.
10. Obtenir un identifiant sur le tas par défaut du processus appelant.
11. Obtenir un tableau d'identifiants vers tous les tas alloués dans le processus appelant.
12. Repérer d'éventuelles anomalies signes d'une corruption du tas.
13. Recueillir diverses informations sur un tas : caractéristiques, performances, statistiques d'utilisation, et plus encore.
14. Actualiser certaines informations ou options pour un tas, influençant son comportement de manière à mieux répondre aux besoins de l'application en termes de performance, de sécurité, ou autres.

Sur la notion de tas

Ce que l'on appelle communément *tas* (heap) désigne une région de la mémoire allouée à l'exécution dans la mémoire virtuelle d'une application. Un tel dispositif convient particulièrement bien quand : (1) le nombre et la taille des objets utilisés par une application ne sont pas connus à l'avance, ou (2) quand un objet est trop volumineux pour prendre place sur la pile - ou toute autre forme de mémoire aux dimensions réduites, par exemple les registres du processeur.

Si une grande majorité des systèmes d'exploitation offrent nativement un sous-ensemble de routines consacrées au tas, il n'est pas rare de voir tel ou tel langage de programmation concocter en la matière ses propres recettes, et posséder de ce fait sa propre implémentation d'un pareil modèle. La bibliothèque d'exécution C, par exemple, gère un tas au travers de différents algorithmes dédiés.

Tas et processus

Chaque processus exécuté sous Windows a au moins un tas, le tas par défaut du processus, et s'accompagne éventuellement d'un nombre variable de tas supplémentaires. Le tas par défaut est créé au démarrage du

processus et perdure pendant toute la durée de vie du processus. Les autres tas sont le résultat d'actions menées lors de l'exécution du processus et ont en l'occurrence une durée de vie largement dépendante du contexte. En dehors de cela, il n'existe pas de différence fondamentale entre les différents tas d'un processus.

Le tas par défaut peut être utilisé explicitement par un programme, ou implicitement par des routines internes. Un grand nombre de fonctions incorporées aux DLL Windows fondamentales (Kernel32, Advapi, et ainsi de suite), interagissent par exemple avec le tas par défaut du processus dans lequel elles s'exécutent.

Une application peut interroger le tas par défaut du processus au moyen de la fonction Windows GetProcessHeap, laquelle permet d'obtenir une référence utilisable lors d'appels ultérieurs aux fonctions de tas. En interne, ladite fonction se base sur la valeur de l'attribut ProcessHeap du bloc du processus en cours d'exécution.

Une application peut posséder plusieurs tas. La liste qui suit montre en deux exemples quels bénéfices peut tirer un processus de l'utilisation de plusieurs tas.

- À plus ou moins long terme, l'utilisation d'un tas unique par processus introduit le problème de la fragmentation du tas. Il peut alors être intéressant d'organiser les tas en fonction des informations vouées à y être en transit. Dans ce modèle, chaque tas héberge des objets de même taille : les nouveaux objets alloués prennent automatiquement et naturellement la place des anciens.

- Si la synchronisation de tas est activée (voir plus loin), il y a un verrou par tas qui protège toutes les structures internes du tas. Windows synchronise l'accès entre les threads en s'emparant du verrou avant chaque opération. Si un thread a son propre tas, il est possible d'empêcher cette sérialisation via le flag HEAP_NO_SERIALIZE. Notez que cette façon de procéder requiert une attention toute particulière en ce qui concerne d'éventuels problèmes d'exclusion mutuelle.

L'attribut NumberOfHeaps du bloc d'environnent d'un processus donné comptabilise le nombre de tas que ce processus a en sa possession.

Dans un certain nombre de cas, le fait d'avoir recours à plusieurs tas au sein d'un même processus peut être avantageusement remplacé par d'autres mesures visant à atteindre les mêmes résultats. Une solution envisageable à cet égard consiste, par exemple, à mettre en oeuvre des fonctions supports agissant en tant qu'interface entre les opérations de tas effectuées par les threads d'un processus et le tas par défaut. La finalité de ces traitements, si elle peut être multiple, est de favoriser des stratégies d'allocation personnalisées susceptibles d'améliorer les performances. Ainsi, quand un processus alloue fréquemment de petites quantités de mémoire de tas, il est possible de regrouper ces allocations à l'intérieur d'une même partie du tas par défaut. Une large zone de mémoire est au préalable de cette optique allouée à partir du tas par défaut, et subdivisée par la suite en plusieurs lots de de moindre envergure employés dans le contexte d'une fonction de support interne chargée de répondre aux requêtes d'allocation. Il n'y a dans cette configuration aucun tas supplémentaires de créé, celui par défaut étant le seul à être sollicité.

Gestionnaire de tas

Par contraste avec la granularité d'allocation naturelle de Microsoft Windows, actuellement 64 Ko, celle du gestionnaire de tas est relativement faible : 8 octets sur les systèmes 32 bits, 16 octets sur les systèmes 64 bits.

Sur le plan structurel, le gestionnaire de tas existe à deux endroits du système d'exploitation : d'une part dans la bibliothèque générale de support système (Ntdll.dll), d'autre part au sein de la couche supérieure du noyau (Ntoskrnl.exe). Les API de sous-systèmes, telle l'API Windows de gestion de tas, sollicitent les fonctions incorporées à Ntdll, tandis que les composants de l'exécutif et les pilotes de périphérique appellent les fonctions de Ntoskrnl. Les interfaces natives du gestionnaire de tas (préfixées par Rtl, par exemple RtlCreateHeap ou RtlDestroyHeap) ne sont accessibles qu'aux composants internes de Windows ou aux pilotes mode noyau. Les routines de gestion de tas définies dans l'API Windows principale (préfixées par Heap, par exemple HeapCreate ou HeapDestroy) forment des enveloppes, par ailleurs relativement minces, autour des fonctions natives de Ntdll. On trouve, en outre, toujours au niveau de la DLL fondamentale du sous-système Windows (Kernel32.dll),

des API hérités (préfixées par Local ou Global, par exemple LocalAlloc ou GlocalAlloc), qui ne perdurent qu'à des fins de compatibilité pour les anciennes applications.

À l'instar d'autres composants de Windows étroitement liés à l'utilisation des ressources, le gestionnaire de tas a été conçu de sorte à répondre aux attentes les plus exigeantes du point de vue de la consommation de mémoire et des performances. Il expose à cet égard un large éventail de stratégies optimales pour l'utilisation du tas, par exemple le fait de minimiser la fragmentation du tas, ou la couche de tas frontale, ou tas à faible fragmentation, qui accentue encore plus cette perspective.

En plus des algorithmes d'allocation et de libération de la mémoire, le gestionnaire de tas est aussi responsable de quelques procédés connexes, par exemple l'exclusion mutuelle, qui permet de résoudre les problèmes basiques de compétition entre les divers threads quand ils manipulent des données du tas, ou la vérification d'intégrité, qui améliore la robustesse des applications et vise à prévenir les débordements de tas.

Un défi de taille relevé avec brio par le gestionnaire de tas est de masquer les spécificités internes mises en œuvre pour la gestion des ressources système, par exemple la différence entre la mémoire réservée, libre et engagée, ou la dimension de l'ensemble de travail. Ce que laisse entrevoir une telle approche permet en définitive aux concepteurs de logiciels de rester concentrer uniquement sur des aspects d'ordre fonctionnel, détachés au maximum des détails d'implémentation spécifiques en la matière.

Structure du gestionnaire de tas

La structure générale du gestionnaire de tas s'articule autour de deux entités distinctes, mais néanmoins complémentaires : une couche frontale, qui peut prendre diverses formes, et la couche centrale.

La couche centrale du gestionnaire de tas gère les fonctionnalités de base et elle est, en grande partie, commune aux implémentations mode utilisateur et mode noyau des tas. Les fonctions implantées dans la couche centrale incluent la gestion des segments, la gestion des blocs dans les segments, l'engagement et le désengagement de la mémoire, ainsi que les stratégies d'extension de tas.

Pour les tas mode utilisateur, il peut exister une couche frontale par dessus les fonctionnalités centrales. Les environnements susceptibles d'entrer en ligne de compte au niveau de la couche frontale incluent les listes look-aside et le tas à faible fragmentation, tous deux décrits plus loin dans cette section. Il n'est possible d'utiliser qu'une seule couche frontale pour un tas à la fois.

Synchronisation de tas

Pour satisfaire intelligemment aux contraintes usuelles en matière de cohérence des données, le gestionnaire de tas prend en charge de manière naturelle les accès concurrents effectués depuis de multiples threads. Comme dans n'importe quel autre domaine où il est besoin de surveiller particulièrement ces aspects, la synchronisation de tas garantit des accès mutuellement exclusifs entre les différents threads d'un processus quand ils utilisent des fonctions de tas.

Le but premier de la synchronisation de tas est d'ériger un rempart de protection contre les principales sources de corruption du tas. Si les vulnérabilités à ce niveau peuvent se manifester sous des formes très diverses (et être d'ailleurs très difficile à repérer), elles conduisent surtout à des informations erronées parmi les structures de contrôle du tas, ce qui mène erreurs de traitement, par exemple deux threads se partageant un même bloc du tas.

En interne, la synchronisation de tas est prise en compte par l'intermédiaire de sections critiques, lesquelles font office de verrous entre les threads d'un même processus, et permettent d'apporter en toute sécurité des modifications aux données des tas ainsi protégés.

Un processus peut demander de se passer de la synchronisation du tas en spécifiant le flag HEAP_NO_SERIALIZE lors de la création du tas (HeapCreate) ou au niveau de l'allocation individuelle (HeapAlloc). Cela est notamment utile pour éviter la surcharge induite par la synchronisation, ou afin de se réapproprier la question des dispositifs mis en œuvre pour parvenir à celle-ci. Notez que désactiver la synchronisation de tas ne constitue une solution viable que dans la mesure où un processus est mono thread, ou si un seul de ses threads accède au tas, ou encore s'il protège lui-même les entrées et régions d'un tas en mettant en œuvre ses propres algorithmes.

Verrouillage du tas

Parmi le catalogue des procédés mis en oeuvre relativement au contrôle de l'exclusion mutuelle, le gestionnaire de tas permet aux processus de verrouiller tout le tas, et de la sorte empêcher d'autres threads de faire des opérations de tas. Utilisé au premier stade par le gestionnaire de tas afin de garantir l'intégrité des structures internes du tas, le verrouillage de tas se montre essentiellement utile quand des opérations de tas exigent des états cohérents entre les multiples appels aux fonctions de tas. Par exemple, l'énumération des blocs d'un tas avec la fonction Windows HeapWalk requiert le verrouillage du tas si plusieurs threads sont susceptibles de faire des opérations de tas en même temps.

Un processus réclame le verrou correspondant à un tas par le biais de la fonction Windows HeapLock. A partir du moment où l'appropriation a eu lieu, seul le thread appelant est en mesure d'allouer et de libérer de la mémoire dans le tas. Toute autre entité exécutée au sein de ce processus qui essaie d'utiliser le tas pour une raison quelconque voit son exécution bloquée tant que le thread propriétaire du verrou ne s'est pas désengagé de cette mise sous tutelle, ce qu'il fait via l'appel de fonction HeapUnlock.

Par nature, verrouiller le tas implique que la synchronisation soit active. Si elle ne l'est pas, il en résulte un comportement potentiellement inopportun des routines matérialisant ces aspects. Les fonctions HeapLock et HeapUnlock ne doivent par conséquent jamais être utilisées conjointement avec le flag HEAP_NO_SERIALIZE, leur utilisation combinée pouvant générer des résultats imprévisibles.

Le couple HeapLock/HeapUnlock s'appuie sur les fonctionnalités offertes au niveau des interfaces natives RtlLockHeap et RtlUnlockHeap du gestionnaire de tas.

Chaque verrou existant pour protéger un tas se présente sous la forme d'un objet section critique.

Fonctionnalités de débogage du tas

Parmi toutes les fonctionnalités que propose le gestionnaire de tas, quelques-unes existent pour faciliter le repérage des bogues, notamment :

- *Contrôle de queue* La fin de chaque bloc contient une signature qui est controlée à la suppression (désallocation) de la région. Dans l'éventualité où un débordement de tampon a détruit tout ou partie de la signature, le gestionnaire de tas met en évidence l'erreur par le biais d'une exception. Cette option correspond à FLG_HEAP_ENABLE_TAIL_CHECK.

- *Contrôle des blocs libres* Suit l'évolution du contenu des blocs libres par l'intermédiaire d'un motif que le gestionnaire de tas contrôle quand il doit accéder à un bloc. Dans l'éventualité où un processus a écrit dans un bloc après l'avoir libéré, le gestionnaire de tas reconnait que le motif n'est pas celui attendu et signale l'erreur. Cette option correspond à FLG_HEAP_ENABLE_FREE_CHECK.

- *Contrôle des paramètres* Assure un contrôle exhaustif des paramètres passées aux fonctions de gestion de tas. Cette option correspond à FLG_HEAP_VALIDATE_PARAMETERS.

- *Validation du tas* Vérifie l'intégralité du tas tout entier chaque fois qu'une fonction de gestion de tas est appelée. Cette option correspond à FLG_HEAP_VALIDATE_ALL.

- *Balisage du tas* Cette option correspond à FLG_HEAP_ENABLE_TAGGING.

Toutes les options suscitées ont un point commun, à savoir découvrir les corruptions susceptibles d'apparaitre à l'exécution. Les trois premières sont activées par défaut dès lors que le démarrage du processus a été pris en charge par un débogueur.

Il est possible de spécifier quelles fonctionnalités de débogage de tas ont lieu pour une image exécutable en positionnant différents flags dans l'entête de l'image via l'utilitaire Gflags, et pour une image exécutée en utilisant la commande !heap des débogueurs Windows standard.

Visualisation des tas d'un processus

La commande `!heap` des débogueurs Windows standard procure une vue d'ensemble sur les différents tas qui occupent l'espace d'adressage d'un processus.

```
0:024> !heap
        Heap Address      NT/Segment Heap

         2300c040000         Segment Heap
         2300bfd0000              NT Heap
         2300c130000         Segment Heap
         2300c1f0000         Segment Heap
         2302cac0000         Segment Heap
         2302cae0000              NT Heap
         2302d910000              NT Heap
```

Notez que chacun des tas est présenté avec d'une part son l'adresse, d'autre part son type (segmenté ou standard). Le premier tas est le tas par défaut du processus.

Pour obtenir des détails statistiques plus approfondis sur chaque tas, utilisez la commande `heap -s`.

```
0:024> !heap -s

                                    Process    Total      Total
                             Global     Heap Reserved  Committed
    Heap Address  Signature    Flags     List    Bytes      Bytes
                                     Index      (K)        (K)

     2300c040000   ddeeddee        0        1    27692      10900
     2300c130000   ddeeddee     1000        3     3116       2120
     2300c1f0000   ddeeddee     1000        4     1068         16
     2302cac0000   ddeeddee     1000        5     1068         28

************************************************************************************************************************
                                          NT HEAP STATS BELOW
************************************************************************************************************************
LFH Key                   : 0x2c803bdf5f239ea5
Termination on corruption : ENABLED
          Heap     Flags   Reserv  Commit  Virt   Free  List   UCR  Virt  Lock  Fast
                            (k)     (k)    (k)     (k) length      blocks cont. heap
-------------------------------------------------------------------------------------
000002300bfd0000 00008000      64      4     64      2     1     1    0      0
000002302cae0000 00000001      16     16     16      0     3     1    0    N/A
000002302d910000 00000001      16     16     16     10     5     1    0    N/A
-------------------------------------------------------------------------------------
```

La sortie de la commande susmentionnée se présente en deux parties distinctes, avec d'un coté (section supérieure) les tas segmentés, et de l'autre (section inférieure) les tas standards, qui occupent une place dans l'espace d'adressage du processus.

Le bloc PEB fournit un certain nombre de références en ce qui concerne les tas du processus, y compris le nombre de tas que le processus a en sa possession (champ NumberOfHeaps), un pointeur vers le tas par défaut (champ ProcessHeap) et éventuellement d'autres pointeurs vers les tas supplémentaires (champ ProcessHeaps). lkd> dt _PEB NumberOfHeaps @$peb nt!_PEB +0x0e8 NumberOfHeaps : 3 lkd> dt _PEB ProcessHeap @$peb nt!_PEB +0x030 ProcessHeap : 0x0000015a`d1530000 Void lkd> dt _PEB ProcessHeaps @$peb nt!_PEB +0x0f0 ProcessHeaps : **0x00007ff8`b3f90b40** → 0x0000015a`d1530000 Void lkd> dq 0x00007ff8`b3f90b40 00007ff8`b3f90b40 0000015a`d1530000 0000015a`d1380000 00007ff8`b3f90b50 0000015a`d17c0000 00000000`00000000 00007ff8`b3f90b60 00000000`00000000 00000000`00000000 00007ff8`b3f90b70 00000000`00000000 00000000`00000000 00007ff8`b3f90b80 00000000`00000000 00000000`00000000 00007ff8`b3f90b90 00000000`00000000 00000000`00000000 00007ff8`b3f90ba0 00000000`00000000 00000000`00000000 00007ff8`b3f90bb0 00000000`00000000 00000000`00000000

Tableau 10.3. HEAP_INFORMATION_CLASS

ID	Classe	Type de données associé
0x0000	HeapCompatibilityInformation	
0x0001	HeapEnableTerminationOnCorruption	
0x0003	HeapOptimizeResources	HEAP_OPTIMIZE_RESOURCES_-INFORMATION

Types de tas

Dans les versions de Windows antérieures à Windows 10, les allocations qui dépendent du tas sont satisfaites grâce à l'appui soit du tas standard (ou *tas NT*) soit éventuellement du tas à faible fragmentation. L'introduction de Windows 10 a, entre autres ajouts et améliorations, donné l'élan à un nouveau type de tas, dit *segment*.

Dans la configuration par défaut, le tas segment est utilisé par toutes les applications UWP et certains processus système, tandis que le tas standard l'est par l'ensemble des autres processus. Une telle configuration par néanmoins être ajustée via divers aménagements dans le Registre (lesquels seront vus plus tard dans ce chapitre).

Tas NT

Chaque tas standard est géré par l'intermédiaire d'une structure `HEAP`, laquelle se présente comme suit.

```
0:020> dt _HEAP 22de3a40000
ntdll!_HEAP
   +0x000 Segment          : _HEAP_SEGMENT
   +0x000 Entry            : _HEAP_ENTRY
   +0x010 SegmentSignature : 0xffeeffee
   +0x014 SegmentFlags     : 1
   +0x018 SegmentListEntry : _LIST_ENTRY [ 0x0000022d`e3a40120 - 0x0000022d`e3a40120 ]
   +0x028 Heap             : 0x0000022d`e3a40000 _HEAP
   +0x030 BaseAddress      : 0x0000022d`e3a40000 Void
   +0x038 NumberOfPages    : 0x10
   +0x040 FirstEntry       : 0x0000022d`e3a40740 _HEAP_ENTRY
   +0x048 LastValidEntry   : 0x0000022d`e3a50000 _HEAP_ENTRY
   +0x050 NumberOfUnCommittedPages : 0xf
   +0x054 NumberOfUnCommittedRanges : 1
   +0x058 SegmentAllocatorBackTraceIndex : 0
   +0x05a Reserved         : 0
   +0x060 UCRSegmentList   : _LIST_ENTRY [ 0x0000022d`e3a40fe0 - 0x0000022d`e3a40fe0 ]
   +0x070 Flags            : 0x8000
   +0x074 ForceFlags       : 0
   +0x078 CompatibilityFlags : 0
   +0x07c EncodeFlagMask   : 0x100000
   +0x080 Encoding         : _HEAP_ENTRY
   +0x090 Interceptor      : 0
   +0x094 VirtualMemoryThreshold : 0xff00
   +0x098 Signature        : 0xeeffeeff
   +0x0a0 SegmentReserve   : 0x100000
   +0x0a8 SegmentCommit    : 0x2000
   +0x0b0 DeCommitFreeBlockThreshold : 0x100
   +0x0b8 DeCommitTotalFreeThreshold : 0x1000
   +0x0c0 TotalFreeSize    : 0x88
   +0x0c8 MaximumAllocationSize : 0x00007fff`fffdefff
   +0x0d0 ProcessHeapsListIndex : 2
   +0x0d2 HeaderValidateLength : 0x2c0
   +0x0d8 HeaderValidateCopy : (null)
   +0x0e0 NextAvailableTagIndex : 0
   +0x0e2 MaximumTagIndex  : 0
   +0x0e8 TagEntries       : (null)
   +0x0f0 UCRList          : _LIST_ENTRY [ 0x0000022d`e3a40fd0 - 0x0000022d`e3a40fd0 ]
   +0x100 AlignRound       : 0x1f
```

```
+0x108 AlignMask        : 0xffffffff`fffffff0
+0x110 VirtualAllocdBlocks : _LIST_ENTRY [ 0x0000022d`e3a40110 - 0x0000022d`e3a40110 ]
+0x120 SegmentList      : _LIST_ENTRY [ 0x0000022d`e3a40018 - 0x0000022d`e3a40018 ]
+0x130 AllocatorBackTraceIndex : 0
+0x134 NonDedicatedListLength : 0
+0x138 BlocksIndex      : 0x0000022d`e3a402e8 Void
+0x140 UCRIndex         : (null)
+0x148 PseudoTagEntries : (null)
+0x150 FreeLists        : _LIST_ENTRY [ 0x0000022d`e3a40750 - 0x0000022d`e3a40750 ]
+0x160 LockVariable     : 0x0000022d`e3a402c0 _HEAP_LOCK
+0x168 CommitRoutine    : 0x7a8c65e7`60707f48     long  +7a8c65e760707f48
+0x170 StackTraceInitVar : _RTL_RUN_ONCE
+0x178 CommitLimitData  : _RTL_HEAP_MEMORY_LIMIT_DATA
+0x198 FrontEndHeap     : (null)
+0x1a0 FrontHeapLockCount : 0
+0x1a2 FrontEndHeapType : 0 ''
+0x1a3 RequestedFrontEndHeapType : 0 ''
+0x1a8 FrontEndHeapUsageData : (null)
+0x1b0 FrontEndHeapMaximumIndex : 0
+0x1b2 FrontEndHeapStatusBitmap : [129]  ""
+0x238 Counters         : _HEAP_COUNTERS
+0x2b0 TuningParameters : _HEAP_TUNING_PARAMETERS
```

- **BlocksIndex** Voir structure `_HEAP_LIST_LOOKUP`.
- **Masque d'encodage (EncodeFlagMask)** Valeur utilisée pour déterminer si l'en-tête de bloc d'un tas (heap chunk) est chiffré.
- **Options d'allocation (*Flags*)** Contrôle la façon dont les allocations issues de ce tas sont gérées et organisées. Voir fonction HeapCreate ; service RtlSetUserFlagsHeap.
- **Couche de tas frontale (FrontEndHeap)** Pointeur vers la couche de tas frontale éventuellement associée à ce tas. Voir structure `LFH_HEAP`.
- **Gestionnaire (*FrontEndHeapType*)** Détermine si ce tas est géré par la couche de tas centrale (0), la couche de tas frontale (2), ou est une liste look-aside (1). Voir fonction HeapQueryInformation (HeapInformationClass = HeapCompatibilityInformation) et RtlQueryHeapInformation (HeapInformationClass = HeapCompatibilityInformation).
- **Verrou d'exclusion mutuelle (*LockVariable*)** Primitive de verrouillage employée de telle sorte à protéger ce tas lors des opérations qui le nécessite. Voir service RtlEnterHeapLock et RtlLeaveHeapLock.
- *Segment* Collection de structure de données HEAP_SEGMENT qui chacune renferme des informations sur les entrées et segments d'un tas. Nous verrons par la suite comment utiliser ces informations en vue de parcourir les tas d'un processus et localiser les allocations.
- **Empreinte (SegmentSignature)** Distingue un tas standard d'un tas segmenté (0xffeeffee dans un cas, 0xddeeddee dans l'autre).

Tas segment

Sur le plan architectural, le tas segment s'appui sur quatre composants majeurs, chacun d'entre eux ayant dans cette perspective une fonction dédiée vis-à-vis de la manière dont est traitée telle ou telle requête d'allocation : (1) la couche de tas frontale (LFH) gère les demandes d'allocation de mémoire pour des blocs de 16 368 octets ou moins ; (2) l'allocateur de blocs de taille variable porte la responsabilité des demandes d'allocation pour des blocs de 128 Ko ou moins ; (3) la couche de tas centrale encadre les demandes d'allocation pour des blocs de plus de 128 Ko jusqu'à 508 Ko ; et enfin (4), l'allocateur de gros blocs, qui prend en charge les demandes d'allocation de mémoire pour des blocs de plus de 508 Ko.

Par défaut, toutes les applications UWP font usage de tas segmentés. Il en va de même pour quelques processus système fondamentaux, y compris csrss.exe, lsass.exe, services.exe, smss.exe et svchost.exe. À l'opposé, les

applications de bureau emploient quant à elles des tas standard, à cause notamment de diverses réserves sur leur compatibilité.

L'utilisation du tas segmenté peut être configurée au niveau applicatif en ajustant les options d'exécution du fichier associé (FrontEndHeapDebugOptions), où les bits 2 et 3 conditionnent respectivement la désactivation ou l'activation d'une la fonctionnalité. À une échelle plus large, la valeur Enabled sous HKLM\ SYSTEM \CurrentControlSet\Control\Session Manager\Segment Heap sert au même objectif, permettant ainsi de contrôler de manière globale l'utilisation du tas segmenté sur l'ensemble du système.

Chaque tas segmenté est défini par la structure `SEGMENT_HEAP`.

```
0:020> dt _SEGMENT_HEAP 22de3ab0000
ntdll!_SEGMENT_HEAP
   +0x000 EnvHandle        : RTL_HP_ENV_HANDLE
   +0x010 Signature        : 0xddeeddee
   +0x014 GlobalFlags      : 0
   +0x018 Interceptor      : 0
   +0x01c ProcessHeapListIndex : 1
   +0x01e AllocatedFromMetadata : 0y0
   +0x020 CommitLimitData  : _RTL_HEAP_MEMORY_LIMIT_DATA
   +0x020 ReservedMustBeZero1 : 0
   +0x028 UserContext      : (null)
   +0x030 ReservedMustBeZero2 : 0
   +0x038 Spare            : (null)
   +0x040 LargeMetadataLock : 0
   +0x048 LargeAllocMetadata : _RTL_RB_TREE
   +0x058 LargeReservedPages : 0
   +0x060 LargeCommittedPages : 0
   +0x068 StackTraceInitVar : _RTL_RUN_ONCE
   +0x080 MemStats         : _HEAP_RUNTIME_MEMORY_STATS
   +0x0d8 GlobalLockCount  : 0
   +0x0dc GlobalLockOwner  : 0
   +0x0e0 ContextExtendLock : 0
   +0x0e8 AllocatedBase    : 0x0000022d`e3ab3f00  ""
   +0x0f0 UncommittedBase  : 0x0000022d`e3ab4000  ""
   +0x0f8 ReservedLimit    : 0x0000022d`e3abb000  ""
   +0x100 SegContexts      : [2] _HEAP_SEG_CONTEXT
   +0x280 VsContext        : _HEAP_VS_CONTEXT
   +0x340 LfhContext       : _HEAP_LFH_CONTEXT
```

- **Empreinte (SegmentSignature)** Distingue un tas standard d'un tas segmenté (0xffeeffee dans un cas, 0xddeeddee dans l'autre).

Options de contrôle du tas

La liste qui suit passe en revue les diverses options qui influent sur le comportement d'un tas donné.

- **HEAP_CREATE_ENABLE_EXECUTE** Autorise l'exécution de code depuis les blocs de mémoire alloués à partir du tas. (Les pages sous-jacentes sont alors marquées `PAGE_EXECUTE_READWRITE` au lieu de `PAGE_READWRITE`.) Une telle configuration s'avère particulièrement utile dans des contextes tels que la compilation à la volée, où du code machine doit être écrit dans le tas avant d'être exécuté par le processeur.

- **HEAP_GENERATE_EXCEPTIONS** Sollicite le traitement de toute défaillance, telle qu'une condition de mémoire insuffisante, sous la forme d'une exception. En l'absence de cet attribut, les fonctions de tas signalent généralement un échec en renvoyant une valeur nulle.

- **HEAP_NO_SERIALIZE** Transfère à l'appelant la responsabilité de l'exclusion mutuelle, ce qui signifie que les opérations subséquentes n'exigent pas de verrouillage automatique pour assurer leur sérialisation.

- **HEAP_ZERO_MEMORY** Assure une allocation des données avec une initialisation systématique à zéro, réduisant ainsi les probabilités d'erreurs découlant de valeurs inattendues au sein de la mémoire allouée.

ASLR (*Address Space Layout Randomization*)

Pour rendre plus difficile le développement de code malveillant, Windows met en oeuvre la distribution aléatoire de l'espace d'adressage (ASLR, *Address Space Layout Randomization*), laquelle se réfère au placement de certains espaces clés de la mémoire virtuelle (tas, piles, bibliothèques d'exécution…) et consiste à donner un caractère imprévisible à la géographie ainsi faite.

Dans les versions de Windows antérieures à l'introduction du schéma ASLR, il était relativement aisé pour une personne dotée de mauvaises intentions de concocter un scénario d'attaque basé sur un itinéraire précis. Pour transférer le contrôle à une fonction système, par exemple, il suffisait dans cette configuration de connaitre, en fonction de la version du logiciel d'exploitation et de son niveau de service pack, l'adresse de chargement de la fonction désirée et de s'y rendre très simplement via une instruction de saut. Faisant entrer de l'aléa parmi les différentes sections d'un processus, ASLR tend de la sorte à diminuer la faisabilité d'attaques par débordement de tampon.

Popularisée par les systèmes d'exploitation libres, tels qu'OpenBSD ou encore Linux, la technologie ASLR n'a d'abord été disponible sous Windows que via des implémentations tierces pour, à partir de Windows Vista, être intégré dans la configuration par défaut du système.

Dans la perspective ASLR, chaque amorçage du système entraine une topologie nouvelle. Il est très facile de vérifier cela en comparant la configuration actuelle avant et après le redémarrage de la machine. Voici par exemple la liste, telle qu'obtenue via la commande lm (tronquée pour économiser de l'espace), des modules chargés en mémoire par une instance du processus Bloc-Notes.

```
0:000> lm
start             end                 module name
00007ff7`a0090000 00007ff7`a00d1000   notepad    (deferred)
00007ffd`afce0000 00007ffd`afd64000   WINSPOOL   (deferred)
00007ffd`b2d20000 00007ffd`b2ed7000   urlmon     (deferred)
00007ffd`b5210000 00007ffd`b5227000   FeClient   (deferred)
00007ffd`b5a20000 00007ffd`b5da4000   iertutil   (deferred)
00007ffd`b7320000 00007ffd`b732c000   DAVHLPR    (deferred)
00007ffd`b7a90000 00007ffd`b7d04000   COMCTL32   (deferred)
00007ffd`be630000 00007ffd`be7b6000   PROPSYS    (deferred)
00007ffd`c02a0000 00007ffd`c02c9000   bcrypt     (deferred)
00007ffd`c06a0000 00007ffd`c06eb000   powrprof   (deferred)
00007ffd`c06f0000 00007ffd`c0704000   profapi    (deferred)
00007ffd`c0710000 00007ffd`c071f000   kernel_appcore   (deferred)
00007ffd`c0730000 00007ffd`c07e5000   shcore     (deferred)
00007ffd`c07f0000 00007ffd`c0807000   NETAPI32   (deferred)
00007ffd`c0810000 00007ffd`c0e55000   windows_storage   (deferred)
00007ffd`c0e60000 00007ffd`c1048000   KERNELBASE   (deferred)
00007ffd`c1050000 00007ffd`c10d6000   FirewallAPI   (deferred)
00007ffd`c10e0000 00007ffd`c114a000   bcryptPrimitives   (deferred)
00007ffd`c1260000 00007ffd`c12a3000   cfgmgr32   (deferred)
00007ffd`c1530000 00007ffd`c15f1000   OLEAUT32   (deferred)
00007ffd`c1600000 00007ffd`c1756000   USER32     (deferred)
00007ffd`c17a0000 00007ffd`c1a1d000   combase    (deferred)
00007ffd`c1be0000 00007ffd`c1d66000   GDI32      (deferred)
00007ffd`c1d80000 00007ffd`c1e1d000   msvcrt     (deferred)
00007ffd`c1e90000 00007ffd`c1f37000   ADVAPI32   (deferred)
00007ffd`c1f50000 00007ffd`c34ac000   SHELL32    (deferred)
00007ffd`c3630000 00007ffd`c373b000   COMDLG32   (deferred)
00007ffd`c37b0000 00007ffd`c385d000   KERNEL32   (deferred)
00007ffd`c3860000 00007ffd`c397c000   RPCRT4     (deferred)
00007ffd`c3db0000 00007ffd`c3e02000   SHLWAPI    (deferred)
00007ffd`c3e70000 00007ffd`c3ecb000   sechost    (deferred)
00007ffd`c4080000 00007ffd`c4241000   ntdll      (export symbols)       C:\windows\SYSTEM32\ntdll.dll
```

Voici maintenant ce que donne la même commande après le redémarrage.

```
0:000> lm
```

```
start             end                 module name
00007ff7`41d00000 00007ff7`41d41000   notepad    (deferred)
00007ffb`8be70000 00007ffb`8bef4000   WINSPOOL   (deferred)
00007ffb`90770000 00007ffb`90927000   urlmon     (deferred)
00007ffb`92450000 00007ffb`92467000   FeClient   (deferred)
00007ffb`933b0000 00007ffb`93734000   iertutil   (deferred)
00007ffb`95320000 00007ffb`9532c000   DAVHLPR    (deferred)
00007ffb`95450000 00007ffb`956c4000   COMCTL32   (deferred)
00007ffb`9bbc0000 00007ffb`9bd46000   PROPSYS    (deferred)
00007ffb`9dcf0000 00007ffb`9dd19000   bcrypt     (deferred)
00007ffb`9e0f0000 00007ffb`9e104000   profapi    (deferred)
00007ffb`9e110000 00007ffb`9e15b000   powrprof   (deferred)
00007ffb`9e170000 00007ffb`9e17f000   kernel_appcore   (deferred)
00007ffb`9e230000 00007ffb`9e2e5000   shcore     (deferred)
00007ffb`9e2f0000 00007ffb`9e935000   windows_storage   (deferred)
00007ffb`9e940000 00007ffb`9e9c6000   FirewallAPI   (deferred)
00007ffb`9eba0000 00007ffb`9ed88000   KERNELBASE   (deferred)
00007ffb`9ed90000 00007ffb`9eda7000   NETAPI32   (deferred)
00007ffb`9edb0000 00007ffb`9edf3000   cfgmgr32   (deferred)
00007ffb`9ee00000 00007ffb`9ee6a000   bcryptPrimitives   (deferred)
00007ffb`9eed0000 00007ffb`9ef22000   SHLWAPI    (deferred)
00007ffb`9ef30000 00007ffb`9f0b6000   GDI32      (deferred)
00007ffb`9f4f0000 00007ffb`9f59d000   KERNEL32   (deferred)
00007ffb`9f7a0000 00007ffb`9f847000   ADVAPI32   (deferred)
00007ffb`9f850000 00007ffb`a0dac000   SHELL32    (deferred)
00007ffb`a0db0000 00007ffb`a0e71000   OLEAUT32   (deferred)
00007ffb`a0ef0000 00007ffb`a100c000   RPCRT4     (deferred)
00007ffb`a1120000 00007ffb`a1276000   USER32     (deferred)
00007ffb`a1280000 00007ffb`a131d000   msvcrt     (deferred)
00007ffb`a16d0000 00007ffb`a17db000   COMDLG32   (deferred)
00007ffb`a17e0000 00007ffb`a183b000   sechost    (deferred)
00007ffb`a1840000 00007ffb`a1abd000   combase    (deferred)
00007ffb`a1ad0000 00007ffb`a1c91000   ntdll      (export symbols)       C:\windows\SYSTEM32\ntdll.dll
```

Ainsi que pouvez le voir, les bibliothèques d'exécution sont chargés à une adresse différente entre deux amorçages du système, ce qui rend de facto plus difficile pour un attaquant de localiser, et donc tirer profit des fonctionnalités incorporées à ces DLL.

Copy-on-write

La protection de page copy-on-write (copie lors de l'écriture) ouvre la voie à un modèle transactionnel dans lequel les processus qui partagent les mêmes pages le font initialement sur la base de la même vue physique, le gestionnaire de mémoire ayant dans ce contexte pour rôle de différer la copie des pages jusqu'à ce qu'une opération d'écriture ait lieu.

Lorsque deux processus demandent des copies indépendantes d'une même section (soit un ensemble de pages qui présentent les mêmes caractéristiques), le gestionnaire de mémoire place en réalité une seule copie en mémoire partagée et active la propriété de copie sur écriture pour cette région de mémoire. Si l'un des processus tente de modifier les données d'une page protégée en copie sur écriture, le gestionnaire de mémoire passe par toute une série d'actions ayant pour but de conférer au processus qui la sollicite une copie privée de la page. Exemple typique de la façon dont Windows tire avantageusement parti d'une politique d'évaluation paresseuse, une telle façon de procéder permet d'économiser de la place en mémoire centrale (les pages non modifiés ne sont jamais dupliquées), mais aussi potentiellement du temps, en évitant au système d'exploitation un traitement qui pourrait n'avoir aucun bénéficiaire.

Quand un processus écrit dans une page marquée copy-on-write, il y génération d'une infraction de gestion mémoire, laquelle par le biais d'une trappe noyau chemine jusqu'au gestionnaire de mémoire. Ce dernier, plutôt que de répercuter l'infraction en tant que violation d'accès, alloue une nouvelle page lecture/écriture en mémoire physique, duplique le contenu de la page originale dans la nouvelle, actualise les informations de mappage correspondantes et enfin rejette l'exception, forçant de la sorte la ré exécution de l'instruction à l'origine de l'erreur. L'ensemble des éléments est cette fois en place pour que la tâche s'effectue correctement.

Toute page qui résulte d'une copie lors de l'écriture est privative au processus duquel est issue l'opération et n'est pas visible aux autres processus, comme cela est usuellement la règle en matière de régulation de l'espace d'adressage. Les caractéristiques de la page originale restent quant à elles inchangées.

Une utilisation particulièrement répandue de la stratégie copy-on-write concerne les bibliothèque de liens dynamiques (DLL). Ainsi, lorsque Windows tente de charge une DLL pour le compte d'une application, il le fait en s'appuyant en propriété sur l'adresse de chargement par défaut dudit module, permettant de la sorte à plusieurs applications de partager les mêmes pages physiques. Si, pour une raison quelconque, ce schéma ne peut être respecté, la DLL est alors chargée à une autre adresse, et les pages sous-jacentes marquées copie-sur-écriture. (Bien que les DLL aient par nature une orientation indépendante de la position en mémoire, certains correctifs sont néanmoins indispensables.)

Une autre utilisation du copy-on-write résulte de la manière dont les débogueurs manipulent programmes et processus. Dans la perspective de tels outils, il est important de s'assurer que toutes les modifications apportées à une application se limitent à la seule instance en cours de débogage. Considérons, à titre d'exemple, l'ajout d'une instruction de point d'arrêt (breakpoint) à une page de code. En pareille circonstance, le débogueur commence par faire passer la protection de la page à PAGE_EXECUTE_READWRITE, puis altère le flux d'instructions. Comme la page fait partie d'une section mappée, le gestionnaire de mémoire confère au processus qui renferme le point d'arrêt une copie privée de la page ; les autres processus continuent d'utiliser la page de code non modifiée.

Réservation et engagement de pages

Les pages de l'espace d'adressage d'un processus sont libres, réservées ou engagées, à quoi correspond en interne leur trajet entre mémoire virtuelle et mémoire centrale (et vice versa).

Une page dite libre l'est au sens de sa disponibilité vis-à-vis de toute application qui en ferait la demande. Les pages entrant dans cette catégorie sont stockées dans une liste de pages libres, et contiennent du reste des données encore non initialisées à ce stade. Toute tentative de lire ou écrire dans cette région provoque une violation d'accès.

L'espace d'adressage réservé permet à un thread de retenir pour son propre compte une plage d'adresses virtuelles. Toute tentative d'accéder à de la mémoire réservée entraine une violation d'accès, la page n'ayant dans l'immédiat aucun stockage qui lui est associé.

Les pages engagées sont des pages valides de la mémoire physique. L'accès à une page engagé est régulé par différentes options de protection. Même au stade engagé, une page n'est seulement chargée en mémoire physique que lors du premier accès - autrement dit la première fois que le système a à résoudre une référence se rapportant à elle. Les pages engagées sont soit privées et non partageables, soit mappées à une vue d'une section (qui peut être mappée par d'autres processus).

Si les pages sont privatives pour le processus et qu'aucun accès n'a eu lieu les concernant, elles sont crées lors du premier accès sous la forme de pages remplies de zéros. Les pages engagées qui sont privatives sont inaccessibles à tous les autres processus, sauf si l'on y accède via les fonctions de mémoire inter processus conçues spécialement dans cette optique (Read/WriteProcessMemory). En ce qui concerne leur pérennité sur disque, les pages sont en la matière traitées selon le schéma habituel : les pages transitent du working set du processus à la liste des pages modifiées, puis sont finalement écrites sur le support de stockage approprié. Les pages de fichier mappé peuvent également être enregistrés sur disque via FlushViewOfFile.

Les opérations de réservation et d'engagement peuvent être simultanées ou séparées dans le temps, suivant les besoins de l'application. Un processus peut ainsi dans un premier temps réserver de l'espace d'adressage, puis par la suite engager des pages de cette région. Une autre possibilité consiste à réunir la réservation et l'engagement en un seul et même acte. Les valeurs MEM_RESERVE et MEM_COMMIT sont auquel cas utilisées conjointement lors de l'appel à VirtualAlloc ou VirtualAllocEx.

Les fonctions Windows de gestion mémoire tiennent compte d'un paramètre que les applications peuvent utiliser afin de spécifier à quelle adresse virtuelle la mémoire est allouée. Si elles ne le font pas, c'est alors le gestionnaire

de mémoire qui recherche l'emplacement le plus approprié à chaque situation. L'adresse de base d'une région allouée est toujours un multiple entier de la granularité d'allocation du système (0x100000).

Une application appelle la fonction VirtualFree dans le but de désengager des pages et/ou libérer de l'espace d'adressage. La différence entre désengagement et libération est du même genre qu'entre réservation et engagement : la mémoire désengagée reste réservée, alors que la mémoire libérée n'est ni engagée ni réservée. Les pages engagées étant une ressource système précieuse, il convient de de les libérer sitôt qu'elles ne présentent plus d'utilité objective à court terme.

Une visée essentielle du processus de réservation/engagement en deux temps est de réduire la consommation de mémoire vive. Réserver de la mémoire est à cet égard une opération relativement rapide, et surtout peu onéreuse, dans la mesure où cela ne consomme pas de page physique ni n'a d'impact sur les quotas facturés à un processus. Une telle manière d'agir a également pour intérêt de donner aux applications fortement consommatrices de mémoire la possibilité de mieux disposer de leurs ressources. (Notez que nous faisons référence ici à la quantité utilisée, non au nombre de transactions.) Par exemple, un processus ayant besoin d'un tampon mémoire potentiellement large et virtuellement continu peut, plutôt que d'engager des pages pour tout cet espace, réserver un plus ou moins grand nombre de pages et n'en engager qu'une partie, petit à petit et en fonction de l'évolution des besoins.

#	Fonctions	Services
1	VirtualAlloc	NtAllocateVirtualMemory
2	VirtualFree	NtFreeVirtualMemory
3	VirtualLock	NtLockVirtualMemory
4	VirtualUnlock	NtUnlockVirtualMemory
5	VirtualProtect	NtProtectVirtualMemory
6	VirtualQuery	NtQueryVirtualMemory

Structures	Fonctions apparentées
MEMORY_BASIC_INFORMATION	VirtualQuery
WIN32_MEMORY_REGION_INFORMATION	QueryVirtualMemoryInformation

1. Réserver ou/et engager une région de pages dans l'espace d'adressage du processus appelant.

2. Libèrer ou/et désengager une région de pages dans l'espace d'adressage du processus appelant.

3. Verrouiller une région de pages de l'espace d'adressage du processus appelant dans la mémoire physique, ce qui de la sorte court-circuite l'éventualité d'un défaut de page pour la plage d'adresses concernée.

4. Déverrouiller une région de pages de l'espace d'adressage du processus appelant, redonnant ainsi la possibilité au système de remiser si nécessaire ces pages vers un fichier de pagination.

5. Modifier les paramètres qui régissent l'accès à une région de pages dans l'espace d'adressage virtuel du processus appelant. Pour plus d'informations, consultez la section \<\<_protection_de_la_mémoire\>\>.

6. Obtenir des informations concernant une région de pages dans l'espace d'adressage virtuel du processus appelant.

Informations concernant l'espace d'adressage

Les applications souhaitant obtenir des informations concernant l'espace d'adressage peuvent le faire par le biais des fonctions VirtualQuery et VirtualQueryEx. Fortement similaires dans le principe, à savoir fournir une vue sur une région déterminée de pages, VirtualQuery n'est utilisable que dans le contexte du processus appelant, tandis que VirtualQueryEx l'est pour n'importe quel processus - pourvu que son contexte de sécurité le permette.

Le couple VirtualQuery/VirtualQueryEx fonctionne à partir d'une adresse mémoire spécifiée lors de l'appel, et dont la valeur sert à ce moment de base pour une gamme de pages virtuelles adjacentes partageant les mêmes

caractéristiques. Les attributs considérés à cet égard incluent non seulement l'état global des pages (réservé ou engagé, réserve seulement, et ainsi de suite), mais aussi les options qui les protègent (lecture/écriture, lecture seule, pas d'exécution, etc.)

Lors de l'exécution, les fonctions VirtualQuery et VirtualQueryEx s'appuient sur le service système NtQueryVirtualMemory de sorte à alimenter une structure de type MEMORY_BASIC_INFORMATION.

Tableau 10.4. MEMORY_INFORMATION_CLASS

ID	Classe	Type de données associé
0x0000	MemoryBasicInformation	MEMORY_BASIC_INFORMATION
0x0001	MemoryWorkingSetList	MEMORY_WORKING_SET_INFORMATION
0x0002	MemorySectionName	MEMORY_SECTION_NAME
0x0003	MemoryRegionInformation	MEMORY_REGION_INFORMATION
0x0004	MemoryWorkingSetExInformation	MEMORY_WORKING_SET_EX_-INFORMATION
0x0005	MemorySharedCommitInformation	MEMORY_SHARED_COMMIT_-INFORMATION
0x0006	MemoryImageInformation	MEMORY_IMAGE_INFORMATION

MEMORY_BASIC_INFORMATION.

```
typedef struct _MEMORY_BASIC_INFORMATION {
    PVOID BaseAddress;
    PVOID AllocationBase;
    ULONG AllocationProtect;
    SIZE_T RegionSize;
    ULONG State;
    ULONG Protect;
    ULONG Type;
} MEMORY_BASIC_INFORMATION, *PMEMORY_BASIC_INFORMATION;
```

MEMORY_SECTION_NAME.

```
typedef struct _MEMORY_SECTION_NAME {
    UNICODE_STRING SectionFileName;
} MEMORY_SECTION_NAME, *PMEMORY_SECTION_NAME;
```

Stratégies d'allocation

Le processus de réservation/engagement de pages permet trois sortes de stratégies :

- *Allocation et engagement simultanés* La région est allouée en couplant en une seule opération la réservation et l'engagement des pages. Cette stratégie est notamment la plus adéquate pour des processus qui, par conception, consomme rapidement des portions entières de leur espace d'adressage.
- *Allocation et engagement si nécessaire* La région est allouée via réservation d'un certain nombre de pages qui ne sont engagés que lorsque les circonstances l'exigent. Cette stratégie vise à rationaliser l'usage de la mémoire en n'engageant que les pages strictement nécessaires à un moment donné de l'exécution.
- *Allocation et engagement si nécessaire avec page de garde* La région est allouée en s'appuyant sur des pages réservés, non engagées, et marquées comme page de garde (PAGE_GUARD). La prise en compte des besoins en matière de pages se fait auquel cas au travers du gestionnaire d'exception, et plus particulièrement de la résolution de la faute de page de garde (STATUS_PAGE_GUARD) pour engager des pages.

Protection de la mémoire

D'ordre général, Windows protège la mémoire de façon qu'aucun processus mode utilisateur ne puisse, par inadvertance ou intention, altérer l'espace d'adressage d'un autre processus ou du système d'exploitation lui-même. Windows assure cette protection de trois grandes façons.

Primo, tout le code et toutes les structures de données qui évoluent dans l'espace d'adressage du système ne sont accessibles que depuis le mode noyau ; les threads mode utilisateur ne peuvent pas accéder aux régions sous-jacentes de la mémoire virtuelle. S'ils essaient de le faire, le matériel relève une infraction d'accès et génère une erreur que le gestionnaire de mémoire répercute au thread incriminé en mettant fin à son exécution.

Secondo, chaque processus a un espace d'adressage privé auquel ne peut accéder aucun thread d'un autre processus. Les seules exceptions concernent le cas où le processus entreprend de partager des pages avec d'autres processus, ou le cas où un autre processus bénéficie de droits d'accès lui conférant la possibilité d'interagir avec la mémoire du processus, et exerce ces dits droits en utilisant des fonctions inter processus telles que *ReadProcessMemory* et *WriteProcessMemory*.

Tertio, toutes les plates-formes sur lesquelles Windows prévoit de fonctionner sont munies, à des degrés divers, d'une forme de protection matérielle de la mémoire (du genre lecture/écriture, lecture seule, pas d'exécution, etc.). Par exemple, les pages de l'espace d'adressage d'un processus qui abritent du code machine exécutable sont marqués lecture seule, et de la sorte protégées contre toute modification.

Le tableau ci-après énumère les options de protection de la mémoire définies dans l'API Windows.

Attribut	**Valeur**	**Signification**
PAGE_NOACCESS	0x00000001	Toute tentative de lire, écrire, ou d'exécuter du code dans cette région provoque une violation d'accès.
PAGE_READONLY	0x00000002	Toute tentative d'écrire dans cette région (ou d'y exécuter du code si le processeur intègre la protection contre l'exécution) provoque une violation d'accès ; les opérations de lecture sont en revanche autorisées.
PAGE_READWRITE	0x00000004	La page est lisible et inscriptible, mais pas exécutable.
PAGE_WRITECOPY	0x00000008	Toute tentative d'écrire en mémoire dans cette région force le système à donner au processus une copie privative de la page.
PAGE_EXECUTE	0x00000010	Toute tentative d'écrire du code dans cette région provoque une violation d'accès, mais l'exécution (ainsi que la lecture) est permise.
PAGE_EXECUTE_READ	0x00000020	Toute tentative d'écrire du code dans cette région provoque une violation d'accès, mais l'exécution et la lecture sont permises.
PAGE_EXECUTE_READWRITE	0x00000040	La page est lisible, inscriptible et exécutable.
PAGE_GUARD	0x00000100	Toute tentative d'accès à une page de garde déclenche une exception EXCEPTION_-GUARD_PAGE et désactive le statut de page de garde avant de passer le relais à la protection de page sous-jacente. Les pages de garde agissent ainsi au même titre qu'une alarme d'accès. Notez que ce flag peut accompagner toutes les autres options de protection énumérées ici, sauf PAGE_NOACCESS.
PAGE_NOCACHE	0x00000200	La page est exclue des schémas de mise en cache du système. Cette option, généralement déconseillée, existe essentiellement pour les pilotes dont le fonctionnement normal implique de manipuler des données pour lesquelles il est plus avantageux de procéder ainsi (tampon de trames vidéo par exemple).- Notez que ce flag peut accompagner toutes les autres options de protection énumérées

Attribut	Valeur	Signification
		ici, sauf PAGE_GUARD, PAGE_NOACCES et PAGE_WRITECOMBINE.
PAGE_WRITECOMBINE	0x00000400	La page supporte les écritures combinées.- Lorsque cette option est activée, le processeur court-circuite la mise en cache des écritures (qui peuvent générer un trafic mémoire plus ou moins important) et tente d'agréger (et ce faisant de différer) les requêtes d'écriture, cela dans l'optique d'optimiser les performances. Plusieurs écritures consécutives à la même adresse peuvent par exemple dans ce scénario être répercutées par une seule écriture en mémoire, en l'occurence la dernière. Notez que ce flag peut accompagner toutes les autres options de protection énumérées ici, sauf PAGE_GUARD, PAGE_NOACCES et PAGE_NOCACHE.

Plusieurs fonctions de l'API Windows tiennent compte des quelques options que nous venons de passer en revue, parmi lesquelles VirtualAlloc et VirtualProtect. Quand un processus réserve une région de mémoire ou engage effectivement les pages d'une région, il précise quels attributs de protection doivent être en vigueur. Parallèlement à cela, la fonction VirtualProtect offre la possibilité de modifier à tout moment les permissions en oeuvre en ce qui concerne des pages déjà engagées.

Pour voir quelles options de protection sont en vigueur en ce qui concerne une région de mémoire, utiliser la commande !address.

DEP (*Data Execution Prevention*)

La prévention de l'exécution des données (DEP, Data Execution Prevention) instaure un schéma de protection de la mémoire dans lequel toute tentative de transférer le contrôle à une instruction située dans une page marquée "pas d'exécution" génère une infraction d'accès.

Sous leurs formes les plus abouties, les fonctions prises en charge par DEP le sont au niveau matériel, incarnées en l'occurence par un ensemble de technologies désignés conjointement sous l'appellation NX/XD. Différents registres spéciaux du processeur sont alors mis à contribution dans l'optique de distinguer quelles pages de mémoire encapsulent du code machine exécutable, et lesquelles contiennent des données. Si, pour une raison ou pour une autre, le système tente d'exécuter une instruction stockée dans une zone prévue pour ne renfermer que des données, le processeur s'en rend compte et informe le système d'exploitation qu'un problème est survenu. Celui-ci est alors en mesure d'interrompre l'application afin de l'empêcher d'exécuter du code potentiellement nuisible. Si la configuration matérielle ne peut donner lieu à une telle forme de protection, Windows se replie auquel cas sur une voie entièrement logicielle. Pour plus de détails sur ce dernier point, consultez la section "DEP de niveau logiciel".

S'il y a tentative en mode noyau d'exécuter du code d'une page marquée "pas d'exécution", le système s'effondre avec le code de contrôle ATTEMPTED_EXECUTE_OF_NOEXECUTE_MEMORY. Si la tentative se fait en mode utilisateur, une exception STATUS_ACCESS_VIOLATION est dès lors envoyée au thread duquel est issue la référence illégale. Quand un processus alloue de la mémoire qui doit être exécutable, il doit le faire en spécifiant l'une ou l'autre des constantes de protection mémoire établies à cet égard, à savoir PAGE_EXECUTE, PAGE_EXECUTE_READ, PAGE_EXECUTE_READWRITE, et PAGE_EXECUTE_WRITECOPY.

Sous Windows 64 bits, la protection contre l'exécution est appliquée aux piles de thread (pile mode utilisateur et pile mode noyau), aux pages mode utilisateur qui ne sont pas marquées spécifiquement comme étant exécutables, au pool paginé du noyau et au pool de session du noyau. Les versions 32 bits de Windows offrent par contraste un degré de protection moindre, dans la mesure où la protection contre l'exécution ne concerne dans ce contexte que les piles de thread et les pages utilisateur, mais pas le pool paginé ni le pool de session. Pour subvenir à d'éventuels problèmes de compatibilité, Windows permet de configurer individuellement chaque

programme en fonction de son rapport à la prévention de l'exécution des données. Appliquez les procédures suivantes en vue de désactiver DEP pour une application donnée.

- Ouvrez la boîte de dialogue Propriétés système.
- Allez dans l'onglet Paramètres système avancés, puis dans la section Performances, cliquez sur le bouton Paramètres.
- Sélectionnez l'onglet Prévention de l'exécution des données.
- Cliquez sur Activer la Prévention de l'exécution des données pour tous les programmes et les services, sauf ceux que je sélectionne, puis cliquez sur Ajouter.

Windows prend en charge quatre configurations relatives à DEP :

- OptIn Active la protection DEP pour les images système fondamentales de Windows.
- OptOut Active la protection DEP pour tous les exécutables, sauf ceux spécifiés. Vous pouvez créer manuellement une liste de programmes spécifiques pour lesquels la fonctionnalité n'est pas appliquée à l'aide de la boite de dialogue Propriétés système.
- AlwaysOn Active la protection DEP pour tous les composants, sans aucune possibilité d'exclure certaines applications.
- AlwaysOff Désactive la protection DEP. La configuration ainsi obtenue n'offre aucune couverture à aucun des éléments du système, raison pour laquelle il est déconseillé d'opter pour ce paramètre.

Tableau 10.5. Interfaces concernant DEP

Fonction
GetProcessDEPPolicy
GetSystemDEPPolicy
SetProcessDEPPolicy
GetProcessMitigationPolicy (ProcessDEPPolicy)
SetProcessMitigationPolicy (ProcessDEPPolicy)

PROCESS_MITIGATION_DEP_POLICY.

```
typedef struct _PROCESS_MITIGATION_DEP_POLICY {
  union {
    DWORD  Flags;
    struct {
      DWORD Enable : 1;
      DWORD DisableAtlThunkEmulation : 1;
      DWORD ReservedFlags : 30;
    };
  };
  BOOLEAN Permanent;
} PROCESS_MITIGATION_DEP_POLICY, *PPROCESS_MITIGATION_DEP_POLICY;
```

DEP de niveau logiciel

Dans l'éventualité où le processeur exécutant Windows n'intègre pas de protection des pages contre l'exécution, Windows fournit néanmoins une protection DEP logicielle limitée, qui assure une surveillance accrue de la façon dont un programme gère les exceptions.

Si la construction des fichiers image du programme intègre le traitement structuré sécurisé des exceptions (SafeSEH), avant qu'une exception ne soit relayée, le système vérifie que le gestionnaire d'exception est déclaré dans la table des fonctions contenue dans le fichier image. S'il ne peut s'appuyer sur une telle prise en charge, le DEP logiciel veille à ce que, avant la levée d'une exception, le gestionnaire d'exception se situe dans une région mémoire dont les pages sont marquées exécutable.

Peut être en avez vous déjà pris conscience à la lecture de ce qui précède, mais sinon remarquez que bien qu'on puisse présumer un lien étroit entre le DEP logiciel et la prévention de l'exécution de code dans les pages n'étant pas destinées à en contenir (comprendre DEP au niveau matériel), cette relation n'est pas si tangible, et accorde une forme de protection différente.

Verrouillage de la mémoire

La façon dont Windows gère les ressources de l'ordinateur, orientée par l'économie de la mémoire physique, entraine selon les circonstances un transport des pages depuis la mémoire principale vers la mémoire auxiliaire. Pour échapper à ce scénario, les applications se voient confier la possibilité de verrouiller une partie des pages de leur espace d'adressage, et ce faisant donner à celles-ci un statut de résident dans la mémoire physique.

Les applications Windows désireuses de verrouiller des pages peuvent le faire par l'intermédiaire de la fonction VirtualLock. Le nombre de pages qu'un processus peut verrouiller ne peut pas dépasser sa taille minimale de working set moins huit pages, un chiffre du reste volontairement bas afin de limiter une potentielle dégradation des performances. Par conséquent, si un processus doit verrouiller plus de pages que ce à quoi il est actuellement restreint, il doit pour cela préalablement accroitre son minimum de working (fonction SetProcessWorkingSetSize dans l'API Windows).

Les pilotes de périphérique peuvent appeler les fonctions mode noyau MmProbeAndLockPages, MmLockPagableCodeSection, MmLockPagableDataSection et MmLockPagableSectionByHandle. Les pages ainsi verrouillées restent en mémoire jusqu'à ce qu'elles soient explicitement déverrouillées. Un pilote ne peut pas verrouiller plus de pages que ne le permet le compteur de pages disponibles résidentes. (variable noyau MmResidentAvailablePages.)

Les pilotes de périphérique ont la possibilité de verrouiller des pages à l'aide des routines MmLockPagableCodeSection, MmLockPagableDataSection et MmLockPagableSectionByHandle. Les pages ainsi verrouillées restent en mémoire jusqu'à ce qu'elles soient explicitement déverrouillées. Contrairement aux applications, qui se voient soumises à de strictes obligations en ce qui concerne les dimensions de leur working set, il n'existe stricto sensu pas de limite au nombre de pages que du code système peut verrouiller, à ceci près qu'un pilote ne peut verrouiller plus de pages que ne le permet le compteur de pages disponibles résidentes (variable noyau MmResidentAvailablePages).

Le fait de verrouiller des pages en mémoire physique est surtout intéressant pour des applications dont la rapidité d'exécution est cruciale, ou qui effectuent des opérations très fréquentes sur certains emplacements de mémoire. À l'usage, un tel mécanisme peut par contre se montrer particulièrement handicapant pour les performances du système, cela en réduisant la quantité de mémoire vive disponible et en amenant une utilisation intensive du fichier d'échange, avec en perspective une importante augmentation du nombre d'E/S. Dans la grande majorité des cas, le gestionnaire de mémoire de Windows sait mieux que quiconque quelles pages doivent subsister en mémoire principale, et lesquelles devraient être retirées.

Modified Page Writer

Lorsque le nombre de pages physiques libres se retrouve en dessous d'un seuil minimum, un thread spécial appelé Écrivain de pages modifiées (*modified page writer*) a pour fonction de remiser sur le disque (écrire dans les fichiers de pagination idoines) les pages appartenant à la liste des pages modifiées.

Le gestionnaire de mémoire donne lieu à l'écrivain de pages modifiées (routine MiModifiedPageWriter) lors de la phase d'initialisation du système. Par la suite, le code noyau d'ordonnancement réveille ce thread chaque fois qu'il est nécessaire de procéder à une réduction de la taille de la liste des pages modifiés.

Balance Set Manager

Pour équilibrer dynamiquement les demandes de mémoire physique entre les différents processus, Windows planifie à intervalles réguliers un thread spécial, nommé Balance Set Manager, chargé de contrôler et éventuellement initier divers événements liés à la gestion de la mémoire et, dans une moindre mesure, à l'ordonnancement :

- Suivi des pages modifiées
- Expansion et élagage de working set
- Création de nouveaux threads système auxiliaires
- Ajustement de la profondeur des listes look-aside
- Conciliation du nombre de tampons de trace ETW
- Application des délais d'exécution des processus

Le thread correspondant au balance set manager est créé lors de l'initialisation du système et exécute la routine KeBalanceSetManager.

En interne, KeBalanceSetManager attend deux objets événement : l'un associé à une minuterie (timer) périodique réglée pour se déclencher une fois par seconde, et l'autre, que le gestionnaire de mémoire configure à l'état signalé en diverses occasions, par exemple quand le système subit un fort taux de défauts de page.

Quand le balance set manager se réveille suite au déclenchement du timer, il effectue les opérations suivantes :

- Ajustement éventuel de la profondeur des listes look-aide (routine ExAdjustLookasideDepth).
- Appel du gestionnaire de working set, qui fait partie de gestionnaire de mémoire.
- Recherche des thread susceptibles de mériter une augmentation de leur priorité.
- Une fois sur huit, signalement d'un événement qui réveille un autre thread système appelé swapper (routine KeSwapProcessOrStack)

Examen du thread exécutant l'écrivain de pages modifiés

```
lkd> !object poi(nt!MiModifiedPageWriterThread)
Object: fffffa80024d7b00  Type: (fffffa80024716c0) Thread
    ObjectHeader: fffffa80024d7ad0 (new version)
    HandleCount: 0  PointerCount: 1

lkd> dt nt!_KTHREAD fffffa80024d7b00
   +0x000 Header           : _DISPATCHER_HEADER
   +0x018 SListFaultAddress : (null)
   +0x020 QuantumTarget    : 0x16409fa0
   +0x028 InitialStack     : 0xfffff880`02324dd0 Void
   +0x030 StackLimit       : 0xfffff880`0231f000 Void
   +0x038 StackBase        : 0xfffff880`02325000 Void
   .

lkd> dt nt!_ETHREAD StartAddress fffffa80024d7b00
   +0x370 StartAddress : 0xfffff801`b3d51a0c Void

lkd> ln 0xfffff801`b3d51a0c
(fffff801`b3d51a0c)   nt!MiModifiedPageWriter   |  (fffff801`b3d51cb8)   nt!MiUseLowIoPriorityForModifiedPages
Exact matches:
    nt!MiModifiedPageWriter (<no parameter info>)
```

Défauts de page

Un défaut de page correspond à une série d'événements se déroulant lorsqu'une instruction référence le contenu d'une page qui ne se trouve pas en mémoire physique. Le système d'exploitation doit auquel cas faire face aux circonstances en permettant, d'une manière ou d'une autre, l'accès à la mémoire des données recherchées afin que l'application puisse continuer de s'exécuter comme si l'erreur n'avait jamais eu eu lieu. De ce fait, quand une interruption de type défaut de page se produit, le thread duquel est issu la faute est mis en état d'attente tandis que Windows s'occupe de résoudre l'accès.

Quand un thread fait référence à une page mémoire non résidente, une interruption matérielle est soulevée, marquant l'instruction fautive et transférant le contrôle au gestionnaire de défaut de page (page fault handler). Celui-ci vérifie la validité de l'adresse demandée et, s'il remarque que la page n'est que temporairement inaccessible, essaie d'en localiser une copie dans le fichier de pagination. S'il réussit, la page est chargée du disque vers la mémoire, inclus à l'ensemble de travail du thread et l'instruction fautive exécuté de nouveau.

On distingue deux types de défaut de page, matériel et logiciel. Un défaut de page matériel se produit lorsqu'un processus nécessite du code ou des données qui ne sont ni dans son jeu de travail, ni ailleurs en mémoire physique, et qui doivent par conséquent être récupérés à partir du disque. Un défaut de page logiciel n'implique pas cette procédure et survient quand :

- La page se trouve dans le working set d'un autre processus.
- La page est marquée en transition.
- La page est allouée mais n'a pas encore été accédé.

La plupart des processeurs peuvent sans conséquence notable affronter un grand nombre de défauts logiciels. Les défauts matériels, par contre, du fait de la nature des opérations qu'ils suscitent (en l'occurence des E/S disque), peuvent entraîner des délais significatifs.

Le gestionnaire d'exception associé à un défaut de page est la routine KiPageFault, dont l'entrée se fait à condition d'une référence à une page non présente dans l'espace physique, ou bien si la page ne permet pas le niveau d'accès demandé. Un cadre d'exception standard est construit sur la pile noyau, ainsi qu'un code d'erreur précisant les raisons du défaut de page, ces données présentant assez d'informations pour pouvoir relancer l'instruction fautive. La routine demande ensuite au gestionnaire mémoire de résoudre la faute.

Compteur	Description
Défauts de page/s	Nombre moyen de défauts de page par secondes, incluant les défauts matériels et les défauts logiciels.
Mémoire\Défauts de page/s	Nombre de défauts de page par secondes pour des pages initialisées à zéro.
Lectures de pages/s	Vitesse à laquelle le disque a été lu pour résoudre les défauts de page matériels.
Défauts en transit/s	Vitesse à laquelle les erreurs de page sont résolues en récupérant les pages qui étaient en cours d'utilisation par un autre processus partageant la page ou qui étaient sur la liste des pages modifiées, ou la liste des pages en attente ou en train d'être écrites sur le disque au moment du défaut de page.
Pages en entrée/s	Vitesse (Nombre de pages par seconde) à laquelle des pages ont étées lues depuis le disque pour résoudre des défauts de page matériels.
Pages en sorties/s	Vitesse (Nombre de pages par seconde) à laquelle des pages ont étées écrites sur le disque pour libérer de la mémoire physique.

Élément	Type	Lieu	Description / Autres références
PageFaultCount	ULONG	SYSTEM_FILECACHE_-INFORMATION	Voir service NtQuerySystemInformation (SystemInformationClass = SystemFileCacheInformation)
PageFaultCount	ULONG	SYSTEM_PERFORMANCE_-INFORMATION	
PageFaultCount	ULONG	SYSTEM_PROCESS_-INFORMATION	

Collision de défauts de page

Quand un thread subit un défaut de page se rapportant à une page en cours de rechargement (des suites d'une opération émanant d'un autre thread), on parle alors de *collision de défauts de page*. Le gestionnaire de mémoire

décèle de tels phénomènes en remarquant d'une part que la page est en transition (à savoir qu'elle n'est présente dans aucune aucun working set ni aucune liste de pagination), d'autre part qu'une E/S en cours (par exemple une lecture de page) la concerne. En pareil cas, le gestionnaire de mémoire émet une opération d'attente sur l'objet événement spécifié dans l'enregistrement PFN approprié.

Une fois l'opération d'E/S terminée, tous les threads qui attendaient l'événement voient leur attente satisfaite. Le premier thread à acquérir le verrou de base de données PFN a pour responsabilité d'effectuer les opérations de finalisation du rechargement de page, parmi celles-ci : vérifier le code d'état d'E/S, rénitialiser l'attribut indiquant une E/S en cours, et mettre à jour le PTE de sorte qu'il pointe vers la page physique.

Page système partagée

```
lkd> !kuser
_KUSER_SHARED_DATA at fffff78000000000
TickCount:    fa00000 * 000000000004c219 (0:01:21:12.390)
TimeZone Id: 2
ImageNumber Range: [8664 .. 8664]
Crypto Exponent: 0
SystemRoot: 'C:\WINDOWS'
BootId: 3

lkd> dtx nt!_KUSER_SHARED_DATA fffff78000000000
(*((nt!_KUSER_SHARED_DATA *)0xfffff78000000000))                [Type: _KUSER_SHARED_DATA]
    [+0x000] TickCountLowDeprecated : 0x0 [Type: unsigned long]
    [+0x004] TickCountMultiplier : 0xfa00000 [Type: unsigned long]
    [+0x008] InterruptTime    [Type: _KSYSTEM_TIME]
    [+0x014] SystemTime       [Type: _KSYSTEM_TIME]
    [+0x020] TimeZoneBias     [Type: _KSYSTEM_TIME]
    [+0x02c] ImageNumberLow   : 0x8664 [Type: unsigned short]
    [+0x02e] ImageNumberHigh  : 0x8664 [Type: unsigned short]
    [+0x030] NtSystemRoot     : "C:\WINDOWS" [Type: wchar_t [260]]
    [+0x238] MaxStackTraceDepth : 0x0 [Type: unsigned long]
    [+0x23c] CryptoExponent   : 0x0 [Type: unsigned long]
    [+0x240] TimeZoneId       : 0x2 [Type: unsigned long]
    [+0x244] LargePageMinimum : 0x200000 [Type: unsigned long]
    [+0x248] AitSamplingValue : 0x0 [Type: unsigned long]
    [+0x24c] AppCompatFlag    : 0x0 [Type: unsigned long]
    [+0x250] RNGSeedVersion   : 0xd [Type: unsigned __int64]
    [+0x258] GlobalValidationRunlevel : 0x0 [Type: unsigned long]
    [+0x25c] TimeZoneBiasStamp : 42 [Type: long]
    [+0x260] NtBuildNumber    : 0x3fab [Type: unsigned long]
    [+0x264] NtProductType    : NtProductWinNt (1) [Type: _NT_PRODUCT_TYPE]
    [+0x268] ProductTypeIsValid : 0x1 [Type: unsigned char]
    [+0x269] Reserved0        [Type: unsigned char [1]]
    [+0x26a] NativeProcessorArchitecture : 0x9 [Type: unsigned short]
    [+0x26c] NtMajorVersion   : 0xa [Type: unsigned long]
    [+0x270] NtMinorVersion   : 0x0 [Type: unsigned long]
    [+0x274] ProcessorFeatures [Type: unsigned char [64]]
    [+0x2b4] Reserved1        : 0x7ffeffff [Type: unsigned long]
    [+0x2b8] Reserved3        : 0x80000000 [Type: unsigned long]
    [+0x2bc] TimeSlip         : 0x0 [Type: unsigned long]
    [+0x2c0] AlternativeArchitecture : StandardDesign (0) [Type: _ALTERNATIVE_ARCHITECTURE_TYPE]
    [+0x2c4] BootId           : 0x3 [Type: unsigned long]
    [+0x2c8] SystemExpirationDate : {0} [Type: _LARGE_INTEGER]
    [+0x2d0] SuiteMask        : 0x110 [Type: unsigned long]
    [+0x2d4] KdDebuggerEnabled : 0x1 [Type: unsigned char]
    [+0x2d5] MitigationPolicies : 0xa [Type: unsigned char]
    [+0x2d5 ( 1: 0)] NXSupportPolicy  : 0x2 [Type: unsigned char]
    [+0x2d5 ( 3: 2)] SEHValidationPolicy : 0x2 [Type: unsigned char]
    [+0x2d5 ( 5: 4)] CurDirDevicesSkippedForDlls : 0x0 [Type: unsigned char]
    [+0x2d5 ( 7: 6)] Reserved         : 0x0 [Type: unsigned char]
    [+0x2d6] Reserved6        [Type: unsigned char [2]]
    [+0x2d8] ActiveConsoleId  : 0x1 [Type: unsigned long]
```

```
[+0x2dc] DismountCount    : 0x0 [Type: unsigned long]
[+0x2e0] ComPlusPackage   : 0x1 [Type: unsigned long]
[+0x2e4] LastSystemRITEventTickCount : 0x4a6fe8 [Type: unsigned long]
[+0x2e8] NumberOfPhysicalPages : 0x7fedb [Type: unsigned long]
[+0x2ec] SafeBootMode     : 0x0 [Type: unsigned char]
[+0x2ed] VirtualizationFlags : 0x0 [Type: unsigned char]
[+0x2ee] Reserved12       [Type: unsigned char [2]]
[+0x2f0] SharedDataFlags  : 0x10e [Type: unsigned long]
[+0x2f0 ( 0: 0)] DbgErrorPortPresent : 0x0 [Type: unsigned long]
[+0x2f0 ( 1: 1)] DbgElevationEnabled : 0x1 [Type: unsigned long]
[+0x2f0 ( 2: 2)] DbgVirtEnabled   : 0x1 [Type: unsigned long]
[+0x2f0 ( 3: 3)] DbgInstallerDetectEnabled : 0x1 [Type: unsigned long]
[+0x2f0 ( 4: 4)] DbgLkgEnabled    : 0x0 [Type: unsigned long]
[+0x2f0 ( 5: 5)] DbgDynProcessorEnabled : 0x0 [Type: unsigned long]
[+0x2f0 ( 6: 6)] DbgConsoleBrokerEnabled : 0x0 [Type: unsigned long]
[+0x2f0 ( 7: 7)] DbgSecureBootEnabled : 0x0 [Type: unsigned long]
[+0x2f0 ( 8: 8)] DbgMultiSessionSku : 0x1 [Type: unsigned long]
[+0x2f0 ( 9: 9)] DbgMultiUsersInSessionSku : 0x0 [Type: unsigned long]
[+0x2f0 (10:10)] DbgStateSeparationEnabled : 0x0 [Type: unsigned long]
[+0x2f0 (31:11)] SpareBits        : 0x0 [Type: unsigned long]
[+0x2f4] DataFlagsPad     [Type: unsigned long [1]]
[+0x2f8] TestRetInstruction : 0xc3 [Type: unsigned __int64]
[+0x300] QpcFrequency     : 1078030 [Type: __int64]
[+0x308] SystemCall       : 0x0 [Type: unsigned long]
[+0x30c] SystemCallPad0   : 0x0 [Type: unsigned long]
[+0x310] SystemCallPad    [Type: unsigned __int64 [2]]
[+0x320] TickCount        [Type: _KSYSTEM_TIME]
[+0x320] TickCountQuad    : 0x4c399 [Type: unsigned __int64]
[+0x320] ReservedTickCountOverlay [Type: unsigned long [3]]
[+0x32c] TickCountPad     [Type: unsigned long [1]]
[+0x330] Cookie           : 0x2379cafb [Type: unsigned long]
[+0x334] CookiePad        [Type: unsigned long [1]]
[+0x338] ConsoleSessionForegroundProcessId : 2356 [Type: __int64]
[+0x340] TimeUpdateLock   : 0x347d96 [Type: unsigned __int64]
[+0x348] BaselineSystemTimeQpc : 0x1397d5efa [Type: unsigned __int64]
[+0x350] BaselineInterruptTimeQpc : 0x1397d5efa [Type: unsigned __int64]
[+0x358] QpcSystemTimeIncrement : 0x945f138e2b624065 [Type: unsigned __int64]
[+0x360] QpcInterruptTimeIncrement : 0x946b3b6736303611 [Type: unsigned __int64]
[+0x368] QpcSystemTimeIncrementShift : 0x4 [Type: unsigned char]
[+0x369] QpcInterruptTimeIncrementShift : 0x4 [Type: unsigned char]
[+0x36a] UnparkedProcessorCount : 0x1 [Type: unsigned short]
[+0x36c] EnclaveFeatureMask [Type: unsigned long [4]]
[+0x37c] TelemetryCoverageRound : 0x1 [Type: unsigned long]
[+0x380] UserModeGlobalLogger [Type: unsigned short [16]]
[+0x3a0] ImageFileExecutionOptions : 0x0 [Type: unsigned long]
[+0x3a4] LangGenerationCount : 0x1 [Type: unsigned long]
[+0x3a8] Reserved4        : 0x0 [Type: unsigned __int64]
[+0x3b0] InterruptTimeBias : 0x0 [Type: unsigned __int64]
[+0x3b8] QpcBias          : 0xfffffffc4ad2db5b [Type: unsigned __int64]
[+0x3c0] ActiveProcessorCount : 0x1 [Type: unsigned long]
[+0x3c4] ActiveGroupCount : 0x1 [Type: unsigned char]
[+0x3c5] Reserved9        : 0x0 [Type: unsigned char]
[+0x3c6] QpcData          : 0xa81 [Type: unsigned short]
[+0x3c6] QpcBypassEnabled : 0x81 [Type: unsigned char]
[+0x3c7] QpcShift         : 0xa [Type: unsigned char]
[+0x3c8] TimeZoneBiasEffectiveStart : {131668032448612281} [Type: _LARGE_INTEGER]
[+0x3d0] TimeZoneBiasEffectiveEnd : {131851620000000000} [Type: _LARGE_INTEGER]
[+0x3d8] XState           [Type: _XSTATE_CONFIGURATION]
```

- **ID de console active (ActiveConsoleId)** Identifiant de la session de console actuellement en cours d'utilisation par l'utilisateur interactif, utilisé pour savoir quelle console est en avant-plan dans une session de console. Voir fonction RtlGetActiveConsoleId.

- **Nombre de processeurs actifs (ActiveProcessorCount)** Nombre de processeurs disponibles pour l'exécution de threads. Voir routine KeQueryActiveProcessorCount.

- **Architecture dérivée (AlternativeArchitecture)** Sur des systèmes où plusieurs architectures matérielles peuvent coexister, indique si l'architecture du processeur utilisée par le système est une architecture distincte de l'architecture x86 standard. Voir NEC98x86.
- **Configuration COM+ (ComPlusPackage)** Indique si des fonctionnalités ou composants spécifiques COM + sont disponibles sur le système. Voir service NtQuerySystemInformation (SystemInformationClass = SystemComPlusPackage) ; routine ExpReadComPlusPackage.
- **ConsoleSessionForegroundProcessId** Voir fonction RtlGetConsoleSessionForegroundProcessId.
- *Cookie* Utilisé pour l'encodade et le décodage des pointeurs. Voir fonctions DecodeSystemPointer et EncodeSystemPointer ; services RtlDecodeSystemPointer et RtlEncodeSystemPointer.
- **Compteur de démontages (DismountCount)** Nombre de fois que des volumes ont été démontés sur le système depuis son démarrage. Voir DeviceIoControl (dwIoControlCode = FSCTL_DISMOUNT_VOLUME)
- **Options d'exécution (ImageFileExecutionOptions)** Options spécifiques pour l'exécution d'un fichier image (un fichier exécutable) en modifiant son comportement avant ou pendant son lancement : débogage automatique (le programme idoine est démarrée sous le contrôle d'un débogueur), exécution avec des privilèges administratifs, mise en vigueur avec des paramètres de compatibilité tiers, et bien d'autres possibilités.
- *ImageNumberLow* Voir routine ExpInitializeExecutive.
- *ImageNumberHigh* Voir routine ExpInitializeExecutive.
- **Temps d'interruption (InterruptTime)** Temps total (en nombre d'unités de temps de 100 nanosecondes) depuis le démarrage du système, mesuré à partir des interruptions système.
- **Correction du temps d'interruption (InterruptTimeBias)** Ajustement temporel visant à corriger d'éventuelles dérives ou anomalies dans le suivi du temps d'interruption. Voir fonction QueryUnbiasedInterruptTime.
- **Minima de grandes pages (LargePageMinimum)** Taille minimale des pages mémoire de grande taille. Voir fonction GetLargePageMinimum.
- *ProcessorFeatures* Voir routine ExIsProcessorFeaturePresent.
- *Reserved1* Voir routine ExpInitializeExecutive, variable noyau MmHighestUserAddress.
- *Reserved3* Voir routine ExpInitializeExecutive, variable noyau MmSystemRangeStart.
- **Mode sans échec (SafeBootMode)** Indique si le système d'exploitation a ou non démarré en mode sans échec. Voir routine MmInitSystem.
- *TimeZoneBias* Voir routine ExpSetTimeZoneInformation.
- **Numéro de construction (NtBuildNumber)** Numéro de construction de la version de Windows exécutée sur le système, en se basant sur l'état de développement et les mises à jour appliquées. Voir routine PsGetVersion (paramètre BuildNumber).
- **Version majeure de Windows (NtMajorVersion)** Version majeure du système d'exploitation Windows en cours d'exécution. Voir routine ExpInitializeExecutive, variable noyau NtMajorVersion.
- **Version mineure de Windows (NtMinorVersion)** Version mineure du système d'exploitation Windows en cours d'exécution. Voir routine ExpInitializeExecutive, variable noyau NtMinorVersion.
- **Classe de produit (NtProductType)** Indique si le système en cours d'execution se distingue en tant que station de travail, serveur, ou contrôleur de domaine. Voir fonction RtlGetNtProductType.

- **Répertoire racine du système (NtSystemRoot)** Emplacement sur le système de fichiers du répertoire hébergeant les données du système d'exploitation, usuellement C:\WINDOWS. Voir service RtlGetNtSystemRoot.

- **Nombre de pages physiques (NumberOfPhysicalPages)** Nombre total de pages physiques disponibles sur le système. Plus le chiffre mentionné à cet égard est élevé, meilleures sont censées être les performances de l'ordinateur. Voir fonction GlobalMemoryStatusEx ; service NtQuerySystemInformation (SystemInformationClass = SystemBasicInformation).

- *SharedDataFlags* Voir fonction *RtlQueryElevationFlags*.

- **Masque d'édition (SuiteMask)** Masque binaire représentant quel ensemble de fonctionnalités ou quelle édition de Windows est actuellement en cours d'exécution. Voir fonction RtlGetSuiteMask ; structure de données OSVERSIONINFOEX (wSuiteMask).

- **Instruction d'appel système (SystemCall)** Indique si la gestion des appels système s'effectue sur la base de l'instruction sysenter (valeur 0) ou de l'interruption 2E (valeur 1).

- **Temps système (SystemTime)** Date et heure actuelles de la journée, mesurées sous la forme d'un nombre d'unités de temps de 100 nanosecondes écoulées depuis le 1er janvier 1601. Voir fonction GetSystemTimeAsFileTime.

- **Compteur de battements (TickCount)** Nombre de millisecondes qui se sont écoulées depuis le démarrage du système. Voir fonction GetTickCount, RtlGetTickCount.

- **Multiplicateur de battements (TickCountMultiplier)** Voir routine ExpInitializeExecutive, variable noyau ExpTickCountMultiplier.

- *TimeZoneId* Voir fonctions GetTimeZoneInformation et GetTimeZoneInformation. Voir HKLM\SOFTWARE \Microsoft\Windows NT\CurrentVersion\Time Zones.

- *TestRetInstruction* Code opération de l'instruction machine permettant de quitter sans condition une routine (RET).

- *TscQpcEnabled* Un booléen indiquant si le compteur temporel du processeur (instruction rdtsc) peut être utilisé lors de QueryPerformanceCounter et fonctions assimilées.

Pools de mémoire système

Lors de l'initialisation du système, le gestionnaire de mémoire crée deux types de pools (ou réserves) de mémoire, dimensionnés de façon dynamique, que les composants mode noyau utilisent pour allouer de la mémoire système :

- *Pool non paginé* Se compose des plages d'adresses virtuelles système qui sont assurées de résider en mémoire physique et auxquelles il est de ce fait possible d'accéder en permanence (depuis n'importe quel IRQL et n'importe quel contexte de processus). Les composants mode noyau qui s'appuient sur le pool non paginé le font en premier lieu afin de s'affranchir des risques de défauts de page au niveau DPC/dispatch ou supérieur, en l'occurence impossible à satisfaire dans cette configuration. Les données communément conservées dans le pool non paginé incluent l'image du noyau, les représentations en mémoire des différents objets gérés par le système d'exploitation (processus, threads, sémaphores, événements), les IRP en cours, et bien d'autres.

- *Pool paginé* Région de mémoire virtuelle de l'espace système qui peut être déchargée vers le fichier de pagination puis rechargée en mémoire physique. Les composants mode noyau n'ayant pas besoin d'accéder à la mémoire au niveau DPC/dispatch ou supérieur utilisent généralement le pool paginé, lequel est accessible depuis n'importe quel contexte de processus - mais, par contraste avec le pool non paginé, pas depuis n'importe quel IRQL.

Ces deux pools de mémoire se trouvent se trouvent dans la partie système de l'espace d'adressage et sont mappés dans l'espace d'adressage de chaque processus.

POOL_HEADER.

```
lkd> dt nt!_POOL_HEADER
   +0x000 PreviousSize     : Pos 0, 8 Bits
   +0x000 PoolIndex        : Pos 8, 8 Bits
   +0x002 BlockSize        : Pos 0, 8 Bits
   +0x002 PoolType         : Pos 8, 8 Bits
   +0x000 Ulong1           : Uint4B
   +0x004 PoolTag          : Uint4B
   +0x008 ProcessBilled    : Ptr64 _EPROCESS
   +0x008 AllocatorBackTraceIndex : Uint2B
   +0x00a PoolTagHash      : Uint2B
```

- *Balise de pool (PoolTag)* Séquence de quatre octets donnée à l'allocation.
- *Type de pool (PoolType)* Spécifie laquelle des régions de mémoire virtuelle de l'espace système (pool paginé, pool non paginé, etc.) a honoré la requête d'allocation.
- *Processus débiteur (ProcessBilled)* Pointeur vers le bloc EPROCESS du processus auquel les frais relatifs à l'allocation de mémoire ont été imputés.

POOL_HEADER PoolType.

```
lkd> dt nt!_POOL_TYPE
   NonPagedPool = 0n0
   NonPagedPoolExecute = 0n0
   PagedPool = 0n1
   NonPagedPoolMustSucceed = 0n2
   DontUseThisType = 0n3
   NonPagedPoolCacheAligned = 0n4
   PagedPoolCacheAligned = 0n5
   NonPagedPoolCacheAlignedMustS = 0n6
   MaxPoolType = 0n7
   NonPagedPoolBase = 0n0
   NonPagedPoolBaseMustSucceed = 0n2
   NonPagedPoolBaseCacheAligned = 0n4
   NonPagedPoolBaseCacheAlignedMustS = 0n6
   NonPagedPoolSession = 0n32
   PagedPoolSession = 0n33
   NonPagedPoolMustSucceedSession = 0n34
   DontUseThisTypeSession = 0n35
   NonPagedPoolCacheAlignedSession = 0n36
   PagedPoolCacheAlignedSession = 0n37
   NonPagedPoolCacheAlignedMustSSession = 0n38
   NonPagedPoolNx = 0n512
   NonPagedPoolNxCacheAligned = 0n516
   NonPagedPoolSessionNx = 0n544
```

Pool tagging

Le balisage de pool (*pool tagging*) se rapporte à l'intégration parmi les stratégies régissant la mémoire système de différents éléments d'information et d'orientation, capables alors de retranscrire fidèlement les échanges qui y ont lieu, et plus important encore, de mesurer ponctuellement ou durablement la consommation qu'en fait tel ou tel composant noyau. Au-delà de ces aspects, un tel mécanisme se révèle aussi particulièrement utile lors du diagnostic (a priori la résolution) de problèmes de fuites de mémoire.

Chaque balise de pool se présente en interne sous la forme d'une séquence alphanumérique de quatre octets et fait le lien avec un bloc de mémoire alloué dans la partie système de l'espace d'adressage. L'approche usuelle

en matière d'utilisation de balises de pool consiste à créer une balise unique pour chaque classe de données qui fait partie d'un ensemble et peut être considéré isolement. Pour une description des balises de pool employées par Windows, consultez le fichier \Program Files\Debugging Tools for Windows\Triage\Pooltag.txt.

Au volet opérationnel, les routines liées au pool tagging se superposent à celles déjà en place pour l'allocation/désallocation à partir des pools de mémoire système. Par exemple, ExAllocatePoolWithTag est un sur-ensemble de ExAllocatePool, ExFreePoolWithTag un sur-ensemble de ExFreePool, et ainsi de suite.

Taille des pools

Plusieurs des compteurs qu'offre l'objet de performances Mémoire offrent une vue sur les tailles (tant virtuelles que physiques) du pool paginé et du pool non paginé, à savoir :

- Octets de réserve non paginée
- Octets de réserve paginée
- Octets résidants de réserve paginée

Tableau 10.6. Taille des pools

Variable / Type	**Description**
MmSizeOfNonPagedPoolInBytes	Taille courante du pool non paginé.
MmMaximumNonPagedPoolInBytes	Taille maximale du pool pool non paginé.
MmSizeOfPagedPoolInBytes	Taille (virtuelle) maximale du pool paginé.

Élément / Type	**Lieu**
MaximumNonPagedPoolInPages	ULONG
MI_VISIBLE_STATE	SizeOfPagedPoolInPages
ULONG	MI_VISIBLE_STATE

Surveillance de la consommation des pools

L'utilitaire Poolmon permet de surveiller la consommation des pools noyaux paginés et non paginés du système.

Tableau 10.7. Colones de Poolmon

Colonne	Explication
Tag	Marquage (séquence de quatre octets) donnée à l'allocation
Type	Type de pool (paginé ou non paginé)
Allocs	Nombre d'allocations
Frees	Nombre de libres
Diff	Nombre d'allocations moins nombre de libres
Bytes	Total des octets consommés par ce type de structure
Per Alloc	Taille en octets d'une instance de ce type de structure

Autre façon de voir la consommation que font le noyau, les pilotes et autres codes système des pools : activer la fonctionnalité de suivi de pool (*pool tracking*) du Vérificateur de pilote.

Enfin, il est possible de voir la consommation de pool à l'aide de la commande !poolused du débogueur noyau. L'exemple que voici montre un résultat partiel de la commande exécutée sans paramètres :

```
lkd> !poolused
..
 Sorting by Tag
```

```
               NonPaged                  Paged
 Tag     Allocs         Used     Allocs         Used

 -UMD         1           48          0            0 UNKNOWN pooltag '-UMD', please update pooltag.txt
 .UMD         1          128          0            0 UNKNOWN pooltag '.UMD', please update pooltag.txt
 2UuQ         4        16384          0            0 UNKNOWN pooltag '2UuQ', please update pooltag.txt
 8042         4         4064          0            0 PS/2 keyboard and mouse , Binary: i8042prt.sys
 ALPC      2148      1280000          0            0 ALPC port objects , Binary: nt!alpc
 APpt      1501       192336       3531       345904 UNKNOWN pooltag 'APpt', please update pooltag.txt
 ARFT         0            0          2          128 UNKNOWN pooltag 'ARFT', please update pooltag.txt
 AVNw         1          976          0            0 UNKNOWN pooltag 'AVNw', please update pooltag.txt
 AVns         1           48          0            0 UNKNOWN pooltag 'AVns', please update pooltag.txt
 AVpc         1          976          0            0 UNKNOWN pooltag 'AVpc', please update pooltag.txt
 AVpd         1          976          0            0 UNKNOWN pooltag 'AVpd', please update pooltag.txt
 AVwi         1          352          0            0 UNKNOWN pooltag 'AVwi', please update pooltag.txt
 AcpA         2          160          0            0 ACPI arbiter data , Binary: acpi.sys
 AcpD       158       150192          0            0 ACPI device data , Binary: acpi.sys
 AcpE         1          928          0            0 ACPI embedded controller data , Binary: acpi.sys
 AcpF        12          960          0            0 ACPI interface data , Binary: acpi.sys
 AcpI         0            0         50       116672 ACPI irp data , Binary: acpi.sys
 AcpM         6          272          1          144 ACPI miscellaneous data , Binary: acpi.sys
 AcpO         7          480          0            0 ACPI object data , Binary: acpi.sys
 AcpP         4         1152          0            0 ACPI power data , Binary: acpi.sys
 .
 .
 .
 wdnd         1           48          0            0 UNKNOWN pooltag 'wdnd', please update pooltag.txt
 wdnf        90         7616          0            0 UNKNOWN pooltag 'wdnf', please update pooltag.txt

TOTAL    229512    110273472     434124    210144528
```

La commande !poolused 2 rend compte de la consommation de pool non paginé, triée par balise utilisant le plus de pool.

```
lkd> !poolused 2
..
 Sorting by NonPaged Pool Consumed

               NonPaged                  Paged
 Tag     Allocs         Used     Allocs         Used

 ConT       284     10002432          0            0 UNKNOWN pooltag 'ConT', please update pooltag.txt
 File     20985      8379120          0            0 File objects
 Ntfx     20154      7256160          0            0 Unrecognized NTFS tag (update base\published\pooltag.w) , Binary:
ntfs.sys
 Vi24        38      6709248          0            0 Video memory manager DMA buffer allocation list , Binary: dxgmms2.sys
 MmCa     18336      6158704          0            0 Mm control areas for mapped files , Binary: nt!mm
 EtwB       160      4968704         14       409600 Etw Buffer , Binary: nt!etw
 .
 .
 .
```

La commande !poolused 4 affiche la consommation de pool paginé, encore une fois triée par balise.

```
lkd> !poolused 4
..
 Sorting by Paged Pool Consumed

               NonPaged                  Paged
 Tag     Allocs         Used     Allocs         Used

 MmSt         0            0      35498     78274832 Mm section object prototype ptes , Binary: nt!mm
 FMfn         4         1216     112447     70575328 NAME_CACHE_NODE structure , Binary: fltmgr.sys
 Ntff        19         6992      33015     46485120 FCB_DATA , Binary: ntfs.sys
```

```
NtfF        0           0      16764     26822400 FCB_INDEX , Binary: ntfs.sys
NtFs        9      218624     113705     21138864 StrucSup.c , Binary: ntfs.sys
NtFU        0           0      43886     15260544 usnsup.c , Binary: ntfs.sys
```

Granularité d'allocation

Windows aligne chaque région d'espace d'adressage réservé de processus de façon qu'elle commence sur une frontière entière ; comportement auquel se rapporte en interne la *granularité d'allocation* du système, dont la valeur se monte à l'heure actuelle à 64 ko.

Quand est commandité une opération l'amenant à réserver une région d'espace d'adressage, Windows fait en sorte que la taille et la base de ladite région soit un multiple de la taille de la page système. Par exemple, sur les systèmes x86, qui emploient des pages de 4 Ko, le volume véritablement réservé à la suite d'une demande de 5 Ko est en réalité de 8 Ko (soit 2 pages). Sur le même ordre d'idée, dans l'éventualité où une application réclame une région de mémoire avec une adresse de base de 3 Ko et une taille de 10 Ko, le volume qui en découle est en définitive 16 Ko.

Grandes et petites pages

L'espace d'adressage virtuel est divisé en unité appelées *pages*, lesquelles constituent individuellement le plus petit dénominateur commun auprès de l'unité matérielle de gestion mémoire. Il existe deux tailles de page : petite et grande. Les tailles réelles, qui varient en fonction de l'architecture matérielle, sont données au tableau suivant.

Tableau 10.8. Tailles de page

Architecture	Taille de petite page	Taille de grande page
x86	4 Ko	4 Mo (2 Mo sur systèmes PAE)
X64	4 Ko	2 Mo

Les systèmes x86 utilisent des pages de 4 Ko, taille dont le choix résulte d'un compromis entre la complexité du mécanisme de parcours des structures de tables de pages et l'utilisation effective de la mémoire. En effet, plus la taille des pages est petite, plus il facile d'approximer fidèlement la quantité de mémoire réellement consommée par une application.

L'avantage des grandes pages par rapport aux petites est avant tout de donner lieu à moins d'informations pour gérer la même quantité de mémoire, ce qui permet en définitive de diminuer les coûts qu'implique le processus de traduction des adresses virtuelles en adresses physiques.

Sur les systèmes pourvues de suffisamment de mémoire pour adopter une telle approche, Windows mappe à l'aide de grandes pages les images fondamentales du système d'exploitation (Ntoskrnl.exe, Hal.dll, etc.), ainsi que les données où il fait sens d'employer cette fonction, par exemple les tables de pages.

Fonctions	**Services**
VirtualAlloc (flAllocationType = MEM_LARGE_PAGES)	NtAllocateVirtualMemory (AllocationType = MEM_LARGE_PAGES)

Élément	**Lieu**	**Autres références**
LargePageMinimum	KUSER_SHARED_DATA	GetLargePageMinimum
ImageUsesLargePages	PEB	EPROCESS Flags (PROCESS_CREATE_-FLAGS_LARGE_PAGES)
dwPageSize	SYSTEM_INFO	

Constante	Valeur
MEM_LARGE_PAGES	0x20000000
PROCESS_CREATE_FLAGS_LARGE_PAGES	0x00000010

Géographie de l'espace d'adressage virtuel

Cette section va décrire les composants de l'espace d'adressage utilisateur et de l'espace d'adressage système, avant de détailler de détailler les géographies spécifiques des systèmes 32 bits et 64 bits.

Il existe trois grands types de données à être mappées dans l'espace d'adressage virtuel sous Windows :

- Code et données privatifs de chaque processus
- Code et données de niveau session
- Code et données de niveau système

Chaque processus sous Windows a un espace d'adressage privé qui est inaccessible aux autres processus (sauf s'ils en ont la permission en bonne et due forme). Les threads du processus ne peuvent jamais accéder aux adresses virtuelles situées hors de cette zone, sauf s'ils s'interfacent à des sections de mémoire partagée et/ou emploient les fonctions de mémoire inter processus qui permettent d'accéder à l'espace d'adressage d'un autre processus (pourvu encore une fois que l'appelant en ai le droit). Les données décrivant l'espace d'adressage virtuel du processus, appelées tables de pages, sont marquées comme étant accessibles uniquement depuis le mode noyau, de sorte que les threads utilisateur ne puissent établir de relations fonctionnelles avec elles et par, exemple, modifier la géographie de leur propre espace d'adressage ou rendre visible des données qui ne devraient pas l'être.

L'espace de session contient les informations globales relatives à la prise en charge des sessions multiples. Chaque session a son propre pool paginé, utilisé par la partie mode noyau du sous-système Windows (Win32k.sys) pour allouer des structures de données d'interface utilisateur propres à la session. En outre, chaque session dispose de son propre exemplaire du processus du sous-système Windows (Csrss.exe) et du processus d'ouverture de session (Winlogon.exe). L'espace système se compose des éléments que voici :

- *Code système* Contient les images appropriées du noyau et de la couche couche d'abstraction matérielle (Ntoskrnl.exe et Hal.dll par défaut), ainsi que les pilotes utilisés pour amorcer le système.
- *Vues mappées du système* Contient la partie mode noyau du sous-système Windows (Win32k.sys), ainsi que les pilotes graphiques mode noyau qu'elle emploie.
- *Liste du working set système* Structures de données décrivant le working set système.
- *Cache système*
- *Pool paginé* Région de mémoire virtuelle dont les pages physiques qui la constituent peuvent être remisées vers le fichier de pagination puis rechargées lorsque les circonstances le demandent.
- *PTE système*
- *Pool non paginé* Région de mémoire composée de plages d'adresses virtuelles qui résident en permanence en mémoire physique et et auxquelles il est de ce fait possible d'accéder en permanence depuis n'importe quel espace d'adressage.
- *Données de dump d'effondrement* Zone réservée pour l'enregistrement d'informations sur le système suite à un effondrement de ce dernier.
- *Zone réservée HAL* Mémoire système réservée à des structures propres à HAL.

Géographie de l'espace d'adressage utilisateur x86

Par défaut, les versions de Windows calibrées pour l'architecture x86 32 bits offrent à chaque processus 2 Go d'espace d'adressage privé ; le système d'exploitation se réservant les 2 Go restants. Différentes options de

démarrage permettent néanmoins de tempérer ces restrictions, et de la sorte attribuer plus de mémoire aux applications (3 Go) et moins aux composants du noyau (1 Go).

Pour qu'un processus puisse croître au-dela de 2 Go d'espace d'adressage, le ichier image sous-jacent doit impérativement avoir le flag IMAGE_FILE_LARGE_ADDRESS_AWARE positionné dans son entête. Autrement, Windows réserve l'espace d'adressage supplémentaire de ce processus de façon que l'application n'ai pas conscience de ladite zone. Pour mettre en avant ce flag, spécifiez lors de la construction de l'exécutable l'option d'édition de liens /LARGEADDRESSAWARE.

Il existe plusieurs images système marquées comme compatibles avec les grandes adresses, leur permettant ainsi en définitive de tirer profit de plus larges plages d'adresses virtuelles. En voici quelques-unes :

- Chkdsk.exe Utilitaire Check Disk
- Lsass.exe Sous-système d'autorisation de sécurité locale
- Inetinfo.exe Internet Information Server
- Smss.exe Gestionnaire de sessions

Etant donné que les allocations mémoire (émanant de fonctions telles VirtualAlloc et VirtualAlloc) commencent par défaut aux adresses virtuelles basses, à moins qu'un processus n'alloue une importante quantité de mémoire, ou n'ait un espace d'adressage très fragmenté, il est fortement improbable que ledit processus soit amené à manipuler des adresses significativement hautes. Vous avez la possibilité d'agir sur ces aspects, et en l'occurence forcer les allocations mémoire à partir des adresses hautes en spécifiant le flag MEM_TOP_DOWN lors de l'appel à une fonction d'allocation, ou en ajoutant une valeur de registre, HKLM\SYSTEM \CurrentControlSet\Control\Session Manager\Memory Management\AllocationPreference, et lui conférant la valeur 0x100000.

Géographie de l'espace d'adressage système x86

Cette section va détailler la géographie et le contenu de l'espace système dans la perceptive des systèmes Windows 32 bits, qui peuvent selon les options de configuration employées disposer de 2 ou de 3 Go d'espace système.

Le tableau suivant énumère les variables noyau qui font le lien avec les adresses de début et de fin de diverses régions de l'espace système. Certaines de ces régions ont des frontières figées, et sont de la sorte immuables entre toutes les versions de Windows ; d'autres sont dimensionnées en fonction de la taille de la mémoire et de l'édition (client ou serveur) de Windows exécutée.

Tableau 10.9. Variables système décrivant les régions de l'espace système

Variable	Description	x86/2 Go	x86/1 Go
MmNonPagedPoolEnd	Fin pool non paginé	0xFFBE0000	0xFFBE0000
MmNonPagedPoolStart	Début pool non paginé	Calculé	Calculé
MmPagedPoolEnd	Fin pool paginé	Calculé	Calculé
MmPagedPoolStart	Début pool paginé	Calculé	Calculé
MmSystemCacheEnd	Fin cache système	Calculé	Calculé
MmSystemCacheStart	Début cache système	Calculé	Calculé
MmSystemRangeStart	Début espace système	0x80000000	0xC0000000

Géographie de l'espace d'adressage virtuel utilisateur

De façon similaire à ce qui se produit dans la sphère noyau, l'espace d'adressage destiné aux applications en mode utilisateur est géré de manière dynamique. Les adresses associées aux éléments tels que les piles de

threads, les tas de processus et les images chargées sont de la sorte sujettes à diverses variations, atténuant ainsi leur prévisibilité dans les circonstances où la sécurité du système est mise en jeu.

Concrètement, l'espace d'adressage utilisateur est structuré en plusieurs régions distinctes. L'exécutable enveloppant le programme et les bibliothèques dynamiques (DLL) sur lesquelles il s'appuie y sont en premier lieu matérialisés sous la forme de fichiers mappés en mémoire. Ces éléments sont ensuite suivis par le ou les tas du processus et par la ou les piles de ses threads respectifs.

Visualisation de l'espace d'adressage virtuel utilisateur

Entre autres utilitaires via lesquels examiner l'espace d'adressage, VMMap fournit une analyse détaillée de l'utilisation de la mémoire par un processus, affichant des représentations visuelles et des informations sur les ressources consommées.

Au démarrage, VMMap affiche une liste des processus s'exécutant pour le compte de l'utilisateur en cours. Le bouton *Show All Processes*, au bas de la fenêtre, ouvre la possibilité d'interagir avec un plus grand nombre de processus (moyennant des droits administrateur). Notez que même ainsi, l'accès reste limité aux processus mode utilisateur et non protégés. Une fois un processus sélectionné, la vue principale de l'application est alors actualisée conformément aux allocations de mémoire ayant eu lieu dans ce contexte.

La partie supérieure de la fenêtre intègre trois compteurs :

- **Mémoire engagée (*Committed memory*)** Quantité totale de mémoire virtuelle revenant au processus, incluant à la fois des pages de mémoire privée (attribuées exclusivement au processus) et des pages de mémoire partagée (partagées entre plusieurs processus).
- **Octets privés (*Private Bytes*)** Quantité de mémoire allouée exclusivement au processus (qui ne peut autrement dit pas être partagée avec d'autres). Cela comprend généralement les données et les ressources utilisées uniquement par ce processus.
- **Jeu de travail (*Working Set*)** Quantité totale de mémoire physique (RAM) utilisée par le processus. Cela inclut à la fois les pages de mémoire privée et partagée qui sont actuellement chargées en mémoire vive.

Les informations présentées par la suite (volet central) sont réparties en catégories pour chaque type d'allocation.

Type	Description
Image	Fichiers mappés en mémoire associées causalement à un programme (EXE) et à ses dépendances (DLL)
Mapped File	Fichiers mappés qui ne sont pas des images, et hébergent en l'occurence des données
Shareable	Fichiers mappés sauvegardés dans un fichier d'échange
Heap	Tas du processus
Managed Heap	Mémoire gérée par le runtime .NET
Stack	Piles de threads
Private Data	Pages privées (allouées via les fonctions Windows de la famille VirtualAlloc)
Free	Pages libres

Géographie de l'espace d'adressage 64 bits

L'espace d'adressage virtuel 64 bits fait théoriquement 16 exaoctets, soit environ 17 milliards de Go. Les limites d'implémentation actuelles des processeurs (voir plus loin) plafonnent concrètement ce chiffre à 256 téraoctets. La plage d'adresses qui en découle est divisée en deux parties égales (128 To), moitié pour un processus, moitié pour le système.

Un fait intéressant à relever est que, dans la pratique, les applications 32 bits compatibles avec un espace d'adressage étendu (option `largeaddressaware`) tirent un avantage supplémentaire d'une exécution sous un

système 64 bits (Wow64), se voyant auquel cas offert la possibilité d'utiliser réellement 4 Go d'espace d'adressage utilisateur (contre 3 dans la limite traditionnelle x86). Voici, à titre d'exemple, quels résultats produit l'utilitaire TestLimit (version x86) sur une machine dotée d'un système d'exploitation Windows 64 bits.

```
C:\>Testlimit.exe -r

Testlimit v5.24 - test Windows limits
Copyright (C) 2012-2015 Mark Russinovich
Sysinternals - www.sysinternals.com

Process ID: 6112

Reserving private bytes (MB)...
Leaked 4029 MB of reserved memory (4029 MB total leaked). Lasterror: 8
```

Les processeurs installés dans le paysage informatique contemporain exposent pour des raisons pratiques seulement 48 lignes d'adresse. De fait, pour une adresse 64 bits donnée, seuls sont réellement pris en compte les 48 premiers. Les 16 autres (bits 48 à 63) doivent être définis de la même manière (autrement dit avoir la même valeur) que le bit le plus significatif (bit 47). Une adresse conforme à ces exigences est dite *canonique*. Dans une telle configuration, la moitié inférieure de l'espace d'adressage commence à 0x0000000000000000 et s'étend jusque l'adresse 0x00007FFFFFFFFFFF, tandis que la moitié supérieure occupe la plage 0xFFFF800000000000 - 0xFFFFFFFFFFFFFFFF.

Traduction d'adresse

Windows reconnait trois sortes d'adresses :

- Les adresses *virtuelles*, traduites en adresses physiques avant utilisation réelle.
- Les adresses *physiques*, employées par les bus et la mémoire.
- Les adresses logiques, à l'aide desquelles HAL assure la communication avec certain pilote de périphérique.

Il est essentiel dans chaque cas d'utilisation de bien comprendre quel type d'adresse et quelles modalités sont en vigueur. Ne pas le faire pourrait entrainer des modifications non désirées des données, menant à terme à l'instabilité, au pire des cas à l'effondrement, du système.

Les sections suivantes vont décrire les mécanismes sur lesquels Windows s'appuie pour associer des adresses virtuelles à des adresses physiques.

Traduction d'adresse virtuelle x86

Par défaut, sur les systèmes x86, Windows emploie pour traduire les adresses virtuelles en adresses physiques une structure de tables de pages à deux niveaux. (Les systèmes x86 exécutant le noyau PAE utilisent un niveau supplémentaire.) Chaque adresse virtuelle se présente dans cette configuration sous forme de trois composants distincts : l'index de répertoire de pages, l'index de table de pages et l'index d'octet.

Répertoire de pages

La structure par laquelle le gestionnaire de mémoire enregistre l'emplacement de toutes les tables de pages qu'emploie un processus est dite répertoire de pages. Chaque processus a un et un seul répertoire de pages, concrétisé en interne par un attribut du bloc KPROCESS (DirectoryTableBase), qui contient en l'occurence l'adresse physique du répertoire de pages. (A noter que sur les systèmes x86, l'adresse physique du répertoire de pages est aussi immanquablement mappée à l'adresse virtuelle 0xC0300000.)

Le matériel connait l'emplacement du répertoire de pages du processus courant grâce à un registre spécial du processeur (CR3 sur les systèmes x86), dont le système d'exploitation régit la valeur en fonction des activités

d'ordonnancement. Chaque fois qu'il y a un basculement de contexte en faveur d'un thread appartenant à un autre processus que celui du thread en cours, ce registre est actualisée de sorte à refléter l'adresse physique du répertoire de pages du processus donnant un contexte au thread nouvellement élue. Les basculements de contexte entre threads d'un même processus n'entrainent pas un tel changement, cela dans la mesure où tous les threads d'un processus se partagent le même espace d'adressage (et par extension, le même répertoire de pages).

Le répertoire de pages se compose d'entrées de répertoire de pages (PDE), chacune d'entre elles correspondant à une table de pages. Chaque entrée parmi cette table de pages donne l'adresse de la page physique correspondante.

Une partie de l'espace d'adressage étant alloué en privé à chaque processus, chacun dispose par conséquent de propre jeu de tables de pages correspondant au code et aux données qui lui sont privatifs. Les tables de pages crées dans ce contexte le sont à la demande, de sorte que le répertoire de pages de la plupart des processus ne renferme qu'un petit nombre de références. En revanche, les tables de pages décrivant l'espace système sont partagées entre tous les processus. De ce fait, quand un processus est créé, les entrées de répertoire de pages décrivant l'espace système sont initialisées de façon à pointer vers les tables de pages existantes du système.

Traitement des défauts de page

Le tableau suivant examine les différentes circonstances dont il peut émaner un défaut de page.

Tableau 10.10. Circonstances apparentées aux erreurs d'accès

Circonstance	Résultat
Accès à une page qui n'est pas en mémoire, mais se trouve sur disque dans un fichier de pagination ou un fichier mappé	Alloue une page puis charge la page désirée depuis le disque vers le working set du processus
Accès à une page qui n'est pas engagée	Violation d'accès
Accès depuis le mode utilisateur à une page accessible uniquement en mode noyau	Violation d'accès
Écriture dans une page marquée lecture seule	Violation d'accès
Accès à une page qui doit être satisfaite à l'aide d'une page initialisée à zéro	Ajoute une page remplie de zéros au working set du processus
Écriture dans une page de garde	Violation de page de garde
Écriture dans une page marquée copie sur écriture (copy-on-write)	Donne au processus une copie privative de la page
Tentative d'exécuter du code que renferme une page marquée pas d'exécution	Violation d'accès

Tables de pages et PTE

Tableau 10.11. Signification d'un PTE

Position du bit	Nom	Signification
0	Valid	Indique si oui ou non la page se trouve en mémoire physique. Toute référence à une page qui ne réside pas en mémoire physique génère un défaut de page.
1	Write	
2	Owner	Indique si le code mode utilisateur a la possibité d'accéder à la page ou si cette dernière n'est accessible qu'au code mode noyau
3	Write through	Désactive la mise en cache des écritures sur cette page, de sorte que les modifications la concernant sont enregistrées sur disque immmédiatement après qu'elles aient eu lieu.
4	Cache disabled	Désactive la mise en cache pour cette page.
5	Accessed	La page a été lue.
6	Dirty	La page a été modifiée.
7	Reserved	

Position du bit	Nom	Signification
8	Global	La traduction s'applique à tous les processus
9	Copy-on-write	La page utilise la copie sur écriture (décrite précédemment).
10	Prototype	
11	Write	
12-35	PFN	Numéro PFN (Page Frame Number) de la page physique
63	No eXecute	

Ainsi que vous le verrez ultérieurement, les bits repérables par un astérisque dans le tableau qui précède sont interprétées uniquement au niveau logiciel. Ce n'est dès lors pas le processeur qui les manipule, mais le gestionnaire de mémoire intégrée à Windows.

Base de données PFN

Le gestionnaire de mémoire enregistre la trace de chaque page de mémoire physique dans une table appelée la base de données PFN. La liste qui suit énumère les états de page :

- *Active (ou Valide)* La page fait partie d'un working set (de processus ou du système) ou n'appartient à aucun working set (par exemple, page noyau non paginée), et un PTE valide pointe vers elle.
- *En transition* État temporaire d'une page qui n'est propriété d'aucun working set et qui ne figure sur aucune liste de pagination. Une page se trouve dans cette configuration quand une opération d'E/S en cours fait référence à ladite page.
- *En attente* La page appartenait précédemment à un working set, mais a été supprimée. La page n'a pas été modifiée depuis sa dernière écriture sur le disque. LE PTE référence toujours la page physique, mais est marqué comme invalide et en transition.
- *Modifiée* La page appartenait précédemment à un working set, mais a été supprimée. Cependant, la page a été modifiée et son contenu actuel n'a pas encore été écrit sur le disque. LE PTE référence toujours la page physique, mais est marqué comme invalide et en transition. Il doit être écrit sur le disque avant que la page physique puisse être réutilisée.
- *Libre* La page est libre mais contient des données indéfinies. Du fait des exigences de sécurité auxquelles répond le gestionnaire de mémoire, ces pages ne peuvent être confiées en l'état à des processus mode utilisateur ; elles pourront néanmoins l'être après avoir fait l'objet des ajustements nécessaires par le thread de page à zéro.
- *Mise à zéro* La page est libre et a été remplie de zéros par le thread de page à zéro.
- *Erronée* La page a générée des erreurs de parité mémoire, et a par conséquent été reconnue comme défectueuse.

Les pages se rapportant à quelques-uns des états susmentionnés sont organisées en listes chainées, par commodité d'abord, mais essentiellement du fait des gains en performance qu'une telle manière de procéder implique (le gestionnaire de mémoire peut ainsi trouver rapidement les pages d'un certain type.

Tableau 10.12. Base de données PFN

Variable	Type
MmPfnDatabase	MMPFN
PageLocationList	PMMPFNLIST [8]

Élément	Type	Lieu
BadPageListHead	MMPFNLIST	MI_PARTITION_PAGE_LISTS

Élément	Type	Lieu
FreePageListHead	MMPFNLIST	MI_PARTITION_PAGE_LISTS
ModifiedNoWritePageListHead	MMPFNLIST	MI_PARTITION_PAGE_LISTS
ModifiedPageListHead#	MMPFNLIST	MI_PARTITION_PAGE_LISTS
StandbyPageListHead	MMPFNLIST	MI_PARTITION_PAGE_LISTS
ZeroedPageListHead	MMPFNLIST	MI_PARTITION_PAGE_LISTS

Gestion de la mémoire physique. [cols="a", grid="none", frame="none"]

Élément	Lieu	Description
AvailablePages	MI_VISIBLE_PARTITION	Nombre total de pages disponibles (total des listes Libres, Mises à zéro et En attente)
HighestPhysicalPage	MI_VISIBLE_PARTITION	
HighestPhysicalPageNumber	SYSTEM_BASIC_INFORMATION	Voir routine ExpGetSystemBasicInformation.
LowestPhysicalPage	MI_VISIBLE_PARTITION	
LowMemoryThreshold	MI_PARTITION_PAGE_LISTS	Seuil de mémoire faible
NumberOfPhysicalPages	MI_VISIBLE_PARTITION	Nombre total de pages physiques disponibles. Voir routine MmGetLowestPhysicalPage.
NumberOfPhysicalPages	KUSER_SHARED_DATA	
NumberOfPhysicalPages	SYSTEM_BASIC_INFORMATION	Voir routine ExpGetSystemBasicInformation.

Dynamique des listes de pages

Lorsque les circonstances exigent une page initialisée à zéro, le gestionnaire de mémoire essaie d'en obtenir une en premier lieu depuis la liste Mises à zéro. Si ladite liste est vide, il tente alors d'en acquérir une depuis la liste Libres et, dans l'éventualité où l'opération à eu lieu, remplit de zéros la page. Autrement (c'est à dire dans l'optique où la liste Libres est vide) le gestionnaire de mémoire sélectionne une page de la liste En attente et la remplit de zéros.

Quand le gestionnaire de mémoire n'a pas besoin de page mise à zéro, ses préférences vont, dans l'ordre, aux listes Libres, Mises à zéro et En attente.

Quand un processus se voit dans l'obligation de céder une page de son working set (soit parce qu'une opération qui en émane a référencé une nouvelle page et que son working set a atteint le seuil maximum autorisé, soit sous l'effet d'un élagage de son working set par le gestionnaire de mémoire), la page est remisée dans la liste En attente si elle est propre (non modifiée) et, dans la liste Modidiées si elle ne l'est pas. Quand un processus prend fin, toutes les pages privées s'y rattachant vont dans la liste Libres.

Priorité de page

Afin d'orienter de manière optimale les décisions stratégiques en matière de remplacement de pages, Windows attribue à chacune une valeur de priorité (comprise entre 0 et 7) : plus celle-ci est élevée, plus la page est protégée contre l'éviction (par extension, moins elle risque d'être reléguée sur disque). En interne, un tel mécanisme s'appuie sur la liste des pages en attente (stand-by), elle-même divisée en huit sous-listes, conformément aux différents niveaux de priorité. Lorsqu'un besoin de mémoire impose la réutilisation de pages, le système parcourt ces sous-listes dans l'ordre croissant de priorité, favorisant ainsi l'éviction des pages les moins critiques.

Chaque processus, ainsi que chaque thread, se voit également attribuer un rang de priorité par le gestionnaire de mémoire, rang dont dépend en pratique la priorité des pages allouées. En règle générale, la priorité d'une page reflète celle du thread à l'origine de son allocation. Par défaut, un thread hérite de la priorité de page du processus

auquel il appartient. À l'état initial, processus et threads ont une priorité de page de 5, mais peuvent ajuster cette valeur par le biais, respectivement, des fonctions `SetProcessInformation` et `SetThreadInformation`.

La priorité dont rend compte une page reflète généralement la priorité de page du thread lui ayant donné lieu - et dans l'éventualité où la page est partagée, la priorité de page la plus élevée parmi les différents threads qui concourent à ce partage. Un thread hérite de la valeur de priorité de la page du processus auquel il appartient.

Pour voir les listes de pages priorisées, employez l'utilitaire MemInfo en spécifiant l'option -c.

Visualisation de la base de données PFN

La commande !memusage du débogueur noyau permet d'afficher les tailles des diverses listes de pagination. La sortie de cette commande ressemble à ce qui suit :

```
lkd> !memusage
 loading PFN database
loading (100% complete)
Compiling memory usage data (99% Complete).
             Zeroed:      620 (    2480 kb)
               Free:        2 (       8 kb)
            Standby:   335569 ( 1342276 kb)
           Modified:    32251 (  129004 kb)
    ModifiedNoWrite:       29 (     116 kb)
       Active/Valid:   407001 ( 1628004 kb)
         Transition:      119 (     476 kb)
         SLIST/Temp:     5038 (   20152 kb)
                Bad:        0 (       0 kb)
            Unknown:        0 (       0 kb)
              TOTAL:   780629 ( 3122516 kb)

lkd> !pfn 0 1

 Page    Flink  Blk/Shr Ref V    PTE   Address  SavedPTE Frame  State
    1        0        1     1 fffff5fbf0400000 fffff7e080000000 200000000080   3804 Active   M
    2        0        1     1 fffff5fae5bf8818 fffff5cb7f103000 200000000080   3829 Bad
    3        0        1     1 fffff5fae5bf8820 fffff5cb7f104000 200000000080   3829 Bad
    4        0        1     1 fffff5fbf0400010 fffff7e080002000 200000000080   3804 Active   M
    5        0        1     1 fffff5fbf0400018 fffff7e080003000 200000000080   3804 Active   M
    6        0        1     1 fffff5fbf0400008 fffff7e080001000 200000000080   3804 Active   M
    7        0        1     1 fffff5fbf0400020 fffff7e080004000 200000000080   3804 Active   M
    8        0        1     1 fffff5fbf0400028 fffff7e080005000 200000000080   3804 Active   M
    9        0        1     1 fffff5fbf0400030 fffff7e080006000 200000000080   3804 Active   M
    a        0        1     1 fffff5fbf0400038 fffff7e080007000 200000000080   3804 Active   M
    b        0        1     1 fffff5fbf0400040 fffff7e080008000 200000000080   3804 Active   M
    c        0        1     1 fffff5fbf0400048 fffff7e080009000 200000000080   3804 Active   M
    d        1       50     1 ffffc4840fc02378        0 200000000080   3804 Active   MP
    e        0        1     1 fffff5fbf0400058 fffff7e08000b000 200000000080   3804 Active   M
    f        0        1     1 fffff5fbf0400068 fffff7e08000d000 200000000080   3804 Active   M
   10        0        1     1 fffff5fbf0400070 fffff7e08000e000 200000000080   3804 Active   M
   11        0        1     1 fffff5fbf0400060 fffff7e08000c000 200000000080   3804 Active   M
   12        0        1     1 fffff5fbf0400080 fffff7e080010000 200000000080   3804 Active   M
   13        0        1     1 fffff5fbf0400078 fffff7e08000f000 200000000080   3804 Active   M
   14        0        1     1 fffff5fbf0400088 fffff7e080011000 200000000080   3804 Active   M
   15        0        1     1 fffff5fbf0400090 fffff7e080012000 200000000080   3804 Active   M
   16        0        1     1 fffff5fbf04000a8 fffff7e080015000 200000000080   3804 Active   M
    .
    .
    .

lkd> !pfn 0 1

 Page    Flink  Blk/Shr Ref V    PTE   Address  SavedPTE Frame  State
    1        0        1     1 ffff81fbdd800000 fffff7bb00000000 200000000080   3404 Active   M
```

```
lkd> dt nt!_MMPFN poi(nt!MmPfnDatabase)+(@@c++(sizeof(nt!_MMPFN))*1)
   +0x000 ListEntry        : _LIST_ENTRY [ 0x00000000`00000000 - 0xffff81fb`dd800000 ]
   +0x000 TreeNode         : _RTL_BALANCED_NODE
   +0x000 u1               : <anonymous-tag>
   +0x008 PteAddress       : 0xffff81fb`dd800000 _MMPTE
   +0x008 PteLong          : 0xffff81fb`dd800000
   +0x010 OriginalPte      : _MMPTE
   +0x018 u2               : _MIPFNBLINK
   +0x020 u3               : <anonymous-tag>
   +0x024 NodeBlinkLow     : 0
   +0x026 Unused           : 0y0000
   +0x026 Unused2          : 0y0000
   +0x027 ViewCount        : 0 ''
   +0x027 NodeFlinkLow     : 0 ''
   +0x028 u4               : <anonymous-tag>

lkd> dt nt!_MMPFNLIST
   +0x000 Total            : Uint8B
   +0x008 ListName         : _MMLISTS
   +0x010 Flink            : Uint8B
   +0x018 Blink            : Uint8B
   +0x020 Lock             : Uint8B

lkd> dt nt!_MMLISTS
   ZeroedPageList = 0n0
   FreePageList = 0n1
   StandbyPageList = 0n2
   ModifiedPageList = 0n3
   ModifiedNoWritePageList = 0n4
   BadPageList = 0n5
   ActiveAndValid = 0n6
   TransitionPage = 0n7
```

Visualisation des threads Prêt

La commande !ready du débogueur noyau permet de voir la liste des threads prêts à être exécutés à chaque niveau de priorité.

```
lkd> !ready
KSHARED_READY_QUEUE fffff801610f4700: (00) **``````````````````````````````````````````````````--
SharedReadyQueue fffff801610f4700: Ready Threads at priority 12
    THREAD ffffa704a0976080  Cid 1b4c.1cf4  Teb: 000000bb896e2000 Win32Thread: ffffa7049ddd8520 WAIT
SharedReadyQueue fffff801610f4700: Ready Threads at priority 8
    THREAD ffffa704a84d7080  Cid 0698.08f4  Teb: 0000000008565000 Win32Thread: ffffa704a52f75e0 RUNNING on processor
 80000000
    THREAD ffffa704a64a0080  Cid 1b4c.1e3c  Teb: 000000bb897f8000 Win32Thread: ffffa704a8289370 READY on processor 80000001
SharedReadyQueue fffff801610f4700: Ready Threads at priority 0
    THREAD ffffa7049bf9b080  Cid 0004.0038  Teb: 0000000000000000 Win32Thread: 0000000000000000 READY on processor 80000001
Processor 0: No threads in READY state
Processor 1: No threads in READY state

lkd> dt nt!_KSHARED_READY_QUEUE fffff801610f4700
   +0x000 Lock             : 0
   +0x008 ReadySummary     : 0x500
   +0x010 ReadyListHead    : [32] _LIST_ENTRY [ 0xfffff801`610f4710 - 0xfffff801`610f4710 ]
   +0x210 RunningSummary   : [64]  "???"
   +0x250 Span             : 0x2 ''
   +0x251 LowProcIndex     : 0 ''
   +0x252 QueueIndex       : 0x9 ''
   +0x253 ProcCount        : 0x2 ''
   +0x254 ScanOwner        : 0 ''
   +0x255 Spare            : [3]  ""
   +0x258 Affinity         : 3
```

```
+0x260 ReadyThreadCount : 4
+0x268 ReadyQueueExpectedRunTime : 0x1c1d40
```

Thread de page à zéro

Divers facteurs en matière de sécurité imposent que les processus mode utilisateur soient alimentés en trames de pages initialisés à zéro, expurgés en l'occurence de toute information issue de l'exécution d'un précédent processus. Pour répondre à ces exigences, le gestionnaire de mémoire donne lieu à un thread système spécial, dit thread de page à zéro, dont la fonction première est de remplir de zéros les pages de la liste des pages libres, et ce faisant assainir le contenu desdites pages.

Le thread de page à zéro attend un objet événement (MmZeroingPageEvent) qui lui indique quand entrer en scène, lequel objet est mis à l'état signalé quand la liste Libres a huit pages ou plus. Cependant, le thread de page à zéro n'est exécuté que s'il n'y a pas d'autres threads en cours d'exécution, le système lui attribuant un niveau de priorité très faible - en l'occurrence la priorité 0, alors que la priorité minimale d'un thread utilisateur est 1.

L'utilité de pages mises à zéro découle directement d'exigences de sécurité en ce qui concerne les vecteurs informationnels dont les processus disposent entre eux. Les processus mode utilisateur reçoivent automatiquement des trames de pages initialisées (qui n'ont de ce fait strictement aucune signification d'aucune sorte), de façon à les empêcher de lire le contenu mémoire d'un précédent processus.

```
lkd> !process 0 0 System
PROCESS ffffda8e17470380
    SessionId: none  Cid: 0004    Peb: 00000000  ParentCid: 0000
    DirBase: 001ad002  ObjectTable: ffffc4840fc05580  HandleCount: 2644.
    Image: System

lkd> !process ffffda8e17470380
PROCESS ffffda8e17470380
    SessionId: none  Cid: 0004    Peb: 00000000  ParentCid: 0000
    DirBase: 001ad002  ObjectTable: ffffc4840fc05580  HandleCount: 2631.
    Image: System
    VadRoot ffffda8e1abc5be0 Vads 8 Clone 0 Private 20. Modified 1490119. Locked 0.
    DeviceMap ffffc4840fc138a0
    Token                             ffffc4840fc067c0
    ElapsedTime                       30 Days 23:09:55.902
    UserTime                          00:00:00.000
    KernelTime                        00:06:52.453
    QuotaPoolUsage[PagedPool]         0
    QuotaPoolUsage[NonPagedPool]      272
    Working Set Sizes (now,min,max)  (32, 50, 450) (128KB, 200KB, 1800KB)
    PeakWorkingSetSize                451
    VirtualSize                       3 Mb
    PeakVirtualSize                   13 Mb
    PageFaultCount                    13111
    MemoryPriority                    BACKGROUND
    BasePriority                      8
    CommitCharge                      49

        .
        .
        .
        THREAD ffffda8e174d3080  Cid 0004.0038  Teb: 0000000000000000 Win32Thread: 0000000000000000 READY on processor 1
        Not impersonating
        DeviceMap                 ffffc4840fc138a0
        Owning Process            ffffda8e17470380       Image:         System
        Attached Process          N/A            Image:         N/A
        Wait Start TickCount      2689342        Ticks: 68 (0:00:00:01.062)
        Context Switch Count      288769         IdealProcessor: 1
        UserTime                  00:00:00.000
        KernelTime                00:00:40.218
        Win32 Start Address nt!MiZeroPageThread (0xfffff80529d82120)
```

```
Stack Init ffffed0c3e88ac10 Current ffffed0c3e88a640
Base ffffed0c3e88b000 Limit ffffed0c3e884000 Call 0000000000000000
Priority 0 BasePriority 0 PriorityDecrement 0 IoPriority 2 PagePriority 5
Child-SP          RetAddr           Call Site
ffffed0c`3e88a680 fffff805`29c318c2 nt!KiSwapContext+0x76
ffffed0c`3e88a7c0 fffff805`29c30f54 nt!KiSwapThread+0x3f2
ffffed0c`3e88a860 fffff805`29c58287 nt!KiCommitThreadWait+0x144
ffffed0c`3e88a900 fffff805`29c57d24 nt!KeWaitForMultipleObjects+0x287
ffffed0c`3e88aa10 fffff805`29d821f1 nt!MiWaitForFreePagesToZero+0x1c4
ffffed0c`3e88ab40 fffff805`29d30925 nt!MiZeroPageThread+0xd1
ffffed0c`3e88ab90 fffff805`29dc3d5a nt!PspSystemThreadStartup+0x55
ffffed0c`3e88abe0 00000000`00000000 nt!KiStartSystemThread+0x2a

.
.
.
```

Listes look-aside

Windows offre un mécanisme d'allocation rapide de mémoire, dit listes look-aside, dont le principal avantage tient au fait au fait de l'optimisation de l'utilisation de chaque liste individuelle. La différence fondamentale entre pools et listes look-aside est la suivante : alors que les allocations des pools généraux (paginé ou non paginé) peuvent varier en taille, une liste look-aside ne contient que des blocs de taille fixe.

Pour plus d'informations concernant les listes look-aside, voir routines ExInitializeNPagedLookasideList, ExAllocateFromNPagedLookasideList, ExDeletePagedLookasideList, ExFreeToNPagedLookasideList, ExDeleteNPagedLookasideList, ExInitializePagedLookasideList, ExAllocateFromPagedLookasideList et ExFreeToPagedLookasideList.

Pagination à la demande

Le gestionnaire de mémoire de Windows emploie un algorithme de pagination à la demande avec regroupement (*clustering*) pour charger les pages en mémoire. Quand un thread fait l'objet d'un défaut de pages, le gestionnaire de mémoire charge la page manquante plus un petit nombre de pages voisines (d'où le terme regroupement), cela en application du principe de localité. Cette stratégie a pour objectif de minimiser le nombre d'E/S de pagination subies par un thread.

Dans un système avec pagination à la demande, il est naturel qu'un processus subisse lors de son démarrage de nombreux défaut de page, ses threads devant en la circonstance référencer le jeu essentiel de pages dont il ont besoin pour leur exécution. Une fois ces pages chargées en mémoire, l'activité de pagination du processus diminue fortement.

Prefetcher

Au démarrage de l'ordinateur ou d'une application, les nombreux défauts de page qu'encourt chaque processus fragilisent les performances et nuisent à la réactivité globale du système. Aussi, pour optimiser ces différentes phases, Windows s'oriente vers un moteur de pré extraction intelligent appelé *prefetcher logique*.

Quand le prefetcher est actif, le gestionnaire de mémoire informe le prefetcher des défauts de page noyau, que ces défauts concernents des données à lire depuis le disque (défauts de page matériels) ou ailleurs en mémoire (défauts de page logiciels). Le processus du prefetcher surveille par défaut les 10 premières secondes du démarrage des applications, et conserve des fichiers de traces pour les 120 programmes les plus récemment utilisés, de telle sorte que les anciennes données prefech sont inmanquablement et automatiquement écrasées au profit des plus récentes. En ce qui concerne le système, le prefetcher intensifie ses activités de surveillance depuis le début de l'amorcage jusqu'aux 30 secondes qui suivent le démarrage du shell utilisateur (en principe, l'explorateur) ou, à défaut, jusqu'aux 60 secondes qui suivent l'initialisation des services Windows ou, à défaut

encore, pendant 120 secondes ; de ces trois délais, celui qui se termine en premier met fin à la procédure. Par la suite, lors d'un démarrage/lancement ultérieur, les données ainsi repertoriées sont pré chargées en mémoire de manière asynchrone, plutot que chargés au fur et à mesure des besoins.

Le prefetcher enregistre les fichiers auquel il donne lieu dans le répertoire \Windows\Prefetch. Le nom du fichier est formé du nom de l'application tracée, suivi d'un trait d'union et de la représentation hexadécimale d'un hachage du chemin d'accès au fichier. Le fichier résultant porte l'extension .pf. Cette règle de dénomination admet deux exceptions. La première concerne les images qui hébergent d'autres composants, ce qui inclut par exemple la console MCC (\Windows\System32\Mmc.exe), le processus hôte de services (\Windows \System32\Svchost.exe) et DllHost (\Windows\System32\Dllhost.exe). Étant donné que les logiciels additionnels sont spécifiés sur la ligne de commande pour ces applications, le prefetcher inclut la ligne de commande dans son hachage. L'autre exception concerne le nom du fichier qui renferme la trace de l'amorcage, qui s'appelle toujours NTOSBOOT-B00DFAAD.PF.

Lors de l'initialisation du système ou quand une appplication démarre, le prefetcher regarde dans le répertoire Prefetch pour voir s'il existe un fichier de trace pour le scénario en question. Si tel est le cas, il sollicite alors le pilote NTFS en vue de pré extraire toutes références au fichier de méta données MFT, lit le contenu de chacun des repertoires référencés et, finalement, ouvre chaque fichier référencé. Il fait ensuite appel au gestionnaire de mémoire pour lire le code et les données qui sont spécifiés dans la trace mais qui ne sont pas encore en mémoire. Le gestionnaire de mémoire effectue l'ensemble des lectures sur une base asynchrone, puis attend qu'elles soient terminées avant de laisser se poursuivre le démarrage.

Les informations de configuration se rapportant au prefetcher sont stockées dans le registre sous la clé HKLM \SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\PrefetchParameters.

Working sets

Les sous-ensembles de pages virtuelles qui résident en mémoire physique sont dits working sets. Il existe trois sortes de working set :

Il existe deux sortes de working set :

- Les working sets de processus contiennent les pages que référencent les threads d'un processus. Par extension, le working set d'un processus donné désigne la partie de l'espace d'adressage virtuel qui à la fois résidente en mémoire physique et propriété de ce processus.

- Le working set système contient le sous-ensemble résident du code système paginable, ce qui inclut par exemple le noyau et les pilotes, du pool paginé et du cache système.

La mémoire physique allouée à un processus aux divers stades de son exécution est dite plage de travail du processus. Une plage de travail privée représente la mémoire physique propre à un processus, qui ne peut pas par conséquent pas être employée par d'autres processus. Les processus réalisent également des opérations sur d'autres pages de mémoire physique, qui sont quant à elles partagées et auxquelles plusieurs processus peuvent faire référence.

Le working set de chaque processus, y compris celui du système, a une taille minimale, une taille courante et une taille maximale. S'il a un working set trop petit, un processus risque de référencer une page absente de la mémoire physique, avec comme conséquence un nombre élevé de défauts de page et de défauts de cache. S'il a un working set trop grand, le processus s'expose plus faiblement lui-même aux erreurs de page, mais met en danger d'autres processus, les privant de la mémoire dont ils pourraient avoir besoin.

Le gestionnaire de mémoire virtuelle essaie de dimensionner les working sets dans l'idée que tous les processus du système soient en équilibre. Hors situation de ressources faibles, il assure le maintien en mémoire physique d'un nombre suffisant de pages, et permet à la taille courante de chacun des working set d'augmenter si nécessaire jusqu'à son maximum.Les processus disposant du privilège PROCESS_SET_QUOTA peuvent dépasser ces valeurs au moyen de l'API SetProcessWorkingSetSize.

Working set système

De même que les processus ont des workings sets, le code et les données paginables du système d'exploitation sont gérés par un *working set système*, lequel contient des pages qui se rapportent à l'un ou l'autre des composants suivants :

- Pages du cache système
- Pool paginé
- Code et données paginables du noyau
- Code et données paginables des pilotes de périphérique
- Vues mappées du système

Tableau 10.13. Working set système

Variable système	Compteur de performance	Description
MmPagedPoolPage		Mémoire physique consommée par le pool paginé.
MmSystemCachePage	Mémoire: Octets résidents dans le cache système	Mémoire physique consommée par le cache système.
MmSystemCacheWs.PageFaultCount	Mémoire: Défauts de cache	Défauts de page dans le working set système
MmSystemCacheWs.Peak	Mémoire: Octets max. du cache	Taille maximale du working set système.
MmSystemCacheWs.WorkingSetSize	Mémoire: Octets du cache	Taille totale du working set système.
MmSystemCodePage		Mémoire physique consommée par le code paginable de Ntoskrnl.
MmSystemDriverPage		Mémoire physique consommée par le code paginable des pilotes.

Working Set Manager

L'analyse et l'élagage de working sets s'effectuent dans le cadre d'une routine noyau spéciale appelée *working set manager*, à laquelle le balance set manager passe le relais toutes les secondes, et aussi quand la mémoire libre tombe en deçà d'un certain seuil.

Windows s'efforce d'équilibrer automatiquement les demandes de mémoire physique en écrivant sur le disque les pages modifiées ; mais, quand les modifications de page s'enchaînent à un rythme trop soutenu, il faut alors plus de mémoire pour répondre aux besoins. À ce titre, dès que la place vient à manquer en mémoire physique, le working set manager est sollicitée afin de mener à bien l'élagage des jeux de travail, et ce faisant accroitre la mémoire libre sur le système.

Le gestionnaire de working set examine suite à son entrée en scène la mémoire disponible, actualise sur le sujet diverses métriques et, partant de là, détermine s'il faut procéder à un élagage. Si la quantité de mémoire libre est suffisante, le gestionnaire agit de manière proactive en calculant le nombre de pages susceptibles d'être supprimées au cas où le besoin se faisait sentir. Le gestionnaire ajuste dynamiquement la fréquence à laquelle il examine les working sets, et optimise automatiquement l'ordre de la liste des processus candidats à l'élagage.

Dans l'éventualité d'un élagage, le gestionnaire de working set repère quels processus il a le plus intérêt à cibler (en fonction de leur consommation), quelles pages il a plus avantage à supprimer (en fonction de leur âge, voir paragraphes suivants) et entame la procédure idoine. Si, malgré ces ajustements, la mémoire est encore trop faible, le gestionnaire continue à libérer des pages des working sets de processus jusqu'à atteindre le seuil minimal.

En interne, l'élagage de working set s'effectue sur la base des pages auxquelles on n'a pas accédé récemment. Le gestionnaire de working set s'appuie pour ce faire sur le bit Accessed du PTE matériel. Si ledit bit vaut zéro, il

augmente l'âge de la page, cela afin de mettre en évidence que cette dernière n'a pas été référencée depuis la dernière analyse d'élagage de working set. Si le bit Accessed du PTE matériel est armé, le gestionnaire de working set le réinitialise (comprendre le remet à zéro) et passe à la page suivante du working set. Cet examen dure jusqu'à ce que le gestionnaire de working set ait supprimé le nombre de pages désirées ou jusqu'à ce que l'analyse soit revenue à son point de départ.

Working sets de processus

Il est possible d'obtenir des informations sur un working set donné en faisant appel à GetProcessMemoryInfo, laquelle fonction alimente une structure `PROCESS_MEMORY_COUNTERS` dont deux attributs sont particulièrement pertinents dans ce contexte : `WorkingSetSize`, qui correspond à la quantité de mémoire physique actuellement allouée pour le compte d'un processus, et `PeakWorkingSetSize`, soit la valeur maximale atteinte par cet ensemble au cours de l'exécution.

Chaque processus est au niveau de son working set assujetti à une limite basse et une limite haute. La limite inférieure, ou minima de working set (MinimumWorkingSetSize, généralement 50 pages) se concentre sur le fait d'offrir aux processus en cours certaines garanties vis-à-vis de la mémoire qu'ils sont en droit d'exploiter. La limite supérieure, ou maxima de working set (MaximumWorkingSetSize, généralement 345 pages), fixe un seuil quant à la quantité de mémoire physique sur laquelle un processus est en mesure de trouver appui. Notez qu'en pratique, ces restrictions sont de base purement indicatives : un processus peut temporairement voir son working set réduit en dessous de sa limite minimale si la mémoire du système arrive à saturation, ou au contraire bénéficier d'une extension au-delà de sa limite maximale s'il existe suffisamment de mémoire disponible.

Ainsi que nous venons de le mentionner, par défaut, Windows n'applique pas strictement les tailles minimale et maximale de working sets, mais ajuste ces valeurs de manière dynamique, selon que la mémoire se trouve plus ou moins sollicitée. Contrepoint de cette perspective, la fonction `SetProcessWorkingSetSizeEx` permet l'imposition de limites strictes (*hard limits*), et honore conformément à ces aspects deux options : `QUOTA_LIMITS_HARDWS_MIN_ENABLE` et `QUOTA_LIMITS_HARDWS_MAX_ENABLE`. Notez, sur le plan des performances, que lorsqu'un processus est configuré de manière à respecter une limite strictement imposée sur la taille maximale de son working set, toute tentative d'allocation supplémentaire de mémoire au sein de ce dernier entraîne inévitablement l'éviction d'autres pages déjà présentes en mémoire physique. (Dit de façon plus imagée, le processus en question joue contre lui-même, engendrant alors un nombre anormalement élevée de défauts de page.) Ainsi, une telle contrainte, bien qu'appréciable dans quelques scénarios visant à limiter la consommation d'un processus, peut paradoxalement affecter de manière négative les performances du système, forcé pour l'occasion d'échanger continuellement entre la mémoire et le fichier de pagination.

Compteur	Description
Processus: Plage de travail	Taille courante (en octets) du working set du processus
Processus: Plage de travail max.	Taille maximale (en octets) du working set du processus
Processus: Défauts de page/s	Nombre de défauts de page se produisant chaque seconde pour le processus

Pour connaitre le total de tous les working sets de processus, sélectionnez le processus _Total dans la zone d'instances de la console Performances. (Notez que ce dernier ne correspond pas en réalité à un vrai processus, mais à une instance cumulative dérivée des compteurs relatifs à l'ensemble des processus en cours d'exécution.) La valeur montrée dans ce contexte devrait toutefois être accueillie avec un certain scepticisme, dans la mesure où la taille de chaque working set de processus inclut les pages partagées entre plusieurs processus. De ce fait, dans l'éventualité où deux processus ou plus se partagent une page, celle-ci est comptée dans chaque working set.

Fonctions	**Services**
EmptyWorkingSet	NtSetInformationProcess (ProcessQuotaLimits)
QueryWorkingSet	NtQueryVirtualMemory (MemoryWorkingSetList)
QueryWorkingSetEx	NtQueryVirtualMemory (MemoryWorkingSetExList)
GetProcessMemoryInfo	NtQueryInformationProcess (ProcessVmCounters)

Fonctions	Services
GetProcessWorkingSetSize(Ex)	NtQueryInformationProcess (ProcessQuotaLimits)
SetProcessWorkingSetSizeEx(Ex)	NtSetInformationProcess

VAD (Virtual Address Descriptor)

Le gestionnaire de mémoire gère la mémoire virtuelle à l'aide d'un algorithme de pagination à la demande, et ce faisant attend qu'un thread référence une adresse (et partant, subisse un défaut de page) pour charger les pages idoines depuis le disque. Cette façon de procéder est une forme d'*évaluation paresseuse* : une tâche n'est accomplie qu'en dernier ressort, exclusivement si les circonstances l'exigent.

Le gestionnaire de mémoire emploie l'évaluation paresseuse non seulement pour amener des pages en mémoire, mais aussi pour construire les tables qui décrivent les pages. Ainsi, quand un thread alloue de la mémoire virtuelle (VirtualAlloc), le gestionnaire de mémoire ne donne pas immédiatement lieu aux tables de pages idoines (lesquelles permettraient dans ce contexte d'accéder à toute la plage de mémoire allouée), mais diffère cette tâche jusqu'à ce que le thread subisse un défaut de page, à la suite de quoi il crée alors une table de pages pour cette page. Une telle stratégie améliore significativement les performances des processus qui réservent et/ou engagent beaucoup de mémoire, mais y accèdent peu souvent.

Afin de suivre les adresses virtuelles qui ont été réservées par un processus (et, par extension, lesquelles ne l'ont pas été), le gestionnaire de mémoire s'appuie sur un ensemble de structures de données dites descripteurs d'adresse virtuelle (VAD, Virtual Address Descriptor), qui décrivent l'état de l'espace d'adressage du processus, et enregistrent les plages d'adresses qui lui sont accessibles. En interne, Les VAD sont gérés en tant qu'arbre binaire auto équilibré.

Quand un thread réserve de l'espace d'adressage ou mappe une vue d'une section, le gestionnaire de mémoire crée un VAD dont les attributs dérivent des informations qui émanent de la requête, y compris l'étendue (comprendre la taille) de la plage d'adressage qui est réservée, le fait de savoir si la plage est partagée ou privée, si un processus enfant peut hériter du contenu de la plage, et ainsi de suite.

Les descripteurs d'adresse virtuelle ne supplantent aucunement les structures de pagination via lesquelles le processeur implémente la mémoire, que le gestionnaire de mémoire doit pour l'occasion continuer d'honorer. De ce fait, quand un thread accède pour la première fois à une adresse, le gestionnaire de mémoire doit créer un PTE pour la page contenant l'adresse. Pour ce faire, il considère le VAD approprié et s'appuie sur les informations dudit VAD de sorte à alimenter le PTE. Si l'adresse est hors de la plage couverte par le VAD ou dans une plage d'adresses qui sont réservées mais pas engagées, le gestionnaire de mémoire reconnait que le thread n'a pas alloué la mémoire avant de l'utiliser et génère une violation d'accès.

Chaque VAD est en interne représenté par une structure MMVAD.

Visualisation de VAD

L'extension !vad du débogueur noyau permet de voir les VAD d'un processus donné. La commande prend en entrée l'adresse de la racine de l'arbre VAD associé au processus (champ VadRoot du bloc EPROCESS), et affiche un résultat similaire à ce qui suit :

```
lkd> !process 0 1 notepad.exe
PROCESS ffffda8e1f8f0080
    SessionId: 1  Cid: 10e0    Peb: e91f5dd000  ParentCid: 0db4
    DirBase: 5576e002  ObjectTable: ffffc484135f57c0  HandleCount: 229.
    Image: notepad.exe
    VadRoot ffffda8e1f952c60 Vads 95 Clone 0 Private 535. Modified 0. Locked 0.
    .
    .
    .

lkd> !vad ffffda8e1f952c60
```

```
VAD             Level         Start             End              Commit
ffffda8e1d853790  6           3c5e0           3c5e0              1 Private      READWRITE
ffffda8e21266ca0  5           3c5f0           3c5f0              1 Private      READWRITE
ffffda8e1d8479e0  6           7ffe0           7ffe0              1 Private      READONLY
ffffda8e1d848e80  4           7ffed           7ffed              1 Private      READONLY
ffffda8e1d84f870  6         e91f380         e91f3ff             20 Private      READWRITE
ffffda8e1d849ba0  5         e91f400         e91f5ff              9 Private      READWRITE
ffffda8e1d853ba0  3         e91f700         e91f77f             20 Private      READWRITE
ffffda8e1d853380  5         e91f780         e91f7ff             20 Private      READWRITE
ffffda8e1d853600  6         e91f800         e91f87f             20 Private      READWRITE
ffffda8e1f943ee0  4        26abc5e0        26abc5ef              0 Mapped       READWRITE          Pagefile section,
 shared commit 0x10
ffffda8e1d852ca0  6        26abc5f0        26abc5fc              2 Private      READWRITE
ffffda8e1f9532a0  5        26abc600        26abc61a              0 Mapped       READONLY           Pagefile section,
 shared commit 0x1b
ffffda8e1f952620  6        26abc620        26abc623              0 Mapped       READONLY           Pagefile section,
 shared commit 0x4
ffffda8e1f9526c0  2        26abc630        26abc631              0 Mapped       READONLY           Pagefile section,
 shared commit 0x2
ffffda8e1d8523e0  5        26abc640        26abc641              2 Private      READWRITE
ffffda8e1f9533e0  6        26abc650        26abc716              0 Mapped       READONLY           \Windows
\System32\locale.nls
.
.
.
Total VADs: 93, average level: 5, maximum depth: 7
Total private commit: 0x24d pages (2356 KB)
Total shared commit:  0x2714 pages (40016 KB)
```

Page de garde

Le mécanisme de page de garde est un mécanisme permettant d'assigner une page frontière dans le cas de ressources allouées dynamiquement. C'est par exemple utile dans le traitement de structures de données amenées à évoluer en fonction des accès, typiquement pour la gestion d'une pile.

La création d'une page de garde se fait au moyen du positionnement de l'attribut PAGE_GUARD sur la région. (Voir fonctions VirtualAlloc, VirtualAllocEx, VirtualProtect, VirtualProtectEx.)

Lorsqu'un accès est effectué dans la page incriminée, l'exception STATUS GUARD PAGE VIOLATION est soulevée et le système désarme le drapeau PAGE_GUARD, laissant le programme gérer la faute.

Fichiers de pagination

Au démarrage du système, le processus Gestionnaire de session (Smss.exe) détermine quels sont les fichiers de pagination à ouvrir en s'appuyant sur la valeur de registre HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\PagingFiles, laquelle contient le nom, la taille minimale et la taille maximale de chaque fichier de pagination.

Windows autorise jusqu'à 16 fichiers de pagination et fixe une taille seuil pour chaque fichier de pagination : 4095 Mo sur les systèmes x86 exécutant le noyau standard, 16 To en ce qui concerne les systèmes x64 et les systèmes x86 exécutant le noyau PAE.

Une fois ouverts, les fichiers de pagination ne peuvent pas être supprimés tandis que le système est en cours d'exécution, impossibilité qui découle du fait que le processus System conserve un handle ouvert sur chacun d'entre eux.

Comme le fichier de pagination contient des parties de la mémoire virtuelle des processus et du noyau, il est possible de configurer le système en vue, au cours du processus d'arrêt, de la suppression automatique dudit fichier. Pour ce faire, définissez la valeur HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\ClearPageFileAtShutdown à 1.

Par défaut, le fichier de pagination est situé à la racine de la partition d'amorçage et porte le nom pagefile.sys. Sa taille initiale est égale à une fois la taille de la mémoire vive (1,5 fois sur les appareils dotés de moins d'un Go) et sa taille maximale à 3 fois la taille de la mémoire vive).

Les composants dont émane des fichiers de pagination (en premier lieu, Smss lors du démarrage du système) emploient le service système NtCreatePagingFile, lequel exige de l'appelant le privilège SeCreatePagefilePrivilege. Les fichiers de pagination sont toujours créés en tant que fichiers non compressés, y compris si le repertoire les contenant montre la particularité contraire.

Visualisation de fichiers d'échange. Chaque fichier d'échange dans Windows est représenté au moyen d'une structure MMPAGING_FILE, observable via l'extension !dt du débogueur noyau :

```
lkd> dt nt!_MMPAGING_FILE 0xffffa704a02dd220
   +0x000 Size             : 0x28000
   +0x008 MaximumSize      : 0x240000
   +0x010 MinimumSize      : 0x28000
   +0x018 FreeSpace        : 0x1f056
   +0x020 PeakUsage        : 0xcef9
   +0x028 HighestPage      : 0
   +0x030 FreeReservationSpace : 0x23e9c
   +0x038 File             : 0xffffa704`a65fc3f0 _FILE_OBJECT
   +0x040 Entry            : [2] 0xffffa704`a051b010 _MMMOD_WRITER_MDL_ENTRY
   +0x050 PfnsToFree       : _SLIST_HEADER
   +0x060 PageFileName     : _UNICODE_STRING "\??\C:\pagefile.sys"
   +0x070 Bitmaps          : 0xffffa704`9e540000 _MI_PAGING_FILE_SPACE_BITMAPS
   +0x078 AllocationBitmapHint : 2
   +0x07c LargestAllocationCluster : 0x100
   +0x080 RefreshAllocationCluster : 0x1533d1
   +0x084 LastRefreshAllocationCluster : 0
   +0x088 ReservedClusterSizeAggregate : 0x400
   +0x08c MaximumRunLengthInBitmaps : 0x1f
   +0x090 BitmapsCacheLengthTree : _RTL_RB_TREE
   +0x0a0 BitmapsCacheLocationTree : _RTL_RB_TREE
   +0x0b0 BitmapsCacheFreeList : _LIST_ENTRY [ 0xffffa704`a60fbfe8 - 0xffffa704`a60fcc60 ]
   +0x0c0 BitmapsCacheEntries : 0xffffa704`a60f9000 _MI_PAGEFILE_BITMAPS_CACHE_ENTRY
   +0x0c8 ToBeEvictedCount : 0
   +0x0c8 HybridPriority   : 0
   +0x0cc PageFileNumber   : 0y0000
   +0x0cc WsSwapPagefile   : 0y0
   +0x0cc NoReservations   : 0y1
   +0x0cc VirtualStorePagefile : 0y0
   +0x0cc SwapSupported    : 0y1
   +0x0cc NodeInserted     : 0y1
   +0x0cc StackNotified    : 0y1
   +0x0cc BackedBySCM      : 0y0
   +0x0cc Spare0           : 0y0000
   +0x0ce AdriftMdls       : 0y0
   +0x0ce Spare1           : 0y0000000 (0)
   +0x0cf IgnoreReservations : 0y0
   +0x0cf Spare2           : 0y0000000 (0)
   +0x0d0 PageHashPages    : 0x47
   +0x0d4 PageHashPagesPeak : 0x49
   +0x0d8 PageHash         : 0xffffa6ff`39a00000  -> 0
   +0x0e0 FileHandle       : 0xffffffff`80000564 Void
   +0x0e8 Lock             : 0
   +0x0f0 LockOwner        : (null)
   +0x0f8 FlowThroughReadRoot : _RTL_AVL_TREE
   +0x100 Partition        : 0xfffff801`64465140 _MI_PARTITION
   +0x108 FileObjectNode   : _RTL_BALANCED_NODE
```

- *Partition d'amorçage (BootPartition)* Indique si le fichier se situe sur la partition de démarrage, permettant dans cette éventualité de l'utiliser pour l'enregistrement d'informations sur le système suite à un effondrement de ce dernier (données d'effondrement).

- *Objet fichier (File)* Adresse de l'objet fichier représentant une instance du fichier
- *Identifiant d'objet (FileHandle)* Handle ouvert sur le fichier. Voir service NtCreatePagingFile.
- *Taille maximale (MaximumSize)* Taille maximale que le fichier est en droit d'occuper sur le support de stockage où il réside. Voir service NtCreatePagingFile (paramètre MaximumSize).
- *Taille minimale (MinimumSize)* Voir service NtCreatePagingFile (paramètre MinimumSize).
- *Nom du fichier de pagination (PageFileName)* Nom pleinement qualifié du fichier de pagination. Voir service NtCreatePagingFile (paramètre FileName) ; routine MmGetPageFileInformation ; structure SYSTEM_PAGEFILE_INFORMATION (attribut PageFileName).

Tableau 10.14. Eléments auxiliaires sous-jacents aux fichiers de pagination

Elément	Lieu	Description
NumberOfPagingFiles	MI_VISIBLE_PARTITION	
PageFileCreationLock	MI_PARTITION_MODWRITES	
PagingFile	MI_VISIBLE_PARTITION	

Objets section

Les constructions mode noyau par lesquelles Windows octroi à deux processus ou plus la possibilité de partager un même emplacement de mémoire sont dits *objets section*. Un objet section peut être mappé vers le fichier de pagination ou vers tout autre fichier du disque. (Nous reviendrons plus loin sur la raison de cette distinction.)

Les sections sont utilisées à diverses fins, notamment celles que voici :

- L'exécutif emploie des sections pour charger les images exécutables en mémoire.
- Le gestionnaire de cache utilise des sections pour accéder aux données d'un fichier mis en cache.
- Les applications Windows s'appuient sur des sections de sorte à mapper un fichier dans un espace d'adressage de processus.

Pour chaque fichier ouvert, il existe une structure de pointeurs d'objet section qu'emploie le gestionnaire de mémoire et le gestionnaire de cache pour enregistrer des informations concernant le mappage et la mise en cache du fichier. Cette structure pointe vers une ou deux zones de contrôle. Une zone de contrôle sert à mapper le fichier quand il est utilisé en tant que fichier de données, l'autre zone servant à mapper le fichier quand il est exécuté en tant qu'image exécutable.

Les objets section, à l'instar de nombreuses autres entités, sont alloués et démantelés par le gestionnaire d'objets, le gestionnaire de mémoire définissant quand à lui le corps - et partant, les attributs et les services génériques qui s'y rapportent - de l'objet section.

La liste qui suit décrit les attributs spécifiques des objets section.

- *Protection de page (InitialPageProtection)* Protection mémoire de niveau page, assignée à toutes les pages de la section lors de sa création.
- *Taille maximale (SizeOfSection)* Plus grosse taille (en octets) que la section puisse atteindre ; si mappage d'un fichier, la taille maximale est celle du fichier.
- *Basée (Flags.Based)* Indique si la section est une section basée ou non basée. Voir constante SEC_BASED.
- *Pages engagées (Flags.Commit)* Indique si les pages alloués pour la section le sont à l'état engagé.

- *Fichier mappé (Flags.File)* Indique si la section est créée à partir d'un fichier.
- *Fichier image (Flags.Image)* Indique si la section est créée à partir d'une image exécutable. Voir constante SEC_IMAGE.
- *Aucun changement (Flags.NoChange)* Indique si la section, une fois mappé vers un fichier, peut être démappée ou voir ses options de protection modifiées. Voir constante SEC_NO_CHANGE.
- *Pages réservés (Flags.Reserve)* Indique si les pages alloués pour la section le sont à l'état réservé (lesdites pages pourront être engagées plus tard).

Tableau 10.15. Autorisations spécifiques aux sections

Constante	Valeur	Signification
SECTION_QUERY	0x0001	Requis pour demander des informations relatives à une section. Controlé par NtQuerySection.
SECTION_MAP_WRITE	0x0002	Requis pour obtenir l'accès en écriture à une section.
SECTION_MAP_READ	0x0004	Requis pour obtenir l'accès en écriture à une section.
SECTION_MAP_EXECUTE	0x0008	Requis pour obtenir l'accès en exécution à une section.
SECTION_EXTEND_SIZE	0x0010	Requis pour étendre une section. Controlé par NtExtendSection.

Tableau 10.16. SECTION_INFORMATION_CLASS

Nom	Valeur	Type de données associé
SectionBasicInformation	00	SECTION_BASIC_INFORMATION
SectionImageInformation	01	SECTION_IMAGE_INFORMATION

SECTION_BASIC_INFORMATION.

```
typedef struct _SECTION_BASIC_INFORMATION {
  PVOID BaseAddress;
  ULONG Attributes;
  LARGE_INTEGER Size;
} SECTION_BASIC_INFORMATION, *PSECTION_BASIC_INFORMATION;
```

Tableau 10.17. Valeurs initiales des champs de la structure SECTION_BASIC_INFORMATION

Champ	Valeur
BaseAddress	SECTION Address.StartingVpn
Attributes	SECTION Flags
MaximumSize	SECTION SizeOfSection

```
typedef struct _SECTION_IMAGE_INFORMATION
{
    PVOID TransferAddress;
    ULONG ZeroBits;
    SIZE_T MaximumStackSize;
    SIZE_T CommittedStackSize;
    ULONG SubSystemType;
    union {
        struct {
            WORD SubSystemMinorVersion;
            WORD SubSystemMajorVersion;
        };
        ULONG SubSystemVersion;
    };
    ULONG GpValue;
```

```
    USHORT ImageCharacteristics;
    USHORT DllCharacteristics;
    USHORT Machine;
    UCHAR ImageContainsCode;
    union {
        UCHAR ImageFlags;
        struct {
            UCHAR ComPlusNativeReady: 1;
            UCHAR ComPlusILOnly: 1;
            UCHAR ImageDynamicallyRelocated: 1;
            UCHAR ImageMappedFlat:1;
            UCHAR BaseBelow4gb : 1;
            UCHAR ComPlusPrefer32bit : 1;
            UCHAR Reserved : 2;
        };
    };
    ULONG LoaderFlags;
    ULONG ImageFileSize;
    ULONG CheckSum;
} SECTION_IMAGE_INFORMATION, *PSECTION_IMAGE_INFORMATION;
```

Tableau 10.18. Valeurs initiales des champs de la structure SECTION_IMAGE_INFORMATION

Champ	Valeur
CommittedStackSize	IMAGE_OPTIONAL_HEADER SizeOfStackCommit
DllCharacteristics	IMAGE_OPTIONAL_HEADER DllCharacteristics
ImageCharacteristics	IMAGE_FILE_HEADER Characteristics
ImageContainsCode	IMAGE_OPTIONAL_HEADER SizeOfCode
LoaderFlags	IMAGE_OPTIONAL_HEADER LoaderFlags
Machine	IMAGE_FILE_HEADER Machine
MaximumStackSize	IMAGE_OPTIONAL_HEADER SizeOfStackReserve
SubSystemType	IMAGE_OPTIONAL_HEADER Subsystem
SubSystemMajorVersion	IMAGE_OPTIONAL_HEADER MajorSubsystemVersion
SubSystemMinorVersion	IMAGE_OPTIONAL_HEADER MinorSubsystemVersion
TransferAddress	IMAGE_OPTIONAL_HEADER ImageBase + IMAGE_OPTIONAL_-HEADER AddressOfEntryPoint

Tableau 10.19. Opérations sur les sections

Opération	Fonction	Service	Routine
Créer un objet section	CreateFileMapping	NtCreateSection	MmCreateSection
Ouvrir un objet section	OpenFileMapping	NtOpenSection	ObOpenObjectByName
Mapper une vue d'une section	MapViewOfFile(Ex)	NtMapViewOfSection	MmMapViewOfSection
Démapper la vue d'une section	UnmapViewOfFile	NtUnmapViewOfSection	MiUnmapViewOfSection
Recueillir des informations concernant un objet section	-	NtQuerySection	-
Étendre la taille d'une section		NtExtendSection	MmExtendSection
Reconnaître que deux noms correspondent au même fichier	-	NtAreMappedFilesTheSame	-

Tableau 10.20. Attributs de section

Constante	Valeur
SEC_BASED	0x00200000
SEC_NO_CHANGE	0x00400000
SEC_FILE	0x00800000
SEC_IMAGE	0x01000000
SEC_RESERVE	0x04000000
SEC_COMMIT	0x08000000

SEC_NOCACHE	0x10000000
SEC_GLOBAL	0x20000000
SEC_LARGE_PAGES	0x80000000

- `SEC_IMAGE` Le fichier doit être traité comme une image exécutable. Les pages mémoire mappées en la circonstance le sont conformément aux permissions d'exécution et de protection spécifiées dans l'en-tête PE. Cela signifie que certaines sections (comme .text) seront marquées comme exécutables mais non modifiables.

- `SEC_IMAGE_NO_EXECUTE` Similaire à `SEC_IMAGE`, à ceci près qu'aucune page mémoire contenant du code n'est exécutable. Toute tentative d'aller contre un tel schéma, autrement dit exécuter une routine située parmi ces pages, se verrait immédiatement sanctionnée par une exception de protection mémoire.

- `SEC_NOCACHE` Désactive la mise en cache du fichier. Les échanges dont il est le vecteur (lectures et écritures depuis et vers le fichier) s'effectuent sans intermédiaire entre la mémoire et le disque, sans passer par le cache du gestionnaire de fichiers.

- `SEC_WRITECOMBINE` Optimise les accès mémoire via réunion de plusieurs écritures en une seule transaction.

- `SEC_LARGE_PAGES` Caractérise l'emploi pour une section de pages mémoire de grande taille.

Piles

Piles utilisateur

Chaque thread utilisateur sous Windows possède deux piles : une pile mode utilisateur, qui est extensible, et une pile mode noyau, plus petite et non extensible. Le passage d'une pile à l'autre est pris en charge automatiquement lors des transitions entre les modes (anneaux) de protection.

Lors de la création d'un thread, le gestionnaire de mémoire réserve automatiquement une quantité de mémoire prédéterminée, par défaut 1 Mo, et confère à cette région les propriétés caractéristiques d'un espace de pile. Sur ce Mo réservé, seuls les 64 premiers Ko sont engagés, plus une page de garde. Si la pile d'un thread se développe assez pour mettre en route les mécanismes régissant ce type de page, une exception se produit et le système comprend de cette manière qu'il doit réévaluer la quantité de mémoire octroyée au thread pour constituer son espace de pile.

Les tailles initialement réservé et engagé pour la pile d'un thread sont stipulées au niveau du fichier image sous-jacent, plus précisément dans l'entête optionnelle par le biais des valeurs, respectivement, SizeOfStackReserve et SizeOfStackCommit.

Un thread peut modifier l'espace de pile initialement engagé à l'aide du paramètre dwStackSize des fonctions CreateThread et CreateRemoteThread. Il est possible soit de définir une taille personnalisée, que le système arrondira à la page la plus proche en mémoire, soit de laisser ce paramètre à 0, ce qui aura pour effet d'utiliser la taille véhiculée dans l'en-tête de l'exécutable.

Dès qu'un thread termine son exécution, sa pile est libérée, y compris lorsque la situation résulte d'une terminaison forcée.

Attribut	**Lieu**	**Autres références**
SizeOfStackCommit	IMAGE_NT_HEADERS	ImageNtHeader
SizeOfStackReserve	IMAGE_NT_HEADERS	ImageNtHeader
StackBase	TEB.NtTib	NtQueryInformationThread (ThreadInformationClass = ThreadBasicInformation)

Attribut	Lieu	Autres références
StackLimit	TEB.NtTib	NtQueryInformationThread (ThreadInformationClass = ThreadBasicInformation)

Pile DPC

```
En plus des structures de stockage qui font partie de la dynamique normale des threads (piles mode utilisateur et piles
 mode noyau), Windows gère pour chaque processeur une pile spéciale utilisée lors de la concrétisation des appels de
 procédure différée (DPC). Une telle façon de procéder vise à isoler le code exécutant un DPC de la pile noyau du thread
 actuel - dont le contenu, vu que les DPC s'exécutent dans le contexte d'un thread arbitraire, est sans rapport avec les
   opérations réelles qui doivent être menées à l'issue d'une interruption DPC. Au cours des différentes actions dont il
 résulte un changement de privilège (SYSENTER/SYSCALL), la pile DPC est également configurée en tant que pile initiale
 associée au thread en cours d'exécution.
```

Pour trouver l'adresse de la pile DPC liée à un processeur, consultez la valeur de l'attribut DpcStack au niveau de la région de contrôle dudit processeur.

Pile d'interruption

Pour éviter les allocations dynamiques durant l'exécution, les interruptions issues du matériel sont gérées à l'aide d'une mémoire de stockage spécifique, appelée pile d'interruption (*interrupt stack*).

La création d'une pile d'interruption s'effectue au sein de la routine MmAllocateIsrStack, qui à l'issue des opérations met à jour le champ IsrStack dans le bloc de contrôle du processeur.

Piles noyau

Tandis que les piles utilisateur occupent généralement un espace confortable (1 Mo), la quantité de mémoire dédiée aux piles noyau est sensiblement inférieure : 12 Ko. Il est de ce fait attendu que le code exécuté en mode noyau emploie avec une parcimonie la plus grande des appels de fonction récursifs, lesquels peuvent à terme entrainer des débordements de pile, et se concentre sur une gestion intelligente des variables locales. Qui plus est, étant donné les piles noyau résident dans l'espace d'adressage du système, l'utilisation qui en est faite a un impact particulièrement significatif.

Si, comme nous venons de le souligner, les algorithmes que le noyau met en oeuvre sont par nature plus itératif que récursif, diverses interactions entre composants mode noyau et mode utilisateur se fondent néanmoins sur une perspective réentrante. Pour offrir une solution à cet égard, Windows fournit un mécanisme permettant d'agrandir et de réduire de manière dynamique la taille de la pile d'exécution des threads au sein du noyau. Ledit mécanisme offre des garanties sur deux fronts : une prise en charge fiable des appels système récursifs, et une utilisation optimale de l'espace d'adressage système.

Les concepteurs de pilotes peuvent en vue d'obtenir des garanties concernant une quantité minimale d'espace de pile prendre appui sur les routines KeExpandKernelStackAndCallout et KeExpandKernelStackAndCalloutEx. Toutes les deux ont un comportement similaire, à ceci près que KeExpandKernelStackAndCalloutEx ajoute des paramètres supplémentaires et peut être sollicitée depuis un IRQL inférieur ou égal à DISPATCH_LEVEL - tandis que KeExpandKernelStackAndCallout peut seulement l'être depuis un niveau inférieur ou égal à APC_LEVEL.

Notifications de statut de la mémoire

Windows offre aux applications et aux composants système la possibilité d'être tenu informé de divers événements concernant la mémoire : taux d'utilisation élevé, niveau de disponibilité important, portions (pages) corrompus, etc. Ces informations peuvent en premier lieu servir à optimiser l'utilisation de la mémoire. Par exemple, dans l'éventualité où la mémoire arrive à saturation, une application peut en conséquence adopter une

attitude parcimonieuse dans sa consommation. Autre exemple : quand la quantité de mémoire noyau disponible à partir du pool paginé est élevée, un pilote peut dès lors favoriser une politique d'allocation optimiste.

L'API Windows permet aux processus mode utilisateur de prendre la mesure du niveau de la mémoire par le biais de deux fonctions : CreateMemoryResourceNotification et QueryMemoryResourceNotification.

Les processus en mode utilisateur peuvent prendre la mesure du niveau de la mémoire physique (mais pas de l'état) via la fonction CreateMemoryResourceNotification, en spécifiant durant l'appel si notification demandée concerne une mémoire faible, ou au contraire une mémoire élevée. Le handle retourné est utilisable dans n'importe laquelle des fonctions d'attente. Quand la quantité de mémoire physique excède ou au contraire devient inférieur à un certain seuil, l'attente se trouve, de facto, satisfaite. Une autre technique consiste à employer la fonction QueryMemoryResourceNotification, laquelle opère en mode non bloquant.

Les pilotes qui vérifient l'état et le niveau de la mémoire le font en s'appuyant sur un ensemble d'objets événement que le noyau entrepose dans un répertoire spécial appelé \KernelObjects. Lorsque le gestionnaire de mémoire rencontre une situation se prêtant à une notification (voir tableau ci-après), il signale l'événement idoine, ce qui pour effet de réveiller tous les threads en attente dudit événement.

Tableau 10.21. Notifications de statut de la mémoire

Nom de l'événement	Configuré à l'état signalé
HighMemoryCondition	Quand la quantité de mémoire physique disponible dépasse un certain seuil.
HighNonPagedPoolCondition	Quand la quantité de mémoire noyau non paginable dépasse un certain seuil.
HighPagedPoolCondition	Quand le système rencontre un dysfonctionnement parmi les structures de pagination, par exemple une page dans l'état Erroné.
LowMemoryCondition	Quand la quantité de mémoire physique disponible tombe sous un certain seuil.
LowNonPagedPoolCondition	Quand la quantité de mémoire noyau non paginable disponible tombe sous un certain seuil.
LowPagedPoolCondition	Quand le système fait face à quantité de mémoire noyau paginable disponible tombe sous un certain seuil.

Dans la configuration par défaut, le système met en évidence les éventuelles anomalies rencontrées au niveau de la mémoire, par exemple une page incorrecte, à l'aide d'un objet spécifiquement conçu à cet égard (\KernelObjects\MemoryErrors). Ce comportement peut être redéfini en ajoutant au registre la clé PageValidationAction sous l'arborescence HKLM\SYSTEM\CurrentControlSet\Session Manager\Memory Management, dont la présence conditionne en cas de dysfonctionnement de la mémoire un effondrement du système. La variable noyau MmPageValidationAction constitue un réplicat de la valeur de registre PageValidationAction.

```
lkd> !object \KernelObjects
Object: ffffc4840fc09870  Type: (ffffda8e17496900) Directory
    ObjectHeader: ffffc4840fc09840 (new version)
    HandleCount: 0  PointerCount: 21
    Directory Object: ffffc4840fc092a0  Name: KernelObjects

    Hash Address          Type                      Name
    ---- -------          ----                      ----
     00  ffffc4840fc316d0 SymbolicLink              MemoryErrors
     02  ffffda8e17475ea0 Event                     LowNonPagedPoolCondition
     04  ffffda8e1aafbc50 Session                   Session1
     05  ffffda8e17475220 Event                     SuperfetchScenarioNotify
         ffffda8e17475320 Event                     SuperfetchParametersChanged
     10  ffffc4840fc31a30 SymbolicLink              PhysicalMemoryChange
     12  ffffc4840fc310a0 SymbolicLink              HighCommitCondition
     13  ffffda8e17492960 Mutant                    BcdSyncMutant
     14  ffffc4840fc31910 SymbolicLink              HighMemoryCondition
         ffffda8e17475b20 Event                     HighNonPagedPoolCondition
     17  ffffda8e17498ea0 Partition                 MemoryPartition0
     21  ffffc4840fc0b060 KeyedEvent                CritSecOutOfMemoryEvent
     22  ffffda8e174755a0 Event                     SystemErrorPortReady
     23  ffffc4840fc31d00 SymbolicLink              MaximumCommitCondition
```

```
     25  ffffc4840fc31c70 SymbolicLink              LowCommitCondition
     26  ffffda8e17475a20 Event                     HighPagedPoolCondition
     28  ffffc4840fc31be0 SymbolicLink              LowMemoryCondition
     32  ffffda8e1aafabd0 Session                   Session0
         ffffda8e174753a0 Event                     LowPagedPoolCondition
     34  ffffda8e17475120 Event                     PrefetchTracesReady
```

Partition mémoire

```
lkd> dt nt!_MI_PARTITION_PAGE_LISTS
   +0x000 FreePagesByColor : [2] Ptr64 _MMPFNLIST
   +0x040 ZeroedPageListHead : _MMPFNLIST
   +0x080 FreePageListHead : _MMPFNLIST
   +0x0c0 StandbyPageListHead : _MMPFNLIST
   +0x100 StandbyPageListByPriority : [8] _MMPFNLIST
   +0x240 ModifiedPageListNoReservation : _MMPFNLIST
   +0x280 ModifiedPageListByReservation : [16] _MMPFNLIST
   +0x500 MappedPageListHead : [16] _MMPFNLIST
   +0x780 BadPageListHead  : _MMPFNLIST
   +0x7c0 EnclavePageListHead : _MMPFNLIST
   +0x7e8 FreePageSlist    : [2] Ptr64 _SLIST_HEADER
   +0x7f8 PageLocationList : [8] Ptr64 _MMPFNLIST
   +0x838 StandbyRepurposedByPriority : [8] Uint4B
   +0x880 TransitionSharedPages : Uint8B
   +0x888 TransitionSharedPagesPeak : [6] Uint8B
   +0x8b8 MappedPageListHeadEvent : [16] _KEVENT
   +0xa38 DecayClusterTimerHeads : [4] _MI_DECAY_TIMER_LINK
   +0xa58 DecayHand        : Uint4B
   +0xa5c StandbyListDiscard : UChar
   +0xa5d FreeListDiscard  : UChar
   +0xa5e LargePfnBitMapsReady : UChar
   +0xa60 LastDecayHandUpdateTime : Uint8B
   +0xa68 LastChanceLdwContext : _MI_LDW_WORK_CONTEXT
   +0xac0 AvailableEventsLock : Uint8B
   +0xac8 AvailablePageWaitStates : [3] _MI_AVAILABLE_PAGE_WAIT_STATES
   +0xb28 MirrorListLocks  : Ptr64 Void
   +0xb40 TransitionPrivatePages : Uint8B
   +0xb48 LargePfnBitMap   : [2] _RTL_BITMAP_EX
   +0xb68 LargePageListHeads : Ptr64 _MI_FREE_LARGE_PAGE_LIST
   +0xb70 LargePageCandidates : [2] _MI_LARGE_PAGE_CANDIDATES
   +0xf80 RebuildLargePageWorkItem : _WORK_QUEUE_ITEM
   +0xfa0 RebuildLargePageActive : UChar
   +0xfa4 LargePageRebuildLock : Int4B
   +0xfa8 LowMemoryThreshold : Uint8B
   +0xfb0 HighMemoryThreshold : Uint8B

lkd> !partition
Partition0 fffff80164465140 MemoryPartition0

lkd> !partition fffff80164465140
PartitionObject @ ffffa7049be77c00 (MemoryPartition0)
_MI_PARTITION 0 @ fffff80164465140
 MemoryRuns: 0000000000000000
 MemoryNodeRuns: ffffa7049be433a0
 TotalHugeIoSpaceRanges: 0 GB
 AvailablePages:         0n416041 ( 1 Gb 601 Mb 164 Kb)
 ResidentAvailablePages: 0n653683 ( 2 Gb 505 Mb 460 Kb)
    0 _MI_NODE_INFORMATION @ ffffea0000007000
   TotalPagesEntireNode:    0xbe955
       Zeroed                        Free
   1GB             0 ( 0)                            0 ( 0)
   2MB             0 ( 0)                            0 ( 0)
   64KB          303 ( 18 Mb 960 Kb)                 0 ( 0)
   4KB         55783 ( 217 Mb 924 Kb)                0 ( 0)
   IoSpace           0 ( 0)                            0 ( 0)
```

```
   Node Free Memory:     ( 236 Mb 860 Kb )
   InUse Memory:         ( 2 Gb 764 Mb 504 Kb )
   TotalNodeMemory:      ( 2 Gb 1001 Mb 340 Kb )
   Node Free IoSpace     ( 0 )

lkd> dtx nt!_MI_PARTITION fffff80164465140
(*((nt!_MI_PARTITION *)0xfffff80164465140))                [Type: _MI_PARTITION]
    [+0x000] Core             [Type: _MI_PARTITION_CORE]
    [+0x1b0] Modwriter        [Type: _MI_PARTITION_MODWRITES]
    [+0x470] Store            [Type: _MI_PARTITION_STORES]
    [+0x500] Segments         [Type: _MI_PARTITION_SEGMENTS]
    [+0x840] PageLists        [Type: _MI_PARTITION_PAGE_LISTS]
    [+0x1c00] Commit           [Type: _MI_PARTITION_COMMIT]
    [+0x1c80] Zeroing          [Type: _MI_PARTITION_ZEROING]
    [+0x1ce8] PageCombine      [Type: _MI_PAGE_COMBINING_SUPPORT]
    [+0x1e78] WorkingSetControl : 0xffffa7049be43430 [Type: void *]
    [+0x1e80] WorkingSetExpansionHead [Type: _MMWORKING_SET_EXPANSION_HEAD]
    [+0x1e90] SessionDetachTimeStamp : 0xfb33c [Type: unsigned long]
    [+0x1ec0] Vp               [Type: _MI_VISIBLE_PARTITION]

lkd> dtx nt!_MI_VISIBLE_PARTITION 0xfffff80164467000
(*((nt!_MI_VISIBLE_PARTITION *)0xfffff80164467000))                [Type: _MI_VISIBLE_PARTITION]
    [+0x000] LowestPhysicalPage : 0x1 [Type: unsigned __int64]
    [+0x008] HighestPhysicalPage : 0x10ffff [Type: unsigned __int64]
    [+0x010] NumberOfPhysicalPages : 0xbe955 [Type: unsigned __int64]
    [+0x018] NumberOfPagingFiles : 0x3 [Type: unsigned long]
    [+0x01c] SystemCacheInitialized : 0x1 [Type: unsigned char]
    [+0x020] PagingFile       [Type: _MMPAGING_FILE * [16]]
    [+0x0c0] AvailablePages   : 0x508ea [Type: unsigned __int64]
    [+0x100] ResidentAvailablePages : 0x9c14b [Type: unsigned __int64]
    [+0x140] PartitionWs      [Type: _MMSUPPORT_INSTANCE [1]]
    [+0x200] PartitionWorkingSetLists [Type: _MMWSL_INSTANCE [1]]
    [+0x228] TotalCommittedPages : 0x83776 [Type: unsigned __int64]
    [+0x240] ModifiedPageListHead [Type: _MMPFNLIST]
    [+0x280] ModifiedNoWritePageListHead [Type: _MMPFNLIST]
    [+0x2a8] TotalCommitLimit : 0xe6955 [Type: unsigned __int64]
    [+0x2b0] TotalPagesForPagingFile : 0x28a1 [Type: unsigned __int64]
    [+0x2b8] VadPhysicalPages : 0x0 [Type: unsigned __int64]
    [+0x2c0] ProcessLockedFilePages : 0xa [Type: unsigned __int64]
    [+0x2c8] SharedCommit     : 0x10250 [Type: unsigned __int64]
    [+0x2d0] SlabAllocatorPages : 0x0 [Type: unsigned __int64]
    [+0x2d8] ChargeCommitmentFailures [Type: unsigned long [4]]
    [+0x2e8] PageFileTraceIndex : 10118 [Type: long]
    [+0x2f0] PageFileTraces   [Type: _MI_PAGEFILE_TRACES [32]]

lkd> dt nt!_MI_VISIBLE_STATE
   +0x000 SessionWsList    : _LIST_ENTRY
   +0x010 SessionIdBitmap  : Ptr64 _RTL_BITMAP
   +0x018 PagedPoolInfo    : _MM_PAGED_POOL_INFO
   +0x030 MaximumNonPagedPoolInPages : Uint8B
   +0x038 SizeOfPagedPoolInPages : Uint8B
   +0x040 SystemPteInfo    : _MI_SYSTEM_PTE_TYPE
   +0x0a0 NonPagedPoolCommit : Uint8B
   +0x0a8 SmallNonPagedPtesCommit : Uint8B
   +0x0b0 BootCommit       : Uint8B
   +0x0b8 MdlPagesAllocated : Uint8B
   +0x0c0 SystemPageTableCommit : Uint8B
   +0x0c8 ProcessCommit    : Uint8B
   +0x0d0 DriverCommit     : Int4B
   +0x0d4 PagingLevels     : UChar
   +0x0d8 PfnDatabaseCommit : Uint8B
   +0x100 SystemWs         : [6] _MMSUPPORT_FULL
   +0x880 SystemCacheShared : _MMSUPPORT_SHARED
   +0x900 AggregateSystemWs : [1] _MMSUPPORT_AGGREGATION
   +0x920 MapCacheFailures : Uint4B
   +0x928 PagefileHashPages : Uint8B
```

```
+0x930 PteHeader        : _SYSPTES_HEADER
+0xa48 SystemVaTypeCount : [15] Uint8B
+0xac0 SystemVaType     : [256] UChar
+0xbc0 SystemVaRegions  : [12] _MI_SYSTEM_VA_ASSIGNMENT
```

TLS (Thread Local Storage)

La table TLS (TlsSlots) représente 64 emplacements de valeurs dépendantes du mot-machine (32 ou 64 bits) contenant des données spécifiques au thread.

AWE (Address Windowing Extensions)

L'accès à plus de 4 gigaoctets (Go) de mémoire sur un système d'exploitation 32 bits nécessite AWE (Address Windowing Extensions) pour gérer la mémoire.

Vérification de la gestion mémoire

Voici quelles options offre Vérificateur de pilote en matière de gestion mémoire :

Pool spécial Force les routines d'allocation de pool à placer les allocations entre des pages invalides, de sorte que toute référence faite en dehors de cet espace (avant ou après) entraine une violation d'accès mode noyau, provoquant donc l'effondrement du système.

Dès lors que l'option Pool spécial est exprimée, Vérificateur de pilotes redirige les demandes d'allocation émanant des pilotes soumis à vérification vers une zone mémoire spécifiquement configurée pour la détection des erreurs de manipulation. Quand un pilote alloue de la mémoire depuis le pool spécial, Windows arrondit l'allocation de façon qu'elle commence sur une frontière de page paire. Comme Vérificateur de pilotes entoure de pages non accessibles la page allouée, toute instruction essayant de lire ou d'écrire au delà de la fin du tampon se verra confronté à une page invalide, le gestionnaire de mémoire soulevant pour l'occasion une violation d'accès mode noyau. Un débogueur attaché au système peut alors prendre le relais en vue de mettre en évidence les causes d'un tel dysfonctionnement.

Vérificateur de pilote prend en charge deux formes de prévention des accès mémoire illégitimes : *buffer overrun* et *buffer underrun*. Dans le premier scénario, à quoi correspond une détection des débordements de fin (overrun), le gestionnaire de mémoire positionne le tampon utilisé par le pilote à la fin de la page allouée et remplit le début de la page avec des motifs spéciaux. Dans le second cas de figure, où Windows met alors en oeuvre la détection de débordement de début (underrun), le système alloue le tampon du pilote au début de la page, et non à la fin.

Dans la configuration par défaut, Vérificateur de pilotes effectue de la détection de débordement de fin et, s'il n'offre pas d'option facilement accessible concernant les débordements de début, donne lieu à ce type de détection dès lors que la valeur de la clé HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\PoolTagOverruns est à 0. (Autre possibilité pour parvenir au même résultat : exécuter l'utilitaire Gflags et sélectionner l'option Verify Start au lieu de Verify End.)

Autre emplacement du Registre Windows qui influent sur le comportement du Vérificateur de pilote : HKLM \SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\PoolTag, à quoi correspond les balises d'allocation que le système emploie pour le pool spécial. Ainsi, quand bien même la configuration active du Vérificateur de pilotes ne l'appelle pas à considérer un pilote particulier, si la balise associée par un pilote à la mémoire qu'il alloue correspond à ce qui est spécifié dans la valeur susmentionnée, les routines d'allocation de pool allouent la mémoire depuis le pool spécial. Il est également envisageable, dans le même ordre d'idée, de paramétrer la valeur PoolTag sur un caractère joker, auquel cas toute la mémoire allouée par les pilotes est issue du pool spécial, pourvu que mémoire virtuelle et physique soient suffisantes.

Suivi de pool Vérifie au moment du déchargement d'un pilote que ledit composant a libéré toutes les allocations de mémoire qu'il a effectué durant son exécution. Si tel n'est pas le cas, Vérificateur de pilotes fait s'effondrer le système en mettant en avant le pilote incriminée (DRIVER_VERIFIER_DETECTED_VIOLATION).

Une autre investigation que mène Vérificateur de pilotes dans ce contexte a trait aux pages restées verrouillées en mémoire après une opération d'E/S. Dans la configuration par défaut, une telle erreur est signalée via le code PROCESS_HAS_LOCKED_PAGES, lequel se voit supplanté par DRIVER_LEFT_ LOCKED_PAGES_IN_PROCESS, plus informatif, quand le suivi de pool est activé.

Forcer la vérification des IRQL Vérifie que le pilote honore la configuration IRQL attendue lors des accès à la mémoire paginable. L'un des dysfonctionnements les plus répandus parmi les composants système de Windows concerne le cas où un pilote sollicite des données ou du code paginables tandis que le processeur (sous entendu le processeur sur lequel le pilote opère) se trouve dans un IRQL élevé. Prennent forme en pareille éventualité deux options : soit les données demandées sont alors physiquement résidentes, auquel cas l'opération s'accomplit sans heurt, soit elles ne le sont pas (leur place est à ce moment dans le fichier de pagination) et doivent par conséquent être ramenées en mémoire vive, ceci se matérialisant sous forme d'un défaut de page. Comme le gestionnaire de mémoire ne peut pas traiter un défaut de page à un niveau IRQL haut (DCP/dispatch ou plus), il s'ensuit un effondrement du système avec le code d'arrêt IRQL_NOT_LESS_OR_EQUAL (c'est à dire que l'IRLQ n'était pas inférieur ou égal au niveau requis).

Pour révéler d'éventuels manquements aux règles établies en ce qui concerne l'IRQL, Vérificateur de pilote force tout le code paginable mode noyau et toutes les données paginables à sortir du working set système chaque fois qu'un pilote soumis à examen augmente l'IRQL. De cette façon, toute instruction qui, en dépit de l'IRQL, suscite un défaut de page se voit immédiatement sanctionnée par un effondrement du système.

Simulation de ressources faibles Contraint le système à faire échouer de manière aléatoire certaines des allocations de mémoire auxquelles procèdent les pilotes vérifiés, cela afin de mesurer leur capacité à apporter une réponse convaincante face à un manque de mémoire noyau.

Passé un certain laps de temps après le démarrage du système (délai émanant de la nécessité de surpasser la phase d'initialisation critique, pendant laquelle un manque de mémoire serait susceptible d'empêcher le chargement d'un pilote), Vérificateur de pilotes donne lieu de manière aléatoire à des appels d'allocation infructueux. Dans l'éventualité où un pilote ne répond pas correctement à une telle erreur, le système s'effondre en identifiant le composant incriminé.

Plusieurs facteurs entrent en compte dans la manière dont Vérificateur de pilotes effectue une simulation de ressources faibles : le délai avant qu'elle ait lieu (point que nous avons abordé plus haut), la probabilité que les appels d'allocation échoue (sauf spécification du contraire, 6%), les applications concernées (toutes par défaut)et les balises de pool affectées.

Vérifications diverses Contrôle les tentatives de libérer la mémoire allouée à des ressources système encore et néanmoins actives.

Memory Descriptor List (MDL)

Une liste de descripteurs mémoire (MDL, Memory Descriptor List) décrit la mémoire physique occupée par un tampon. En interne, Windows a recours à des MDL en de nombreuses circonstances. C'est par ce biais, par exemple, que le gestionnaire d'E/S interagit avec les périphériques qui font de l'accès direct à la mémoire (DMA), donnant alors lieu à une MDL pour chaque tampon d'E/S impliqué lors d'un transfert.

Le tableau qui suit répertorie les fonctions de support qu'exposent gestionnaire de mémoire et gestionnaire d'E/S en matière de MDL.

- *IoAllocateMdl* Alloue une MDL conforme aux exigences spécifiées en paramètre, incluant adresse virtuelle et taille du tampon.
- *IoBuildPartialMdl* Crée une MDL en tant que sous-ensemble d'une MDL existante. Cette routine est généralement employée pour fractionner un transfert en plusieurs petits.
- *IoFreeMdl* Libère la mémoire allouée pour une MDL.

- MmAdvanceMdl
- MmAllocateMappingAddress
- MmAllocatePagesForMdl
- MmAllocatePagesForMdlEx
- MmBuildMdlForNonPagedPool
- *MmGetSystemAddressForMdlSafe* Voir attribut MappedSystemVA de la structure MDL.
- MmPrepareMdlForReuse
- *MmProbeAndLockPages* S'assure que les pages décrites par une MDL autorisent le type d'accès demandé et, dans l'affirmative, verrouille ces pages en mémoire.
- *MmUnlockPages* Déverrouille les pages décrites par une MDL.
- *MmGetMdlVirtualAddress* Retourne l'adresse virtuelle du tampon représenté par une MDL (adresse qui, il est important ne le noter, peut n'être pas valide dans le contexte du thread courant).
- *MmGetMdlByteCount* Retourne la taille, en octets, du tampon auquel se rapporte une MDL.
- *MmMapLockedPages* Mappe les pages physiques décrites par une MDL et retourne l'adresse virtuelle de la région correspondante. Voir attribut StartVa de la structure MDL.
- *MmMapLockedPagesSpecifyCache* Fonctionnement similaire à MmMapLockedPages, à ceci près que la routine permet à l'appelant de spécifier le type de cache à utiliser. Voir attribut StartVa de la structure MDL.

PTE système

Les entrées de table de pages (PTE, Page Table Entries) servent à mapper dynamiquement des pages système telles que espace d'E/S, piles noyau et listes de descripteurs de mémoire.

Pour connaitre le nombre de PTE système disponibles, examinez la valeur de compteur Mémoire: Entrées libres en table des pages système dans la console Performances.

Chapitre 11. Portable Executable

Portable Executable est un format binaire employé dans Microsoft Windows pour l'enregistrement et l'organisation de nombreuses formes de contenu actif, y compris les applications et les pilotes. En tant que nomenclature une véritable leçon sur la manière dont le système d'exploitation gère programmes et processus, PE établit une structure formelle qui fait écho à de nombreux composants et pléthore de mécanismes internes de Windows.

Historique

En matière de structuration des formats binaire, chaque système d'exploitation voit ses besoins évoluer en fonction de l'infrastructure le supportant. Windows, par ses origines conçus pour fonctionner sur des systèmes à base de x86, à été prévu pour supporter d'autres architectures et plateformes matérielles, tels la série des Motorola 68000, des Intel 64, ou encore même des Power PC. Aussi, conséquence directe des objectifs de conception du projet (portabilité), la création du format PE résulte de la volonté de Microsoft d'élaborer une structure de fichier commune, fonctionnant sur différentes architectures matérielles, et s'adaptant facilement à de nouvelles - en peu voire pas de modifications.

Microsoft migra vers le format PE avec l'introduction de la première version publique du noyau, soit avec Windows NT 3.1, mis sur le marché en 1993. L'ancêtre de PE, New Executable (NE), s'il convenait pour une utilisation entre les mains du grand public, souffrait de diverses restrictions - mémoire surtout - qui le rendaient incompatible avec les entreprises. PE fut donc développé dans l'optique de combler ces lacunes, et permettre d'exploiter au mieux les ressources machine. Toutes les versions de Windows depuis la version 3.1 supportent ce format.

Le schéma PE est à la base de nombreux composants exécutables de Windows : programmes 32 et 64 bits (.exe), bibliothèques logicielle (.dll), contrôles ActiveX et OLE (.ocx), pilotes de périphériques (.sys), enfichables du panneau de configuration (.cpl), écrans de veille (.scr), et quelques autres encore. Il est de plus intègré à une variété de dispositifs informatiques, et constitue par exemple le format standard mis en avant dans les environnements EFI. Autre exemple : la console Xbox, qui utilise une version quelque peu modifiée de Portable Executable.

Vue d'ensemble du format

Au plus haut niveau d'abstraction, tout fichier PE peut être vu comme une succession linéaire de sections de données. On peut dans cette perspective délimiter quatre sous-ensembles généraux.

- *Segment DOS* Garanti la conformité avec les standards et les conditions d'exécution des programmes sous DOS, longtemps resté à la base des des systèmes d'exploitation grand public de Microsoft. Cette portion du fichier n'a à l'heure actuelle plus grande utilité. Elle reste toutefois présente par égard à la specification PE telle qu'elle fut conçu a l'origine.
- *Entêtes* Procure au système d'exploitation, plus particulièrement au chargeur d'images, l'ensemble des informations nécessaires pour charger le fichier en mémoire et/ou confectionner l'espace d'adressage qui en résulte. Les propriétés mentionnées ici incluent, par exemple, des informations d'alignement, des caractéristiques de contrôle, divers états contextuels, et ainsi de suite. Plus important encore, les entêtes représentatives des sections, qui établissent les modalités de traitement et de suivi de chacune des sections devant être chargé en mémoire.
- *Sections* Regroupe dans un même sous-ensemble des données ayant une fonction commune.

Structure d'un fichier PE

Entête MS-DOS

Afin de rester conciliable avec l'environnement Microsoft DOS, chaque fichier PE encapsule en son tout début une structure compatible avec ledit système, lui permettant en l'occurence de reconnaitre le fichier comme un exécutable valide, et incidemment, de pouvoir exécuter correctement le micro-programme mode réel que ce dernier héberge. (Nous reviendrons par la suite plus en détail sur cette orientation technique.)

Les exécutables prévus pour MS-DOS peuvent fonctionner sur les gammes 9x de Windows, elles mêmes fondées sur DOS. Les versions 32 bits de Windows prennent en charge ces programmes par l'intermédiaire de la machine DOS virtuelle. Les déclinaisons 64 bits de Windows sont dépourvus de toute forme de compatibilité avec MS-DOS.

Entête DOS (vue disque).

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 F8 00 00 00  ............ø...
```

Du fait de la tombée en désuétude de MS-DOS, Windows fait usage de seulement deux attributs standards fournis au niveau de l'entête DOS.

- *Nombre magique (e_magic)* Suite d'octets utilisée pour l'identification et la caractérisation de tout exécutable DOS. Exemple (offset 0-1) : 4D 5A, soit la chaîne de caractères ASCII MZ - laquelle, à titre informatif, se réfère aux initiales de Mark Zbikowski, qui fut la personne aux commandes du format des fichiers exécutables sous DOS. Lors de l'ouverture d'une image à exécuter, Windows charge au préalable de toute autre mesure la zone cible pour y rechercher une signature valide.
- *Offset de l'entête PE (e_lfanew)* Détermine l'endroit où se trouve l'entête PE. La valeur est exprimée telle un déplacement par rapport au début du fichier - et non relativement à la mémoire, ce qui n'aurait pas grand sens à ce stade. Exemple (offset 60-63) : F8 00 00 00, ce qui signifie que l'entête PE commence dans cet exemple au 248ème octets (0xF8) du fichier.

```
0:000> dt ntdll!_IMAGE_DOS_HEADER
   +0x000 e_magic          : Uint2B
   +0x002 e_cblp           : Uint2B
   +0x004 e_cp             : Uint2B
   +0x006 e_crlc           : Uint2B
   +0x008 e_cparhdr        : Uint2B
   +0x00a e_minalloc       : Uint2B
   +0x00c e_maxalloc       : Uint2B
   +0x00e e_ss             : Uint2B
   +0x010 e_sp             : Uint2B
   +0x012 e_csum           : Uint2B
   +0x014 e_ip             : Uint2B
   +0x016 e_cs             : Uint2B
   +0x018 e_lfarlc         : Uint2B
   +0x01a e_ovno           : Uint2B
   +0x01c e_res            : [4] Uint2B
   +0x024 e_oemid          : Uint2B
   +0x026 e_oeminfo        : Uint2B
   +0x028 e_res2           : [10] Uint2B
   +0x03c e_lfanew         : Int4B
```

Micro-programme mode réel

Accolé éventuellement à l'entête MS-DOS, le micro-programme mode réel (*real mode stub program*) sert d'abri à du code machine 16 bits qui, lorsqu'il est exécuté par DOS, affiche généralement un court message avant de se terminer. Typiquement, le message est là afin d'indiquer une erreur quant au choix de la plateforme, à savoir que le programme n'est pas fait pour une utilisation sous MS-DOS, mais requiert Windows pour fonctionner. Ledit message s'apparente souvent au texte "This program cannot be run in DOS mode", ou "This program must be run under Win32".

Micro-programme mode réel (vue disque).

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
```

```
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode```$```...
```

Entête PE

Offset (h)	Taille (o)	IMAGE_FILE_HEADER
0	2	Machine
2	2	NumberOfSections
4	4	TimeDateStamp
8	4	PointerToSymbolTable
12	4	NumberOfSymbols
16	2	SizeOfOptionalHeader
18	2	Characteristics

- *Plateforme d'accueil (Machine)* Apporte des précisions en ce qui concerne la configuration matérielle pour laquelle le programme est destiné, par exemple machines à base de processeurs compatibles 386.
- *Nombre de sections (NumberOfSections)* Nombre d'entrées contenues dans la table des sections ; par extension, nombre de sections que le module renferme.
- *Nombre de Symboles NumberOfSymbols* Nombre d'entrées définies dans la table des symboles.
- *Pointeur vers la table des symboles PointerToSymbolTable*
- *Taille de l'entête optionnelle (SizeOfOptionalHeader)* Donne le nombre d'octets qui constituent l'entête optionnelle, à quoi correspond sizeof(IMAGE_OPTIONAL_HEADER).
- *Cachet horodaté (TimeDateStamp)* Enregistre l'heure et la date à laquelle le fichier a été généré. Outre sa dimension temporelle, cet attribut peut se montrer utile pour pour distinguer plusieurs versions d'un même fichier.
- *Caractéristiques* (*Characteristics*) Propriétés générales du fichier.

Types de machine

Le champ Machine donne lieu à un ensemble de caractéristiques, répertoriées ci-dessous, qui spécifient l'architecture matérielle pour laquelle le fichier image a été pensé (comprendre compilé). Le programme ne peut être utilisé que sur un système compatible avec l'architecture décrite. Par exemple, une valeur Machine égale à 0x14C indique le programme a été compilé (et lié) avec en perspective les machines à base de processeurs compatibles 386. Sont concevables à cet égard les attributs que voici.

Valeur	Constante	Description
0x014c	IMAGE_FILE_MACHINE_I86	Intel 386 et compatible
0x014d	IMAGE_FILE_MACHINE_I860	Intel i860 (32-bit)
0x0162	IMAGE_FILE_MACHINE_R3000	MIPS R3000 (32-bit)
0x0166	IMAGE_FILE_MACHINE_R4000	MIPS R4000 (64-bit)
0x0168	IMAGE_FILE_MACHINE_R10000	MIPS R10000 (64-bit)
0x0169	IMAGE_FILE_MACHINE_WCEMIPSV2	MIPS Windows Compact Edition v2
0x0183	-	DEC Alpha AXP
0x0184	IMAGE_FILE_MACHINE_ALPHA	Digital Equipment Corporation (DEC) Alpha (32-bit)
0x01a2	IMAGE_FILE_MACHINE_SH3	Hitachi SH-3 little-endian (32-bit)
0x01a3	IMAGE_FILE_MACHINE_SH3DSP	Hitachi SH-3 DSP (32-bit)
0x01a4	IMAGE_FILE_MACHINE_SH3E	SH3E little-endian
0x01a6	IMAGE_FILE_MACHINE_SH4	Hitachi SH-4 little-endian (32-bit)

0x01a8	IMAGE_FILE_MACHINE_SH5	Hitachi SH-5 (64-bit)
0x01c0	IMAGE_FILE_MACHINE_ARM	ARM little endian (32-bit)
0x01c2	IMAGE_FILE_MACHINE_THUMB	-
0x01c4	IMAGE_FILE_MACHINE_ARMV7	-
0x01d3	IMAGE_FILE_MACHINE_AM33	Matsushita MN10300/AM33
0x01f0	IMAGE_FILE_MACHINE_POWERPC	IBM PowerPC Little-Endian
0x01f1	IMAGE_FILE_MACHINE_POWERPCFP	-
0x01F2	IMAGE_FILE_MACHINE_POWERPCBE	Power PC Big Endian
0x0200	IMAGE_FILE_MACHINE_IA64	Intel Itanium (64-bit)
0x0266	IMAGE_FILE_MACHINE_MIPS16	MIPS16 (16-bit)
0x0268	IMAGE_FILE_MACHINE_M68K	Motorola 68000 (32-bit)
0x0284	IMAGE_FILE_MACHINE_ALPHA64	DEC Alpha AXP (64-bit) (IMAGE_FILE_-MACHINE_AXP64)
0x0366	IMAGE_FILE_MACHINE_MIPSFPU	-
0x0466	IMAGE_FILE_MACHINE_MIPSFPU16	-
0x0520	IMAGE_FILE_MACHINE_TRICORE	Infineon AUDO (32-bit)
0xaa64	IMAGE_FILE_MACHINE_ARM64	ARM8+ (64-bit)
0x0cef	IMAGE_FILE_MACHINE_CEF	-
0x0EBC	IMAGE_FILE_MACHINE_EBC	EFI byte code (32-bit)
0x8664	IMAGE_FILE_MACHINE_AMD64	AMD (64-bit)
0x9041	IMAGE_FILE_MACHINE_M32R	Mitsubishi M32R little endian (32-bit)

Caractéristiques

Le champ Caractéristiques contient différents modificateurs qui indiquent les attributs du fichier image.

- Une valeur égale à IMAGE_FILE_RELOCS_STRIPPED (0x1) signifie que le fichier ne contient pas pas d'informations de relogement et indique par conséquent au système d'exploitation de charger le programme uniquement à son adresse de base préférée. Si l'adresse de base préférée est introuvable, le système d'exploitation ne charge pas le fichier.
- Une valeur égale à IMAGE_FILE_EXECUTABLE_IMAGE (0x2) indique que toutes les références externes sollicitées depuis le module ont été résolues (par le programme d'édition des liens), de sorte à pouvoir considérer le fichier résultant comme exécutable.
- Une valeur égale à IMAGE_FILE_LINE_NUMS_STRIPPED (0x4) indique que les numéros de ligne parmi les informations de débogage COFF ont été supprimés.
- Une valeur égale à IMAGE_FILE_LOCAL_SYMS_STRIPPED (0x8) indique les informations concernant les symboles locaux ont été supprimés.
- Une valeur égale à IMAGE_FILE_AGGRESSIVE_WS_TRIM (0x10) force le système à élaguer l'ensemble du travail du processus plus souvent qu'il ne le fait d'ordinaire. Obsolète dans les versions actuelles de Microsoft Windows.
- Une valeur égale à IMAGE_FILE_LARGE_ADDRESS_AWARE (0x20) indique que Le processus peut croitre au-delà de 2 Go d'espace d'adressage.
- Une valeur égale à IMAGE_FILE_BYTES_REVERSED_LO (0x80) indique l'usage d'une représentation des données de type little endian.
- Une valeur égale à IMAGE_FILE_32BIT_MACHINE (0x100) indique que la plateforme matérielle prévue pour faire fonctionner le programme se base sur des des mots de 32 bits.
- Une valeur égale à IMAGE_FILE_DEBUG_STRIPPED (0x200) signifie que les informations de débogage ont été supprimées.

- IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP (0x400) Si l'image réside sur un média amovible, copier et exécuter depuis le fichier d'échange.
- IMAGE_FILE_NET_RUN_FROM_SWAP (0x800) Si l'image réside sur un partage réseau, copier et exécuter depuis le fichier d'échange.
- IMAGE_FILE_SYSTEM (0x1000) Fichier système, par opposition à un programme utilisateur.
- IMAGE_FILE_DLL (0x2000) Le fichier est une DLL.
- IMAGE_FILE_UP_SYSTEM_ONLY (0x4000) L'image ne peut être exécutée que sur un seul processeur.
- IMAGE_FILE_BYTES_REVERSED_HI (0x8000) Indique l'usage d'une représentation des données de type big endian.

Entête optionnelle

Chaque fichier image est pourvu d'un entête facultatif dont le rôle est d'alimenter le chargeur de programmes en informations, dont le sous-système pour lequel l'image est prévu, les options d'alignement, ou encore les quantités de mémoire réservées ou engagées pour la pile et le tas.

L'entête optionnelle l'est uniquement en ce sens que certains fichiers, en particulier les fichiers objet, n'en ont pas. Pour les fichiers image, cet entête est strictement nécessaire, faisant valoir auprès du chargeur comment gérer l'image à l'intérieur d'un processus. Un fichier objet peut éventuellement posséder un entête, mais en général, ce dernier n'a pas de fonctions utiles, sauf augmenter la taille du fichier résultant.

Offset 32/64	Taille 32/64	Nom
0	2	Magic
2	1	MajorLinkerVersion
3	1	MinorLinkerVersion
4	4	SizeOfCode
8	4	SizeOfInitializedData
12	4	SizeOfUninitializedData
16	4	AddressOfEntryPoint
20	4	BaseOfCode
24/--	4/--	BaseOfData
28/24	4/8	ImageBase
32	4	SectionAlignment
36	4	FileAlignment
40	2	MajorOperatingSystemVersion
42	2	MinorOperatingSystemVersion
44	2	MajorImageVersion
46	2	MinorImageVersion
48	2	MajorSubsystemVersion
50	2	MinorSubsystemVersion
52	2	Win32VersionValue
56	4	SizeOfImage
60	4	SizeOfHeaders
64	4	CheckSum
68	2	Subsystem
70	2	DllCharacteristics
72	4/8	SizeOfStackReserve
76/80	4/8	SizeOfStackCommit
80/88	4/8	SizeOfHeapReserve

Offset 32/64	Taille 32/64	Nom
84/96	4/8	SizeOfHeapCommit
88/104	4	LoaderFlags
92/108	4	NumberOfRvaAndSizes
96/112	128	DataDirectory

- *Base du code (BaseOfCode)* Adresse virtuelle relative du début du code. Les sections qui renferment du code viennent généralement avant les sections de données et après l'entête PE en mémoire. Dans une très large majorité des cas, les environnements de programmation Microsoft assignent à cette variable la valeur 0x1000.
- *Base des données (BaseOfData)* Adresse virtuelle relative du début des données. Notez que cet attribut n'est présent que la version 32 bits de l'en-tête facultatif.
- *Somme de contrôle (CheckSum)* Dispositif cryptographique permettant de vérifier l'intégrité de l'image. Sont concernées les pilotes, les DLL d'amorçage et celles chargées dans les processus Windows critiques. (Pour des raisons de performance, Windows ne contrôle pas l'intégrité de toutes les images.) L'algorithme sous-jacent est mis en oeuvre dans la DLL Imagehlp.
- *Version majeure de l'image (MajorImageVersion)*
- *Version mineure de l'image (MinorImageVersion)*
- *MajorSubsystemVersion* Voir fonction GetProcessVersion ; service NtQueryInformationProcess (ProcessInformationClass = ProcessBasicInformation)
- *MinorSubsystemVersion* Voir fonction GetProcessVersion ; service NtQueryInformationProcess (ProcessInformationClass = ProcessBasicInformation)
- *Version majeure du système (MajorOperatingSystemVersion)* Partie d'un système de numérotation (l'autre partie correspondant à l'attribut MinorOperatingSystemVersion) servant à définir quelle version de Windows est requise pour exécuter l'image.
- *Version mineure du système (MinorOperatingSystemVersion)* Partie d'un système de numérotation (l'autre partie correspondant à l'attribut MajorOperatingSystemVersion) servant à définir quelle version de Windows est requise pour exécuter l'image.
- *Taille engagée du tas (SizeOfHeapCommit)* Quantité de mémoire initialement engagée pour le tas du processus. La valeur par défaut correspond à une page. Pour plus d'informations ce qui concerne la différence entre mémoire réservée et mémoire engagée, consultez la section \<<reservation-et-engagement-de-pages> \>.
- *Taille réservée du tas (SizeOfHeapReserve)* Quantité de mémoire virtuelle initialement réservée pour le tas par défaut du processus. Pour plus d'informations ce qui concerne la différence entre mémoire réservée et mémoire engagée, consultez la section \<<reservation-et-engagement-de-pages>\>.
- **Taille de l'image (SizeOfImage)** Taille totale, en octets, de l'image du fichier une fois chargée en mémoire dans l'espace d'adressage du processus. Cette taille inclut toutes les sections du fichier PE et est alignée sur la valeur `SectionAlignment` définie dans l'en-tête du fichier.
- **Alignement de niveau fichier (FileAlignment)** Taille minimale des blocs de données utilisés pour stocker les sections dans le fichier sur disque (hors cadre de l'exécution, donc). La valeur couramment définie à ce égard correspond à 512 (0x200) octets, soit la taille d'un secteur de disque.
- **Adresse de base de l'image (ImageBase)** Adresse préférentielle utilisée lors du chargement (mappage) du fichier dans l'espace d'adressage virtuel d'un processus. La valeur par défaut pour les DLL est 0x10000000, pour les applications 0x400000. Elle doit dans tous les cas être un multiple de 64 kilo-octets - à quoi correspond la granularité d'allocation naturelle de Microsoft Windows. Ce champ est particulièrement important puisque les adresses virtuelles relatives le sont par rapport à cette adresse.

- **Architecture cible (*Magic*)** Architecture pour laquelle le fichier a été prévu. C'est en fonction de ce champ que Windows sait s'il doit exécuter le fichier en tant que programme 32 ou 64 bits. Les valeurs autorisées pour cet attribut sont IMAGE_NT_OPTIONAL_HDR32_MAGIC (0x10b) pour un exécutable 32 bits, IMAGE_NT_OPTIONAL_HDR64_MAGIC (0x20b) pour un exécutable 64 bits, et IMAGE_ROM_OPTIONAL_HDR_MAGIC (0x107) pour une image ROM.
- **Alignement de niveau section (SectionAlignment)** Taille minimale qu'une section peut occuper en mémoire. Par défaut, cette valeur correspond à la taille d'une page telle que la définit l'architecture matérielle, à savoir 4096 (0x1000) octets. Le chargeur Windows ajuste l'adresse de chaque section en la positionnant sur un multiple de cet attribut.
- *Taille des entêtes SizeOfHeaders* Taille cumulée de toutes les entêtes, y compris l'entête DOS, l'entête PE et chaque entête de section. Ce champ doit être multiple de FileAlignment.
- **Sous-système d'environnement (Subsystem)** Sous-système d'environnement requis pour assurer le support du module. Chacun des valeurs envisageables à ce niveau correspond à type particulier d'application : graphique, console, pilote EFI, environnement embarqué, etc. Pour en savoir plus, voir section Sous-système.
- *Numéro de version majeure (MajorLinkerVersion)* Octet mis à la disposition de l'application afin de lui permettre de mettre en avant, par exemple, un numéro de version interne. Voir commutateur /VERSION de dumpbin.
- *Numéro de version mineure (MinorLinkerVersion)* Octet mis à la disposition de l'application afin de lui permettre de mettre en avant, par exemple, un numéro de version interne. Voir commutateur /VERSION de dumpbin.
- *Taille du code (SizeOfCode)* Taille totale occupée par l'ensemble des sections qui renferment du code machine exécutable. Comme la plupart des fichiers ne comportent qu'une seule section de code, ce champ correspond généralement à la taille de la section .code ou .text.
- *Taille des données initialisées* (*SizeOfInitializedData*) Taille totale occupée par les sections contenant des données initialisées (variables globales, constantes etc...). Correspond en principe à la taille de la section .data.
- *Taille des données non initialisées* (*SizeOfUninitializedData*) Taille totale occupée par les sections présentant des données non initialisées. Il s'agit habituellement de la taille de la section .bss ou .rdata.
- *Nombre d'entrées de la table des répertoires (NumberOfRvaAndSizes)* Compte le nombre d'entrées du tableau DataDirectory, situé à la toute fin de l'en-tête optionnelle. Cette valeur est toujours définie à 16 par les outils actuels.
- **Adresse de point d'entrée (*AddressOfEntryPoint*)** Détermine où se situe la première instruction à exécuter après que le programme ait été chargé en mémoire. Correspond dans le cas d'une application mode utilisateur à l'adresse de sa fonction de point d'entrée (WinMain), dans le cas d'un pilote à l'adresse de sa routine d'initialisation (DriverEntry). Pour les bibliothèques, ce champ est facultatif et peut valoir zéro.
- **Caractéristiques (DllCharacteristics)** Ensemble de propriétés relatives au comportement et à la sécurité de l'application.
- *Table des répertoires* (*DataDirectory*) Structure de données utilisées par le système d'exploitation comme moyen commode d'accéder rapidement à quelques-unes des sections standards d'un binaire. Elle est constitué d'un groupe de 16 sous-parties contenant chacune 2 double mots (structure IMAGE_DATA_DIRECTORY), l'un donnant l'adresse virtuelle, l'autre la taille, d'une section spécifique, appelée dans ce contexte repertoire. Chaque répertoire dispose d'une entrée dans la table, y compris s'il n'a pas d'existence concrète - les champs correspondants sont nuls.
- **Taille de pile réservée (SizeOfStackReserve)** Limite supérieure théorique de la pile consacrée à un thread. Contrairement à une allocation immédiate et effective de mémoire, cette portion d'espace d'adressage est simplement marquée comme réservée au sein du processus, permettant de la sorte au système d'anticiper

une expansion éventuelle de la pile sans pour autant consommer immédiatement les ressources physiques correspondantes.

- **Taille de pile engagée (SizeOfStackCommit)** Quantité de mémoire effectivement allouée et mise à disposition du thread au moment de sa création. Contrairement à l'espace réservé, qui constitue une enveloppe théorique, la mémoire engagée est immédiatement utilisable par le thread et pourra, si nécessaire, être étendue progressivement par le système à mesure que la pile croît, jusqu'à atteindre la limite fixée par la taille réservée (attribut `SizeOfStackReserve`).

- **Indicateur de version Win32 (Win32VersionValue)** Valeur censée identifier la version de l'environnement d'exécution Win32 requise par le programme. Hérité des spécifications originelles du format PE, cet attribut n'a en pratique jamais été exploité par le système, et demeure aujourd'hui conservé uniquement pour des raisons de compatibilité structurelle.

TRAVAUX PRATIQUES: Visualisation des informations d'image d'un module. Pour afficher des détaillées concernant un module, commencez par en repérer un qui soit chargé en mémoire, ce par le biais de la commande informations La commande lm.

```
lkd> !lmi fffff800`bc489000
Loaded Module Info: [fffff800`bc489000]
         Module: ntkrnlmp
   Base Address: fffff800bc489000
     Image Name: ntkrnlmp.exe
   Machine Type: 34404 (X64)
     Time Stamp: 5165e551 Thu Apr 11 00:18:57 2013
           Size: 74c000
       CheckSum: 6aef94
Characteristics: 22  perf
Debug Data Dirs: Type  Size     VA  Pointer
             CODEVIEW    25, 201714,  200d14 RSDS - GUID: {7B3C9BFC-DF66-43AB-AACE-89E4C9C33381}
               Age: 2, Pdb: ntkrnlmp.pdb
              CLSID     8, 20170c,  200d0c [Data not mapped]
     Image Type: MEMORY   - Image read successfully from loaded memory.
    Symbol Type: PDB      - Symbols loaded successfully from symbol server.
                 c:\websymbols\ntkrnlmp.pdb\7B3C9BFCDF6643ABAACE89E4C9C333812\ntkrnlmp.pdb
    Load Report: public symbols , not source indexed
                 c:\websymbols\ntkrnlmp.pdb\7B3C9BFCDF6643ABAACE89E4C9C333812\ntkrnlmp.pdb

lkd> !dh fffff800`bc489000 -f

File Type: EXECUTABLE IMAGE
FILE HEADER VALUES
    8664 machine (X64)
      16 number of sections
5165E551 time date stamp Thu Apr 11 00:18:57 2013

       0 file pointer to symbol table
       0 number of symbols
      F0 size of optional header
      22 characteristics
            Executable
            App can handle >2gb addresses

OPTIONAL HEADER VALUES
     20B magic #
   10.10 linker version
  5F4400 size of code
   B4400 size of initialized data
    4C00 size of uninitialized data
  36A1F0 address of entry point
    1000 base of code
         ```- new ```-
0000000140000000 image base
```
```

```
     1000 section alignment
      200 file alignment
        1 subsystem (Native)
     6.02 operating system version
     6.02 image version
     6.02 subsystem version
   74C000 size of image
      600 size of headers
   6AEF94 checksum
 0000000000080000 size of stack reserve
 0000000000002000 size of stack commit
 0000000000100000 size of heap reserve
 0000000000001000 size of heap commit
        0  DLL characteristics
   672000 [   12B65] address [size] of Export Directory
   34E688 [      DC] address [size] of Import Directory
   714000 [   331E0] address [size] of Resource Directory
   30D000 [   4095C] address [size] of Exception Directory
   6A7E00 [    2108] address [size] of Security Directory
   748000 [    3B60] address [size] of Base Relocation Directory
   2016D4 [      38] address [size] of Debug Directory
        0 [       0] address [size] of Description Directory
        0 [       0] address [size] of Special Directory
        0 [       0] address [size] of Thread Storage Directory
   158E00 [      70] address [size] of Load Configuration Directory
        0 [       0] address [size] of Bound Import Directory
   34E000 [     688] address [size] of Import Address Table Directory
        0 [       0] address [size] of Delay Import Directory
        0 [       0] address [size] of COR20 Header Directory
        0 [       0] address [size] of Reserved Directory
\*\*\*\*
.TRAVAUX PRATIQUES:Visualisation du segment DOS

lkd> lm m nt
start             end                 module name
fffff800`bc489000 fffff800`bcbd5000   nt         (pdb symbols)          c:\websymbols\ntkrnlmp.pdb
\7B3C9BFCDF6643ABAACE89E4C9C333812\ntkrnlmp.pdb

lkd> dt nt!_IMAGE_DOS_HEADER e_lfanew fffff800`bc489000
   +0x03c e_lfanew : 0n248

lkd> ur fffff800`bc489000+3c+4 fffff800`bc489000+0n248
GetContextState failed, 0x80004001
nt!`string' <PERF> (nt+0x40):
fffff800`bc489040 0e              push    cs
nt!`string' <PERF> (nt+0x41):
fffff800`bc489041 1f              pop     ds
nt!`string' <PERF> (nt+0x42):
fffff800`bc489042 ba0e00          mov     dx,0Eh
nt!`string' <PERF> (nt+0x45):
fffff800`bc489045 b409            mov     ah,9
nt!`string' <PERF> (nt+0x47):
fffff800`bc489047 cd21            int     21h
nt!`string' <PERF> (nt+0x49):
fffff800`bc489049 b8014c          mov     ax,4C01h
nt!`string' <PERF> (nt+0x4c):
fffff800`bc48904c cd21            int     21h

lkd> da fffff800`bc48904e
fffff800`bc48904e  "This program cannot be run in DO"
fffff800`bc48906e  "S mode```$"
```

DllCharacteristics

| 0x0040 | IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE - Compatibilité ASLR | 0x0080 | IMAGE_DLLCHARACTERISTICS_FORCE_INTEGRITY | 0x0100 | IMAGE_DLLCHARACTERISTICS_NX_COMPAT | 0x0200 | IMAGE_DLLCHARACTERISTICS_NO_ISOLATION | 0x0400 | IMAGE_DLLCHARACTERISTICS_NO_SEH | Désactive la gestion structurée des exceptions. | 0x0800 | IMAGE_DLLCHARACTERISTICS_NO_BIND - Ne pas lier cette image. | 0x2000 | IMAGE_DLLCHARACTERISTICS_WDM_DRIVER | Indique que le module est un pilote Windows Driver Model (WDM). | 0x8000 | IMAGE_DLLCHARACTERISTICS_TERMINAL_SERVER_AWARE

Sous-système

Les valeurs attribuées au niveau de l'attribut Sous-système (champ `Subsystem`) de l'en-tête facultatif (`IMAGE_OPTIONAL_HEADER`) déterminent le sous-système d'exécution attendu en vue d'assurer une prise pour le module concerné, qu'il s'agisse d'une application graphique, d'un programme à interface caractère, d'un pilote EFI, ou d'un environnement spécifique tel Windows CE, Xbox, ou de configurations historiques (OS/2, POSIX).

Constante	Valeur	Description
IMAGE_SUBSYSTEM_-UNKNOWN	0	Sous-système inconnu. S'ensuit qu'aucune prise en charge est envisagable
IMAGE_SUBSYSTEM_NATIVE	1	Pilotes de périphérique et processus Windows natifs
IMAGE_SUBSYSTEM_-WINDOWS_GUI	2	Sous-système d'interface graphique utilisateur Windows
IMAGE_SUBSYSTEM_-WINDOWS_CUI	3	Sous-système de caractères Windows
IMAGE_SUBSYSTEM_OS2_CUI	5	Sous-système de caractères OS/2
IMAGE_SUBSYSTEM_POSIX_CUI	7	Sous-système de caractères Posix
IMAGE_SUBSYSTEM_NATIVE_-WINDOWS	8	Pilote Win9x natif
IMAGE_SUBSYSTEM_-WINDOWS_CE_GUI	9	Windows CE, une variation de Windows spécifique aux systèmes embarqués et autres systèmes minimalistes.
IMAGE_SUBSYSTEM_EFI_-APPLICATION	10	Application EFI
IMAGE_SUBSYSTEM_EFI_-BOOT_SERVICE_DRIVER	11	Pilote EFI, phase de démarrage
IMAGE_SUBSYSTEM_EFI_-RUNTIME_DRIVER	12	Pilote EFI, exécution normale
IMAGE_SUBSYSTEM_EFI_ROM	13	Module EFI stocké en ROM
IMAGE_SUBSYSTEM_XBOX	14	Xbox
IMAGE_SUBSYSTEM_-WINDOWS_BOOT_-APPLICATION	16	Application de démarrage Windows

Répertoires

Pour faciliter l'accès à quelques-unes des sous-parties de l'espace d'adressage mode utilisateur d'un processus, le chargeur d'image enregistre sur le sujet un certain nombre d'éléments et d'informations, dits répertoires de données. La position et la taille des données de ces répertoires sont spécifiés par l'intermédiaire de l'attribut DataDirectory de l'entête facultative, lequel se présente en interne sous la forme d'une série de structures IMAGE_DATA_DIRECTORY. typedef struct _IMAGE_DATA_DIRECTORY { DWORD VirtualAddress; DWORD Size; } IMAGE_DATA_DIRECTORY,*PIMAGE_DATA_DIRECTORY;

Le tableau qui suit montre comment s'enchainent les informations au sein de la table des répertoires.

Position	Nom	Description
00	IMAGE_DIRECTORY_ENTRY_EXPORT	Table des exportations

01	IMAGE_DIRECTORY_ENTRY_IMPORT	Table des importations
02	IMAGE_DIRECTORY_ENTRY_RESOURCE	Table des ressources
03	IMAGE_DIRECTORY_ENTRY_EXCEPTION	Table des exceptions
04	IMAGE_DIRECTORY_ENTRY_SECURITY	Table des certificats
05	IMAGE_DIRECTORY_ENTRY_BASERELOC	Table des relocations
06	IMAGE_DIRECTORY_ENTRY_DEBUG	Informations de débogage
07	IMAGE_DIRECTORY_ENTRY_-ARCHITECTURE	Données spécifiques à l'architecture
08	IMAGE_DIRECTORY_ENTRY_GLOBALPTR	Pointeurs globaux
09	IMAGE_DIRECTORY_ENTRY_TLS	Table de stockage de niveau thread
10	IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG	-
11	IMAGE_DIRECTORY_ENTRY_BOUND_-IMPORT	Table des importations liées
12	IMAGE_DIRECTORY_ENTRY_IAT	Table des adresses des imports
13	IMAGE_DIRECTORY_ENTRY_DELAY_-IMPORT	Descripteur des imports en différé
14	IMAGE_DIRECTORY_ENTRY_COM_-DESCRIPTOR	Entête COM
15	s.-/o.	Réservé

Entête de sections

Unités fondamentales de la structuration offerte par Portable Executable, les sections servent de conteneur pour des données de même nature ou qui ont une causalité commune. Relativement à ces aspects, la *table des sections* détaille les différents sections contenues dans le fichier.

Il existe autant d'entête de sections qu'il y a de sections dans le fichier. Le nombre de sections est défini dans l'attribut NumberOfSections de l'entête du fichier image.

```
typedef struct _IMAGE_SECTION_HEADER {
  BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
  union {
    DWORD PhysicalAddress;
    DWORD VirtualSize;
  } Misc;
  DWORD VirtualAddress;
  DWORD SizeOfRawData;
  DWORD PointerToRawData;
  DWORD PointerToRelocations;
  DWORD PointerToLinenumbers;
  WORD  NumberOfRelocations;
  WORD  NumberOfLinenumbers;
  DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

- *Nom (Name)* Séquence de caractères correspondant au nom de la section. Par convention, les noms de section commencent par un point (par exemple, .code). Si le nom de la section tient sur exactement huit octets, le caractère de terminaison est omis. Dans l'éventualité où les données définies à cet égard sont insuffisantes pour atteindre la dimension fixée, des octets de remplissage à zéro doivent être insérés. Le nom que porte une section est purement arbitraire, le chargeur d'image Windows n'en fait par ailleurs aucun cas. Le tableau plus loin donne quelques noms couramment employées.

- *Taille virtuelle - VirtualSize* - Taille de la section une fois celle-ci chargée en mémoire.

- *Adresse virtuelle* (VirtualAddress)* Adresse virtuelle relative de la section en mémoire. La valeur définie doit l'être conformément aux contraintes d'alignement de sections - à quoi correspond le champ SectionAlignment dans l'entête optionnelle.

- *Taille physique - SizeOfRawData* - Taille réellement occupée par la section sur le système de fichiers. La valeur définie doit l'être conformément aux contraintes d'alignement de fichiers - à quoi correspond le champ FileAlignment dans l'entête optionnelle.
- *Pointeur de données - PointerToRawData* - Emplacement de la section dans le fichier. La valeur définie doit l'être conformément aux contraintes d'alignement de fichiers.
- *Caractéristiques - Characteristics* - Elles définissent des attributs pour l'exécution, par exemple si une section est exécutable, si elle contient du code ou des données, etc.
- IMAGE_SCN_TYPE_NO_PAD -
- 0x00000020 - IMAGE_SCN_CNT_CODE - La section contient du code machine exécutable.
- 0x00000040 - IMAGE_SCN_CNT_INITIALIZED_DATA - La section contient des données initialisées.
- 0x00000080 - IMAGE_SCN_CNT_UNINITIALIZED_DATA - La section contient des données non initialisées.
- 0x00000100 - IMAGE_SCN_LNK_OTHER
- 0x00000200 - IMAGE_SCN_LNK_INFO - La section contient des commentaires ou des informations additionnelles. Correspond en général à la section .drectve. N'est valide que pour les fichiers objets.
- 0x00000400 - IMAGE_SCN_TYPE_OVER
- 0x00000800 - IMAGE_SCN_LNK_REMOVE - La section ne fait pas partie du fichier image. N'est valide que pour les fichiers objets.
- 0x00001000 - IMAGE_SCN_LNK_COMDAT - La section contient des fonctions et données empaquetées.
- 0x00004000 - IMAGE_SCN_NO_DEFER_SPEC_EXC
- 0x00008000 - IMAGE_SCN_MEM_FARDATA
- 0x00008000 - IMAGE_SCN_GPREL - La section peut être lue à partir du pointeur global (registre r1 sur IA64).
- 0x00010000 - IMAGE_SCN_MEM_SYSHEAP
- 0x00020000 - IMAGE_SCN_MEM_PURGEABLE
- 0x00040000 - IMAGE_SCN_MEM_LOCKED
- 0x00080000 - IMAGE_SCN_MEM_PRELOAD
- 0x00100000 - IMAGE_SCN_ALIGN_1BYTES - Alignement des données sur 1 octet.
- 0x00200000 - IMAGE_SCN_ALIGN_2BYTES - Alignement des données sur 2 octets.
- 0x00300000 - IMAGE_SCN_ALIGN_4BYTES - Alignement des données sur 4 octets.
- 0x00400000 - IMAGE_SCN_ALIGN_8BYTES - Alignement des données sur 8 octets.
- 0x00500000 - IMAGE_SCN_ALIGN_16BYTES - Alignement des données sur 16 octets.
- 0x00600000 - IMAGE_SCN_ALIGN_32BYTES - Alignement des données sur 32 octets.
- 0x00700000 - IMAGE_SCN_ALIGN_64BYTES - Alignement des données sur 64 octets.
- 0x00800000 - IMAGE_SCN_ALIGN_128BYTES - Alignement des données sur 128 octets.

- 0x00900000 - IMAGE_SCN_ALIGN_256BYTES - Alignement des données sur 256 octets.
- 0x00A00000 - IMAGE_SCN_ALIGN_512BYTES - Alignement des données sur 512 octets.
- 0x00B00000 - IMAGE_SCN_ALIGN_1024BYTES - Alignement des données sur 1024 octets.
- 0x00C00000 - IMAGE_SCN_ALIGN_2048BYTES - Alignement des données sur 2048 octets.
- 0x00D00000 - IMAGE_SCN_ALIGN_4096BYTES - Alignement des données sur 4096 octets.
- 0x00E00000 - IMAGE_SCN_ALIGN_8192BYTES - Alignement des données sur 8192 octets. - 0x01000000 - IMAGE_SCN_LNK_NRELOC_OVFL - La section contient des relogements étendus.
- 0x02000000 - IMAGE_SCN_MEM_DISCARDABLE - La section peut être détachée du fichier.
- 0x04000000 - IMAGE_SCN_MEM_NOT_CACHED - La section ne peut être mise en cache.
- 0x08000000 - IMAGE_SCN_MEM_NOT_PAGED - La section ne peut être paginée.
- 0x10000000 - IMAGE_SCN_MEM_SHARED - La section est partageable.
- 0x20000000 - IMAGE_SCN_MEM_EXECUTE - La section est exécutable.
- 0x40000000 - IMAGE_SCN_MEM_READ - La section est lisible.
- 0x80000000 - IMAGE_SCN_MEM_WRITE - La section est inscriptible.

Tableau 11.1. Noms de section couramment employés

Nom	Description
.code	Instructions du programme
.data	Variables initialisées
.idata	Table des importations
.rsrc	Ressources (curseurs, sons, menus, etc)

Etant donné qu'aucun des entêtes n'a de référence directe sur la table des sections, l'emplacement de cette table est calculée en fonction de la totale des entêtes (attribut SizeOfHeadears), à laquelle on ajoute la valeur 1.

Dumpbin

L'utilitaire Dumpbin permet l'exploration des données et référentiels de tout fichier conforme a la norme PE/COFF.

Dumpbin met en oeuvre les options suivantes :

- /ALL
- /ARCHIVEMEMBERS
- /CLRHEADER
- /DEPENDENTS
- /DIRECTIVES
- /DISASM
- /ERRORREPORT

- /EXPORTS
- /FPO
- /HEADERS
- /IMPORTS
- /LINENUMBERS
- /LINKERMEMBER
- /LOADCONFIG
- /OUT
- /PDATA
- /PDBPATH
- /RANGE
- /RAWDATA
- /RELOCATIONS
- /SECTION
- /SUMMARY
- /SYMBOLS
- /TLS

Pour exécuter Dumpbin, utilisez la syntaxe 'dumpbin [options]() fichiers...'. Spécifiez en paramètre un ou plusieurs fichiers ainsi que les options requises pour déterminer les informations voulues. Par défaut, Dumpbin affiche l'ensemble des informations sur la sortie standard. Vous pouvez modifier ce comportement et rediriger les informations vers un fichier ou utiliser l'option /out afin de spécifier un nom de fichier pour la sortie.

Chargeur d'image

Si une large majorité des tâches d'initialisation liées aux processus se concrétise au niveau de l'exécutif et du noyau, une autre partie des opérations se déroule en dehors de ces sphères, en l'occurence dans un ensemble spécial appelé chargeur d'image (*loader*).

Compte tenu de sa position centrale dans la structure relationnelle entre processus et système, le chargeur d'image prend corps parmi la bibliothèque générale Ntdll. Les instructions qui le composent sont par conséquent soumises aux mêmes restrictions en termes d'accès à la mémoire et de droits de sécurité que n'importe quel autre code exécuté en mode utilisateur. Le caractère emblématique - et partant la spécificité - de ces instructions tient au fait qu'elles ont une place qui leur est immanquablement assurée dans le processus en cours d'exécution (Ntdll.dll est toujours chargé) et qu'elles constituent les premières à s'exécuter dans le contexte mode utilisateur d'un nouveau processus. (Lorsque le système donne lieu à l'état initial d'un processus, il le fait en enregistrant auprès du registre pointeur d'instruction une valeur correspondant à une fonction d'initialisation dans Ntdll. Reportez-vous au chapitre consacré aux processus et aux threads pour plus d'informations sur ces aspects.)

Le chargeur d'image, étant donné qu'il devance l'exécution du code d'application réel, reste la plupart du temps invisible et ignorée autant des utilisateurs que des concepteurs de logiciels. Hors la procédure d'initialisation les concernant, les programmes interagissent avec le chargeur lors de différentes opérations, par exemple

le chargement ou déchargement de DLL (LoadLibrary) ou l'interrogation de l'adresse de base d'une image (GetModuleHandle).

Les attributions principales du chargeur sont notamment celles que voici :

- Initialisation de l'état mode utilisateur du processus et configuration des vecteurs de stockage qui résident dans l'espace d'adressage afférent, par exemple tas, emplacements TLS et FLS.
- Chargement et déchargement des DLL (au moment de l'exécution ou à la demande) et suivi des modules chargés en mémoire.
- Support du système de mise à jour et prise en charge de l'application des correctifs (hotpatching).
- Gestion des fichiers manifest
- Consultation de la base de données de compatibilité des applications et mise en conformité de l'environnement d'exécution relativement aux exigences voulues.

Une fois le processus mis en place, le chargeur continuer de piloter l'exécution en fonction d'une trame de contextee située sur la pile, alimentée au préalable par le noyau, qui contient le point d'entrée réel de l'application. Sur le plan concret, cette séquence s'appuie sur une API native spéciale, et non via une logique de saut depuis le chargeur vers une autre région de l'espace d'adressage. Cette façon de procéder explique pourquoi l'arborescence des appels pour un un thread ne fait jamais paraitre aucune des fonctions d'initialisation du chargeur.

Base de données des modules

Tout comme le noyau le fait vis à vis des pilotes, le chargeur tient à jour un ensmeble d'informations concernant tous les modules (DLL et fichier d'application principal) chargés par un processus. Il actualise à cet effet trois listes, toutes contenant les mêmes informations, mais ordonnées différemment (soit par ordre de chargement, par emplacement de mémoire ou par ordre d'initialisation). Ces listes, dont l'origine est donnée par la sous-structure Ldr (PEB_LDR_DATA) du bloc PEB d'un processus, contiennent des structures appelées entrées de table de données du chargeur (LDR_DATA_TABLE_ENTRY) qui stockent des informations sur chaque module. La liste suivante répertorie les différentes informations que le chargeur conserve dans une entrée.

```
lkd> dt nt!_LDR_DATA_TABLE_ENTRY
   +0x000 InLoadOrderLinks : _LIST_ENTRY
   +0x010 InMemoryOrderLinks : _LIST_ENTRY
   +0x020 InInitializationOrderLinks : _LIST_ENTRY
   +0x030 DllBase          : Ptr64 Void
   +0x038 EntryPoint       : Ptr64 Void
   +0x040 SizeOfImage      : Uint4B
   +0x048 FullDllName      : _UNICODE_STRING
   +0x058 BaseDllName      : _UNICODE_STRING
   +0x068 FlagGroup        : [4] UChar
   +0x068 Flags            : Uint4B
   +0x068 PackagedBinary   : Pos 0, 1 Bit
   +0x068 MarkedForRemoval : Pos 1, 1 Bit
   +0x068 ImageDll         : Pos 2, 1 Bit
   +0x068 LoadNotificationsSent : Pos 3, 1 Bit
   +0x068 TelemetryEntryProcessed : Pos 4, 1 Bit
   +0x068 ProcessStaticImport : Pos 5, 1 Bit
   +0x068 InLegacyLists    : Pos 6, 1 Bit
   +0x068 InIndexes        : Pos 7, 1 Bit
   +0x068 ShimDll          : Pos 8, 1 Bit
   +0x068 InExceptionTable : Pos 9, 1 Bit
   +0x068 ReservedFlags1   : Pos 10, 2 Bits
   +0x068 LoadInProgress   : Pos 12, 1 Bit
   +0x068 LoadConfigProcessed : Pos 13, 1 Bit
   +0x068 EntryProcessed   : Pos 14, 1 Bit
```

```
+0x068 ProtectDelayLoad : Pos 15, 1 Bit
+0x068 ReservedFlags3   : Pos 16, 2 Bits
+0x068 DontCallForThreads : Pos 18, 1 Bit
+0x068 ProcessAttachCalled : Pos 19, 1 Bit
+0x068 ProcessAttachFailed : Pos 20, 1 Bit
+0x068 CorDeferredValidate : Pos 21, 1 Bit
+0x068 CorImage         : Pos 22, 1 Bit
+0x068 DontRelocate     : Pos 23, 1 Bit
+0x068 CorILOnly        : Pos 24, 1 Bit
+0x068 ChpeImage        : Pos 25, 1 Bit
+0x068 ReservedFlags5   : Pos 26, 2 Bits
+0x068 Redirected       : Pos 28, 1 Bit
+0x068 ReservedFlags6   : Pos 29, 2 Bits
+0x068 CompatDatabaseProcessed : Pos 31, 1 Bit
+0x06c ObsoleteLoadCount : Uint2B
+0x06e TlsIndex         : Uint2B
+0x070 HashLinks        : _LIST_ENTRY
+0x080 TimeDateStamp    : Uint4B
+0x088 EntryPointActivationContext : Ptr64 _ACTIVATION_CONTEXT
+0x090 Lock             : Ptr64 Void
+0x098 DdagNode         : Ptr64 _LDR_DDAG_NODE
+0x0a0 NodeModuleLink   : _LIST_ENTRY
+0x0b0 LoadContext      : Ptr64 _LDRP_LOAD_CONTEXT
+0x0b8 ParentDllBase    : Ptr64 Void
+0x0c0 SwitchBackContext : Ptr64 Void
+0x0c8 BaseAddressIndexNode : _RTL_BALANCED_NODE
+0x0e0 MappingInfoIndexNode : _RTL_BALANCED_NODE
+0x0f8 OriginalBase     : Uint8B
+0x100 LoadTime         : _LARGE_INTEGER
+0x108 BaseNameHashValue : Uint4B
+0x10c LoadReason       : _LDR_DLL_LOAD_REASON
+0x110 ImplicitPathOptions : Uint4B
+0x114 ReferenceCount   : Uint4B
+0x118 DependentLoadFlags : Uint4B
+0x11c SigningLevel     : UChar
```

- *Nom de module (BaseDllName)* Nom de fichier du module, sans le chemin complet d'où il se situe. Voir fonction de support LdrRegisterDllNotification ; attribut BaseDllName de structure LDR_DLL_LOADED_NOTIFICATION_DATA.

- *Adresse de base (DllBase)* Renferme l'adresse de base à laquelle le module a été chargé. Voir fonction de support LdrRegisterDllNotification ; attribut DllBase de structure LDR_DLL_LOADED_NOTIFICATION_DATA.

- *Point d'entrée (EntryPoint)* Adresse de la fonction principale du module (telle que DllMain).

- *Flags* Indicateurs d'état du chargeur pour ce module (voir tableau plus loin une description des indicateurs). Voir fonction de support LdrRegisterDllNotification ; attribut Flags dans les structures LDR_DLL_LOADED_NOTIFICATION_DATA et SYSTEM_MODULE_INFORMATION.

- *Nom de chemin complet (FullDllName)* Nom de chemin pleinement qualifié du module. Voir fonction de support LdrRegisterDllNotification ; attribut FullDllName de structure LDR_DLL_LOADED_NOTIFICATION_DATA.

- *Nombre de charges (LoadCount)* Comptabilise le nombre de fois que le module a été chargé (fait office de compteur de références). Voir fonction LdrpMapDll ; structure LoadCount RTL_PROCESS_MODULE_INFORMATION, et notamment l'attribut LoadCount ; routine MiLoadUserSymbols et MmLoadSystemImage.

- *Données de correctif (PatchInformation)* Informations pertinentes pour l'application de correctifs (hotpatch) sur le module.

- *ServiceTagLinks* Liste des services qui référencent ce module.

- *Taille de l'image (SizeOfImage)* Taille du module en mémoire, exprimée en nombre d'octets. Voir fonction de support LdrRegisterDllNotification ; attribut SizeOfImage de structure LDR_DLL_LOADED_NOTIFICATION_DATA.

- *Cachet horodaté (TimeDateStamp)* Enregistre l'heure et la date à laquelle le module a passé la phase d'éditions des liens (et a à cette occasion trouvé sa forme finale). Le chargeur se procure cette information auprès de l'entête de l'image du module. Voir IMAGE_FILE_HEADER TimeDateStamp.

La liste qui suit répertorie les valeurs qui ont une incidence sur l'attribut Flags de la structure LDR_DATA_TABLE_ENTRY.

- *LDRP_STATIC_LINK (0x00000002)*

- *LDRP_IMAGE_DLL (0x00000004)* Voir fonction LdrpMapDll ; attribut Characteristics de structure IMAGE_FILE_HEADER ; constante IMAGE_FILE_DLL.

- *LDRP_LOAD_IN_PROGRESS (0x00001000)* Voir routine MmLoadSystemImage.

- *LDRP_UNLOAD_IN_PROGRESS (0x00002000)*

- *LDRP_ENTRY_PROCESSED (0x00004000)* Le chargeur a terminé le traitement du module.

- *LDRP_DONT_CALL_FOR_THREADS (0x00040000)*

- *LDRP_PROCESS_ATTACH_CALLED (0x00080000)*

- *LDRP_DEBUG_SYMBOLS_LOADED (0x00100000)*

- *LDRP_IMAGE_NOT_AT_BASE (0x00200000)* L'image a été déplacée de son

- *LDRP_COR_IMAGE (0x00400000)* Le module est une application .NET.

- *LDRP_COR_OWNS_UNMAP (0x00800000)*

- *LDRP_SYSTEM_MAPPED (0x01000000)* Voir routine MmLoadSystemImage.

- *LDRP_IMAGE_VERIFYING (0x02000000)*

- *LDRP_DRIVER_DEPENDENT_DLL (0x04000000)*

- *LDRP_ENTRY_NATIVE (0x08000000)* Voir routine MmLoadSystemImage.

- *LDRP_REDIRECTED (0x10000000)*

- *LDRP_MM_LOADED (0x40000000)* Voir routine MmLoadSystemImage.

- _LDRP_COMPAT_DATABASE_PROCESSED (0x80000000)

Initialisation post traitement

Au terme du chargement des DLL requises, le système doit encore pour assurer le fonctionnement optimal d'une application mener à bien un petit nombre de tâches spécialisées complémentaires.

1. Découverte des informations de relogement que pourraient éventuellement faire apparaitre l'application et, s'il s'en trouve, traitement des entrées de relocation appropriées.

2. Vérification de l'utilisation du stockage local au thread (TLS, Thread Local Storage) et découverte des entrées TLS à allouer et configurer.

3. Mise en conformité par rapport à la protection contre l'exécution (DEP).

4. Dans l'éventualité où le programme présente des problèmes de compatibilité connus, déploiement de la solution automatisée basée sur shim.

5. Exécution des initialiseurs pour tous les modules chargés.

L'exécution des initialiseurs fait partie des dernières opérations que le chargeur effectue pour le compte du système d'exploitation. C'est en ces circonstances que la routine d'initialisation de chaque DLL (DllMain) est appelée - offrant ainsi à chaque DLL l'occasion de mener à bien ses propres taches d'initialisation. Cette étape est aussi l'une des dernières durant laquelle le chargement d'une application peut échouer. Dans l'éventualité où au moins une DLL chargée génère à l'issue de sa fonction DllMain un code de retour apparenté à une erreur, le chargeur interrompt le démarrage de l'application.

Import address Table (IAT)

Chaque programme au format PE contient une table d'importation qui spécifie quelles DLL il utilise et quelles fonctions de ces DLL sont nécessaires à son exécution.

```
C:\dumpbin /imports C:\Windows\System32\notepad.exe

Microsoft (R) COFF/PE Dumper Version 14.31.31107.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file C:\Windows\System32\notepad.exe

File Type: EXECUTABLE IMAGE

  Section contains the following imports:

    GDI32.dll
             14002A930 Import Address Table
             140031D58 Import Name Table
                     0 time date stamp
                     0 Index of first forwarder reference

                         391 SetMapMode
                         3A5 SetViewportExtEx
                         3A9 SetWindowExtEx
                         2F7 LPtoDP
                         37C SetBkMode
                         2E7 GetTextMetricsW
                         3B6 TextOutW
                           0 AbortDoc
                         199 EndDoc
                         376 SetAbortProc

...

    USER32.dll
             14002AA00 Import Address Table
             140031E28 Import Name Table
                     0 time date stamp
                     0 Index of first forwarder reference

                         2AF PostQuitMessage
                          11 BeginPaint
                          F4 EndPaint
                         110 FillRect
                          DE DrawTextW
                          D2 DrawFocusRect
```

```
                      A7 DefWindowProcW
                     3B3 TrackMouseEvent
                     224 InvalidateRect
                      B1 DestroyIcon

...

  Summary

        3000 .data
        1000 .didat
        2000 .pdata
        B000 .rdata
        1000 .reloc
       1F000 .rsrc
       29000 .text
```

Chapitre 12. Sécurité

Garantir une protection efficace contre les accès non autorisés constitue une fonctionnalité essentielle dans tout tout environnement où de multiples utilisateurs ont accès à des ressources communes. Dans cette perspective font sens des prérogatives de nature et de degré divers, par exemple la protection des données personnelles (dossiers et fichiers de l'utilisateur) contre toute lecture ou modification inopportune, l'intégrité de la mémoire, la détection et la prévention des attaques, l'identification, l'authentification et l'autorisation des personnes, le suivi et le signalement des atteintes portées au système d'exploitation ou à ses composants, et bien d'autres mécanismes qui sont à l'oeuvre. Dans ce chapitre, nous allons expliquer comment différents aspects de Microsoft Windows se manifestent et cohabitent afin de lieu à une sécurité robuste et complète.

Composants du système de sécurité

La liste suivante énumère les composants fondamentaux de Windows en matière de sécurité. Notez que le catalogue ainsi constitué ne présente que les entités les plus directement impliquées dans ce domaine, à l'exclusion, donc, des dispositifs périphériques, par exemple la séquence SAS, et des mécanismes internes entrant en jeu dans l'implémentation. - *SRM (_Security Reference Monitor*)_, Moniteur de référence de la sécurité. Composant de l'exécutif Windows responsable de la mise en œuvre et de la stricte application des stratégies de contrôle d'accès. Il procède a ce titre aux vérifications requises pour statuer si une entité (un utilisateur ou un processus agissant pour son compte) demandant d'accéder à une ressource a les droits nécessaires pour le faire, valide ou non les tentatives d'accès aux objets, et par cela génère les messages d'audit utilisés par le sous-système de sécurité.

- *Processus d'ouverture de Session (Winlogon)*. Processus mode utilisateur responsable de la gestion des sessions interactives. Ainsi, quand un utilisateur ouvre une session, Winlogon créé et rend visible à ce dernier l'interface utilisateur du système d'exploitation.

- *LSASS (_Local Security Authority Subsystem*)_, Sous-système d'autorité de la sécurité locale. Processus mode utilisateur, exécutant l'image Lsass.exe, responsable de la stratégie de sécurité du système local, incluant l'identification et l'authentification des utilisateurs ainsi que la gestion de leurs privilèges, les politiques de mot de passe et les paramètres d'audit de la sécurité du système.

- *LSA database*, Base de données des stratégies LSA. Base de données hébergeant un contenu en rapport avec les aspects rationnels de la stratégie de sécurité du système local. Elle est stockée dans le registre sous HKEY_LOCAL_MACHINE\Security, dont l'accès est protégé au niveau système afin d'interdire sa manipulation par des utilisateurs standards.

- *Network Logon Service*, Service d'ouverture de session réseau. Service Windows (\Windows \System32\Netlogon.dll) prenant en charge les événements d'ouverture de session pour les ordinateurs d'un domaine. L'ouverture de session en réseau est traitée comme une ouverture de session interactive, avec au préalable l'ouverture d'un canal sécurisé vers un contrôleur de domaine.

- *SAM database*, Base de données SAM. Base de données qui, sur les systèmes ne faisant pas office de contrôleurs de domaine, contient les noms des utilisateurs et des groupes locaux, avec leurs mots de passe, leurs identifiants de sécurité et leurs attributs. Elle est située dans le registre, sous HKEY_LOCAL_MACHINE \SAM, dont l'accès est protégé.

- *Security Accounts Manager*, Service SAM. Ensemble de contrôles et de routines chargés de gérer la base de données des comptes locaux. La base de données SAM (pour Security Account Manager) est l'un des composants du Registre. Le service SAM (pour Security Accounts Manager, avec un s à Account)(Samsrv.dll) est exécuté dans le processus LSASS.EXE.

- *Authentication Package*, Package d'authentification. DLL, exécutée dans le contexte du processus LSASS, qui implémente la stratégie d'authentification de Windows. Cette DLL est chargée de vérifier les informations d'identification fournies pour l'ouverture de session et de retourner le résultat à l'autorité concernée, en l'occurrence le processus Winlogon.

- *Active Directory* Service d'annuaire qui fournit des mécanismes centralisés d'identification et d'authentification à un réseau d'ordinateurs. Le serveur Active Directory (\Windows\System32\Ntdsa.dll) est exécuté dans la processus LSASS. # SRM (Security Reference Monitor)

La centralisation de la gestion des entités système au sein du gestionnaire d'objets permet de donner lieu à un mécanisme uniforme pour contrôler les tentatives d'accès aux ressources accessibles par les utilisateurs. Ainsi, chaque fois qu'un processus se manifeste auprès du système en vue d'obtenir un handle pour un quelconque objet, le moniteur de référence sécurité (SRM, Security Reference Monitor) vérifie le jeton de sécurité du processus et la liste de contrôle d'accès de l'objet, et détermine alors si le processus a des droits d'accès suffisants pour mener à bien l'opération.

Une responsabilité qui incombe au premier chef au SRM est l'application des stratégies de sécurité sur l'ordinateur local. Il définit pour ce faire un ensemble de moyens et de procédés donnant matière matière à des objets pour lesquels il est possible de spécifier des autorisations.

Outre être un médiateur incontournable dans toutes les relations entre objets et acteurs du modèle de sécurité de Windows, le SRM est également responsable de la manipulation des privilèges qu'incorporent les jetons d'accès. Pour mieux maîtriser la protection du système d'exploitation, des privilèges spéciaux sont demandées aux utilisateurs dès lors qu'ils s'essaient à des actions sensibles sur la machine, par exemple surmonter certaines vérifications en tant qu'administrateur, apporter des modifications aux systèmes de fichiers, déboguer des programmes, etc.

Un autre rôle majeur du SRM est d'enregistrer les événements journalisés relatifs à la sécurité. Différentes exigences en la matière imposent que le système ait la capacité de détecter et d'enregistrer toutes les tentatives d'accès aux ressources système. Le SRM étant responsable des contrôles d'accès, il est dès lors naturel que la plupart des enregistrements dans le journal Sécurité soit du fait de ce composant.

Contrôle d'accès

Jetons d'accès

Le moniteur de référence de sécurité (SRM) emploie un objet appelé *jeton* (ou *jeton d'accès*) de sorte à représenter le contexte de sécurité d'un processus ou d'un thread. Un contexte de sécurité se compose d'informations décrivant les privilèges, comptes et groupes associés au thread.

Pendant l'ouverture de session, Winlogon crée un jeton initial pour représenter l'utilisateur autorisé et attache le jeton au processus auquel il donne lieu initialement (par défaut, Userinit.exe). Comme les processus enfant héritent par défaut d'un exemplaire du jeton de leur créateur, tous les processus de la session de l'utilisateur sont exécutés sous le même jeton, par conséquent avec le même profil de sécurité.

Par défaut, un thread n'a pas de jeton propre, dans la mesure où c'est le jeton du processus auquel il appartient qui suffit à ce niveau à lui conférer un contexte de sécurité. Par contre, dans l'éventualité où un thread s'appuie sur de l'emprunt d'identité pour agir au nom d'un sujet ou d'un processus client, un jeton d'accès spécifiquement conçu à cet égard (appelé en interne jeton d'emprunt d'identité) lui est alors automatiquement assigné.

Un jeton est créé par suite d'une ouverture de session utilisateur afin de représenter l'utilisateur ayant mis en marche les procédures de validation d'identité. Par la suite, tous les programmes qu'exécute l'utilisateur héritent d'une copie de ce jeton initial.

Tableau 12.1. Droits d'accès concernant les jetons de sécurité

Constante	Valeur	Description
TOKEN_ADJUST_DEFAULT	0x0080	Obligatoire pour modifier le propriétaire par défaut, le groupe principal ou la liste DACL d'un jeton d'accès.

Constante	Valeur	Description
TOKEN_ADJUST_GROUPS	0x0040	Obligatoire pour modifier les attributs de groupes dans un jeton d'accès. Controlé par la fonction NtAdjustGroupsToken.
TOKEN_ADJUST_PRIVILEGES	0x0020	Obligatoire pour activer ou désactiver des privilèges dans un jeton d'accès. Controlé par la fonction NtAdjustPrivilegesToken.
TOKEN_ADJUST_SESSIONID	0x0100	Obligatoire pour ajuster l'ID de session d'un jeton d'accès. Le privilège SE_-TCB_NAME est requis. Controlé par la fonction NtSetInformationToken (TokenInformationClass = TokenSessionId).
TOKEN_ASSIGN_PRIMARY	0x0001	Obligatoire pour attacher un jeton principal à un processus. Controlé par la routine PspSetPrimaryToken.
TOKEN_DUPLICATE	0x0002	Obligatoire pour dupliquer un jeton d'accès. Controlé par les fonctions NtDuplicateToken et NtFilterToken.
TOKEN_EXECUTE	-	Combine STANDARD_RIGHTS_EXECUTE et TOKEN_IMPERSONATE.
TOKEN_IMPERSONATE	0x0004	Obligatoire pour associer un jeton d'emprunt d'identité à un processus. Controlé par la routine PsAssignImpersonationToken.
TOKEN_QUERY	0x0008	Obligatoire pour interroger un jeton d'accès. Controlé par la fonction NtQueryInformationToken.
TOKEN_QUERY_SOURCE	0x0010	Obligatoire pour interroger la source d'un jeton d'accès. Controlé par NtQueryInformationToken, service appelé depuis GetTokenInformation.
TOKEN_READ	-	Combine STANDARD_RIGHTS_READ et TOKEN_QUERY.
TOKEN_WRITE	-	Combine STANDARD_RIGHTS_WRITE, TOKEN_ADJUST_PRIVILEGES, TOKEN_-ADJUST_GROUPS et TOKEN_ADJUST_-DEFAULT.
TOKEN_ALL_ACCESS	-	Combine l'ensemble des droits d'accès admissibles pour un jeton d'accès.

```
lkd> dt nt!_TOKEN
   +0x000 TokenSource      : _TOKEN_SOURCE
   +0x010 TokenId          : _LUID
   +0x018 AuthenticationId : _LUID
   +0x020 ParentTokenId    : _LUID
   +0x028 ExpirationTime   : _LARGE_INTEGER
   +0x030 TokenLock        : Ptr64 _ERESOURCE
   +0x038 ModifiedId       : _LUID
   +0x040 Privileges       : _SEP_TOKEN_PRIVILEGES
   +0x058 AuditPolicy      : _SEP_AUDIT_POLICY
   +0x078 SessionId        : Uint4B
   +0x07c UserAndGroupCount : Uint4B
   +0x080 RestrictedSidCount : Uint4B
   +0x084 VariableLength   : Uint4B
   +0x088 DynamicCharged   : Uint4B
   +0x08c DynamicAvailable : Uint4B
   +0x090 DefaultOwnerIndex : Uint4B
   +0x098 UserAndGroups    : Ptr64 _SID_AND_ATTRIBUTES
   +0x0a0 RestrictedSids   : Ptr64 _SID_AND_ATTRIBUTES
   +0x0a8 PrimaryGroup     : Ptr64 Void
   +0x0b0 DynamicPart      : Ptr64 Uint4B
   +0x0b8 DefaultDacl      : Ptr64 _ACL
```

```
   +0x0c0 TokenType        : _TOKEN_TYPE
   +0x0c4 ImpersonationLevel : _SECURITY_IMPERSONATION_LEVEL
   +0x0c8 TokenFlags       : Uint4B
   +0x0cc TokenInUse       : UChar
   +0x0d0 IntegrityLevelIndex : Uint4B
   +0x0d4 MandatoryPolicy  : Uint4B
   +0x0d8 LogonSession     : Ptr64 _SEP_LOGON_SESSION_REFERENCES
   +0x0e0 OriginatingLogonSession : _LUID
   +0x0e8 SidHash          : _SID_AND_ATTRIBUTES_HASH
   +0x1f8 RestrictedSidHash : _SID_AND_ATTRIBUTES_HASH
   +0x308 pSecurityAttributes : Ptr64 _AUTHZBASEP_SECURITY_ATTRIBUTES_INFORMATION
   +0x310 Package          : Ptr64 Void
   +0x318 Capabilities     : Ptr64 _SID_AND_ATTRIBUTES
   +0x320 CapabilityCount  : Uint4B
   +0x328 CapabilitiesHash : _SID_AND_ATTRIBUTES_HASH
   +0x438 LowboxNumberEntry : Ptr64 _SEP_LOWBOX_NUMBER_ENTRY
   +0x440 LowboxHandlesEntry : Ptr64 _SEP_LOWBOX_HANDLES_ENTRY
   +0x448 pClaimAttributes : Ptr64 _AUTHZBASEP_CLAIM_ATTRIBUTES_COLLECTION
   +0x450 VariablePart     : Uint8B

lkd> !process 0 1 windbg.exe
PROCESS fffffa8001097a80
    SessionId: 1  Cid: 0e8c    Peb: 7f74288c000  ParentCid: 06c4
    DirBase: 2f11d000  ObjectTable: fffff8a00143b700  HandleCount: <Data Not Accessible>
    Image: windbg.exe
    VadRoot fffffa8001ce64c0 Vads 98 Clone 0 Private 3921. Modified 669. Locked 1.
    DeviceMap fffff8a00431e9d0
    Token                             fffff8a00208b060
    ElapsedTime                       00:26:21.351
    UserTime                          00:00:00.000
    KernelTime                        00:00:00.000
    QuotaPoolUsage[PagedPool]         244528
    QuotaPoolUsage[NonPagedPool]      12688
    Working Set Sizes (now,min,max)  (7643, 50, 345) (30572KB, 200KB, 1380KB)
    PeakWorkingSetSize                11548
    VirtualSize                       155 Mb
    PeakVirtualSize                   156 Mb
    PageFaultCount                    20303
    MemoryPriority                    BACKGROUND
    BasePriority                      8
    CommitCharge                      8185
    Job                               fffffa800316f060

lkd> !token fffff8a00208b060
_TOKEN fffff8a00208b060
TS Session ID: 0x1
User: S-1-5-21-334366372-3902264010-286681719-1000
User Groups:
 00 S-1-5-21-334366372-3902264010-286681719-513
    Attributes - Mandatory Default Enabled
 01 S-1-1-0
    Attributes - Mandatory Default Enabled
 02 S-1-5-21-334366372-3902264010-286681719-1001
    Attributes - Mandatory Default Enabled
 03 S-1-5-32-544
    Attributes - Mandatory Default Enabled Owner
 04 S-1-5-32-545
    Attributes - Mandatory Default Enabled
 05 S-1-5-4
    Attributes - Mandatory Default Enabled
 06 S-1-2-1
    Attributes - Mandatory Default Enabled
 07 S-1-5-11
    Attributes - Mandatory Default Enabled
 08 S-1-5-15
```

```
    Attributes - Mandatory Default Enabled
 09 S-1-5-5-0-83478
    Attributes - Mandatory Default Enabled LogonId
 10 S-1-2-0
    Attributes - Mandatory Default Enabled
 11 S-1-5-64-10
    Attributes - Mandatory Default Enabled
 12 S-1-16-12288
    Attributes - GroupIntegrity GroupIntegrityEnabled
Primary Group: S-1-5-21-334366372-3902264010-286681719-513
Privs:
 05 0x000000005 SeIncreaseQuotaPrivilege          Attributes -
 08 0x000000008 SeSecurityPrivilege               Attributes -
 09 0x000000009 SeTakeOwnershipPrivilege          Attributes -
 10 0x00000000a SeLoadDriverPrivilege             Attributes -
 11 0x00000000b SeSystemProfilePrivilege          Attributes -
 12 0x00000000c SeSystemtimePrivilege             Attributes -
 13 0x00000000d SeProfileSingleProcessPrivilege   Attributes -
 14 0x00000000e SeIncreaseBasePriorityPrivilege   Attributes -
 15 0x00000000f SeCreatePagefilePrivilege         Attributes -
 17 0x000000011 SeBackupPrivilege                 Attributes -
 18 0x000000012 SeRestorePrivilege                Attributes -
 19 0x000000013 SeShutdownPrivilege               Attributes -
 20 0x000000014 SeDebugPrivilege                  Attributes - Enabled
 22 0x000000016 SeSystemEnvironmentPrivilege      Attributes -
 23 0x000000017 SeChangeNotifyPrivilege           Attributes - Enabled Default
 24 0x000000018 SeRemoteShutdownPrivilege         Attributes -
 25 0x000000019 SeUndockPrivilege                 Attributes -
 28 0x00000001c SeManageVolumePrivilege           Attributes -
 29 0x00000001d SeImpersonatePrivilege            Attributes - Enabled Default
 30 0x00000001e SeCreateGlobalPrivilege           Attributes - Enabled Default
 33 0x000000021 SeIncreaseWorkingSetPrivilege     Attributes -
 34 0x000000022 SeTimeZonePrivilege               Attributes -
 35 0x000000023 SeCreateSymbolicLinkPrivilege     Attributes -
Authentication ID:         (0,1469f)
Impersonation Level:       Anonymous
TokenType:                 Primary
Source: User32             TokenFlags: 0x2000 ( Token in use )
Token ID: a53ca            ParentToken ID: 0
Modified ID:               (0, a5826)
RestrictedSidCount: 0      RestrictedSids: 0000000000000000
OriginatingLogonSession: 3e7
PackageSid: (null)
CapabilityCount: 0      Capabilities: 0000000000000000
LowboxNumberEntry: 0000000000000000
```

- *DACL par défaut (DefaultDacl)* Liste d'ACLs par défaut que le système applique aux objets créés avec ce jeton et si aucune ACL n'a été spécifiée lors de la création.

- *Date d'expiration (ExpirationTime)* Champ non utilisé pour le moment mais qui, s'il l'était, ouvrirait une voie à la possibilité de gérer l'expiration de jeton. Voir service NtCreateToken ; routine SepCreateToken.

- *Niveau d'emprunt d'identité (ImpersonationLevel)* Fixe le niveau d'emprunt d'identité pour le jeton d'accès, et détermine les informations dont un autre processus dispose pour le compte d'un client. Voir routine SeTokenImpersonationLevel.

- *ID modifié (ModifiedId)* Identifiant local unique (LUID) employé de telle sorte à mettre en évidence d'éventuelles modifications apportées à un jeton.

- *SID restreints (RestrictedSids)*

- *ID de session (SessionId)* ID de la session à laquelle le jeton est associé. Voir service NtQueryInformationToken (TokenInformationClass = TokenSessionId) ; routine SeQuerySessionIdToken.

- *ID du jeton (TokenId)* Identifiant local unique (LUID) que le SRM assigne au jeton quand il le crée.
- *Flags (TokenFlags)* Modificateurs facultatifs qui définissent le comportement ou les caractéristiques du jeton.
- *Source du jeton (TokenSource)* Composant au nom et pour le compte duquel le jeton à été créé, par exemple le gestionnaire de session, un serveur de fichiers réseau ou le serveur RPC. Voir fonctions GetTokenInformation et SetTokenInformation.
- *Type (TokenType)* Établit une distinction entre un jeton principal et un jeton d'emprunt d'identité. Voir routine SeTokenType et SeQueryInformationToken.

La liste qui suit décrit les différents modificateurs qui se rapportent potentiellement à un jeton.

- TOKEN_HAS_ADMIN_GROUP (0x08) Voir routine SeTokenIsAdmin.
- TOKEN_HAS_BACKUP_PRIVILEGE (0x02)
- TOKEN_HAS_IMPERSONATE_PRIVILEGE (0x80)
- TOKEN_HAS_RESTORE_PRIVILEGE (0x04)
- TOKEN_HAS_TRAVERSE_PRIVILEGE (0x01)
- TOKEN_IS_RESTRICTED (0x10) Voir routine SeTokenIsRestricted.
- TOKEN_SESSION_NOT_REFERENCED (0x20)
- TOKEN_SANDBOX_INERT (0x40)

Les programmes désireux de savoir d'où un jeton tire son origine peuvent prendre connaissance du fait en question par l'intermédiaire de du champ Source du jeton (TokenSource), lequel contient une une brève description textuelle de l'entité ayant créé le jeton. Les possibilités à cet égard incluent le gestionnaire de session Windows, un serveur de fichiers réseau ou le serveur RPC.

Lorsque des changements ont lieu au sein d'un jeton, le système actualise en conséquence l'ID modifié (ModifiedId). En prêtant attention aux modifications apportées à un jeton, par extension aux caractéristiques et données d'identification qu'un jeton représente, les applications peuvent de cette manière s'assurer que le contexte de sécurité véhiculé par ce biais est bel et bien celui escompté, et n'a en l'occurence subi aucune altération depuis sa dernière utilisation.

La liste qui suit énumère les interfaces Windows utilisables en matière de contrôle et de manipulation de jetons d'accès.

- AdjustTokenGroups Modifie les informations de groupe contenues dans un jeton
- DuplicateToken Crée un nouveau jeton à partir d'un autre déjà existant
- GetTokenInformation Retourne des informations concernant le jeton spécifié en paramètre
- IsTokenRestricted Détermine si un jeton comporte des identificateurs de sécurité restreints
- OpenProcessToken Ouvre un handle vers le jeton auquel est associé un processus.
- SetThreadToken Définit un jeton d'emprunt d'identité pour un thread ou au contraire empêche un thread d'en utiliser un.

Fonctions	Services	Routines
	NtCreateToken	

Fonctions	Services	Routines
AdjustTokenGroups	NtAdjustGroupsToken	
AdjustTokenPrivileges	NtAdjustPrivilegesToken	
CheckTokenMembership		
DuplicateToken	NtDuplicateToken	
GetTokenInformation	NtQueryInformationToken	SeQueryInformationToken
OpenProcessToken	NtOpenProcessToken	
OpenThreadToken	NtOpenThreadToken	
SetThreadToken		
SetTokenInformation	NtSetInformationToken	

Tableau 12.2. TOKEN_INFORMATION_CLASS

Nom	Valeur	Type de données associé
TokenUser	01	TOKEN_USER
TokenGroups	02	TOKEN_GROUPS
TokenPrivileges	03	TOKEN_PRIVILEGES
TokenOwner	04	TOKEN_OWNER
TokenPrimaryGroup	05	TOKEN_PRIMARY_GROUP
TokenDefaultDacl	06	TOKEN_DEFAULT_DACL
TokenSource	07	TOKEN_SOURCE
TokenType	08	TOKEN_TYPE
TokenImpersonationLevel	09	SECURITY_IMPERSONATION_LEVEL
TokenStatistics	10	TOKEN_STATISTICS
TokenRestrictedSids	11	TOKEN_GROUPS
TokenSessionId	12	DWORD
TokenGroupsAndPrivileges	13	TOKEN_GROUPS_AND_PRIVILEGES
TokenSessionReference	14	ULONG
TokenSandBoxInert	15	
TokenAuditPolicy	16	TOKEN_AUDIT_POLICY
TokenOrigin	17	TOKEN_ORIGIN
TokenElevationType	18	TOKEN_ELEVATION_TYPE
TokenLinkedToken	19	TOKEN_LINKED_TOKEN
TokenElevation	20	TOKEN_ELEVATION
TokenHasRestrictions	21	ULONG
TokenAccessInformation	22	TOKEN_ACCESS_INFORMATION
TokenVirtualizationAllowed	23	ULONG
TokenVirtualizationEnabled	24	ULONG
TokenIntegrityLevel	25	TOKEN_MANDATORY_LABEL
TokenUIAccess	26	ULONG
TokenMandatoryPolicy	27	TOKEN_MANDATORY_POLICY
TokenLogonSid	28	TOKEN_GROUPS
TokenIsAppContainer	29	ULONG
TokenCapabilities	30	TOKEN_GROUPS
TokenAppContainerSid	31	TOKEN_APPCONTAINER_INFORMATION
TokenAppContainerNumber	32	ULONG
TokenUserClaimAttributes	33	CLAIM_SECURITY_ATTRIBUTES_INFORMATION
TokenDeviceClaimAttributes	34	CLAIM_SECURITY_ATTRIBUTES_INFORMATION
TokenRestrictedUserClaimAttributes	35	
TokenRestrictedDeviceClaimAttributes	36	

Nom	Valeur	Type de données associé
TokenDeviceGroups	37	TOKEN_GROUPS
TokenRestrictedDeviceGroups	38	TOKEN_GROUPS
TokenSecurityAttributes	39	
TokenIsRestricted	40	

TOKEN_USER.

```
typedef struct _TOKEN_USER {
  SID_AND_ATTRIBUTES User;
} TOKEN_USER, *PTOKEN_USER;
```

Types de jetons

Deux types de jetons d'accès coexistent dans le sous-système de sécurité : jeton principal (*primary token*) et jeton d'emprunt d'identité (*impersonation token*). Un jeton d'accès est attribuée à un processus pour représenter les informations par défaut de toute tâche (thread) effectuant des opérations au nom du processus lui-même plutôt que pour le compte d'un client. À l'inverse, un jeton d'emprunt d'identité est un dispositif de saisie du profil de sécurité d'un processus client, permettant à une tâche de se faire passer pour le processus client lors d'opérations de sécurité.

Un jeton d'accès principal résulte de l'authentification d'un utilisateur et de la mise en conformité de l'espace de de travail de cet utilisateur avec les stratégies locales en place. Quand un utilisateur est authentifié par le processus d'ouverture de session, un jeton d'accès est attaché au processus utilisateur. Par la suite, chaque création d'un processus dans la session implique un duplicata du jeton du processus père, assigné pour l'occasion à un processus fils. Des jetons principaux peuvent être créés de toute pièce (CreateToken et le privilège SeCreateTokenPrivilege), par authentification d'un utilisateur (LogonUser) ou restriction d'un jeton existant (CreateRestrictedToken), et assignés ensuite à de nouveaux processus (CreateProcessAsUser et le privilège SeAssignPrimaryTokenPrivilege).

Un jeton d'emprunt d'identité permet de représenter un contexte de sécurité différent du sien. Ce type de jeton, créé après authentification d'un utilisateur ou d'un autre dispositif client (par exemple, un client de canal nommé ou un client RPC) permetde connaitre et d'utiliser le profil de sécurité d'une entité tierce. Dans ce cadre, le jeton utilisé lors du contrôle d'accès est celui du thread, non plus celui du processus, ceci assurant une sécurité à un niveau granulaire relativement bas.

Jetons restreints

Lorsque les relations entre différents acteurs de la sécurité sous Windows ne peuvent être jugées dignes de confiance, existe une variante spéciale du jeton d'accès, plus astreignante que son homologue traditionnel, appelée jeton restreint. Un jeton restreint est le duplicata d'un jeton principal ou d'emprunt d'identité, avec cette différence que certains privilèges peuvent être révoqués, limitant de fait toute entité s'exécutant dans ce contexte à accéder à des objets sécurisables ou effectuer des opérations nécessiteuses de privilèges. Le mécanisme peut, par exemple, être utilisé par un thread qui souhaiterait se faire passer pour un client, celui-ci appartenant à un domaine dont la délégation des services est douteuse, ou afin de démarrer un processus sensible, ce avec juste le niveau de privilège requis (juste au sens de strict nécessaire).

Le sous-système Windows fait apparaître les capacités de création de jetons restreints dans une unique fonction, CreateRestrictedToken, qui peut sur chaque jeton peut appliquer les mesures suivantes :

- Suppression de privilèges
- marquage de SID avec attribut DENY_ONLY
- ajout de SID dans la liste des SID restreint

Traditionnellement, dans le modèle de sécurité de Windows, tout accès qui n'est pas explicitement permis est refusée. Il semble donc logique de conclure qu'omettre un SID dans un jeton à pour effet de de refuser l'accès à une ressource donnée. Cela ne fonctionnera la plupart du temps, mais pas toujours. La raison en est que l'accès peut encore être accordée en fonction des groupes auxquels l'utilisateur appartient, et dont il ne serait pas mention dans le jeton.

Un exemple courant est le dépouillement des jetons nés de compte membres du groupe Administrateurs, qui disposent sinon de droits très importants.

Le système utilise la liste des SID restreint au moment de confronter le jeton d'accès du programme aux entrées de contrôle d'accès de l'objet sollicité. Si aucun SID n'est présent dans la liste (TOKEN RestrictedSids), le contrôle d'accès est effectué sur la base de la liste des SID de l'utilisateur et des groupes auxquels l'utilisateur appartient (TOKEN UserAndGroups). A l'inverse, si au moins un SID est présent, un second contrôle d'accès est réalisé, ce à partir de la liste des SID restreint. Les deux contrôles doivent individuellement valider l'accès à l'objet, à savoir que l'accès accordé consistera alors en le plus restrictif des deux.

Structure commune de sécurité

Pour la plupart des ressources allouées conduisant à la création d'un objet du noyau, il existe une structure commune pour gérer la sécurité. De fait, toutes les fonctions de l'API Windows concernés par ce schéma, tels que CreteFile, CreateMutex, CreateProcess, RegCreateKeyEx et d'autres, incorporent dans leur prototype d'appel un paramètre correspondant à un attribut de sécurité, ce sous la forme d'une structure SECURITY_ATTRIBUTES.

```
typedef struct _SECURITY_ATTRIBUTES {
    DWORD  nLength;
    LPVOID lpSecurityDescriptor;
    BOOL   bInheritHandle;
} SECURITY_ATTRIBUTES, *PSECURITY_ATTRIBUTES, *LPSECURITY_ATTRIBUTES;
```

Le champ nLength indique la taille en octets de la structure SECURITY_ATTRIBUTES. Le champ lpSecurityDescriptor renvoie sur un descripteur de sécurité contrôlant l'accès à l'objet. Le champ bInheritHandle spécifie si le nouveau handle créé fait partie des attributs transmissibles entre processus.

Descripteurs de sécurité

Le descripteur de sécurité est une structure attachée à chaque objet qui peut être protégé par le mécanisme de contrôle d'accès de Windows. Cela concerne des ressources de toute sorte, par exemple fichiers, dossiers et volumes, mais aussi processus, jetons d'accès, sémaphores et d'autres objets système.

Un descripteur de sécurité, lequel se présente en interne par le biais de la structure SECURITY_DESCRIPTOR, se compose des attributs qui voici :

- *Modèle de comportement (Control)* Modificateurs facultatifs qui régissent le comportement ou définissent les caractéristiques du descripteur.
- *Liste de contrôle d'accès discrétionnaire (Dacl)* Spécifie qui a accès à l'objet.
- *Groupe (Group)* SID du groupe principal de l'objet.
- *Propriétaire (Owner)* SID du propriétaire de l'objet. Voir fonction GetSecurityDescriptorOwner.
- *Numéro de révision (Revision)* Version du modèle de sécurité SRM utilisée pour l'expression et la formation du descripteur.
- *Liste de contrôle d'accès système (Sacl)* Spécifie quelles opérations, faites par quels utilisateurs, enregistrer dans le journal d'audit de la sécurité. .Structure SECURITY_DESCRIPTOR

```
lkd> dt nt!_SECURITY_DESCRIPTOR
   +0x000 Revision         : UChar
   +0x001 Sbz1             : UChar
   +0x002 Control          : Uint2B
   +0x008 Owner            : Ptr64 Void
   +0x010 Group            : Ptr64 Void
   +0x018 Sacl             : Ptr64 _ACL
   +0x020 Dacl             : Ptr64 _ACL
```

Le tableau qui suit passe en revue les différentes valeurs admises pour l'attribut Control de la structure.

Tableau 12.3. SECURITY_DESCRIPTOR Control

Valeur	Flag
0x0001	SE_OWNER_DEFAULTED
0x0002	SE_GROUP_DEFAULTED
0x0004	SE_DACL_PRESENT
0x0008	SE_DACL_DEFAULTED
0x0010	SE_SACL_PRESENT
0x0020	SE_SACL_DEFAULTED
0x0040	SE_DACL_UNTRUSTED
0x1000	SE_DACL_PROTECTED
0x2000	SE_SACL_PROTECTED
0x8000	SE_SELF_RELATIVE

- **SE_DACL_DEFAULTED** La DACL (liste de contrôle d'accès discrétionnaire) contenue dans le descripteur a été générée automatiquement par le système d'exploitation, en raison de l'absence (éventuellement de la corruption) de la DACL d'origine pour l'objet concerné.

- **SE_DACL_PRESENT** Indique que le descripteur de sécurité a une liste de contrôle d'accès discrétionnaire (DACL) qui lui est associé. Dans l'éventualité où cet attribut n'est pas défini, ou s'il l'est mais que la valeur de la DACL est nulle, le descripteur de sécurité permet un accès complet à tout le monde.

- **SE_OWNER_DEFAULTED** Les informations d'appartenance d'un objet (SID du propriétaire) ont été automatiquement définis par le système, plutôt que par son propriétaire lui-même. Cet indicateur peut être utilisé par un gestionnaire de ressources pour identifier les objets dont le propriétaire a été défini par un mécanisme par défaut. Voir fonction SetSecurityDescriptorOwner, service RtlSetOwnerSecurityDescriptor.

- **SE_GROUP_DEFAULTED** Un groupe de sécurité par défaut a été automatiquement affecté à l'objet, garantissant ainsi certaines autorisations de base. Voir fonction SetSecurityDescriptorGroup.

- **SE_SELF_RELATIVE** Indique que le descripteur de sécurité se présente sous le format auto-relatif (par opposition au format absolu). Voir fonctions GetSecurityDescriptorControl, MakeSelfRelativeSD et MakeAbsoluteSD.

- **SE_SACL_PRESENT** Le descripteur a une liste de contrôle d'accès système (SACL) qui lui est associé, détaillant quelles opérations doivent être auditées et enregistrées lors d'un accès à l'objet concerné. Voir fonction SetSecurityDescriptorSacl.

- **SE_DACL_UNTRUSTED** L'ACL pointée par la DACL du descripteur de sécurité a été fournie par une source considérée comme non fiable.

- **SE_DACL_PROTECTED** Empêche le descripteur d'hériter des paramètres de sécurité d'un autre objet.

- **SE_SACL_PROTECTED** Empêche que la SACL d'un descripteur de sécurité soit modifiée du fait d'ACE héritables. Voir fonction SetSecurityDescriptorControl.

Diverses fonctions intégrés à l'API, telles que *GetSecurityInfo*, GetFileSecurity, RegGetKeySecurity, GetKernelObjectSecurity, et d'autres, assurent une interface avec la structure commune d''un descripteur de sécurité. Après obtention de la référence idoine, le descripteur peut être manipulée à l'aide de fonctions spécifiques, par exemple RegSetKeySecurity ou SetFileSecurity.

Tableau 12.4. Fonctions Windows concernant les descripteurs de sécurité

Fonction	Description
GetSecurityDescriptorControl	Retourne des informations concernant un descripteur de sécurité.
GetSecurityDescriptorDacl	Donne accès à la liste de contrôle d'accès discrétionnaire d'un descripteur de sécurité.
GetSecurityDescriptorLength	Retourne la taille, en octets, d'une structure descripteur de sécurité (SECURITY_DESCRIPTOR).
InitializeSecurityDescriptor	Initialise un nouveau descripteur de sécurité.
Get/SetSecurityDescriptorOwner	Retourne ou modifie le propriétaire d'un descripteur de sécurité
ConvertStringSecurityDescriptorToSecurityDescriptor	Convertit un descripteur de sécurité au format texte en sa représentation idoine au niveau logiciel
ConvertSecurityDescriptorToStringSecurityDescriptor	Convertit un descripteur de sécurité en sa représentation idoine au format texte

Le descripteur de sécurité d'un répertoire ou d'un dossier peut être affiché ou modifié dans l'onglet Sécurité des propriétés de l'objet, ou grâce à l'outil icacls.exe en ligne de commande.

Tableau 12.5. Descripteurs de sécurité

Fonction	Service
GetKernelObjectSecurity	NtQuerySecurityObject
GetSecurityDescriptorControl	
GetSecurityDescriptorDacl	RtlGetDaclSecurityDescriptor
GetSecurityDescriptorLength	RtlLengthSecurityDescriptor
GetSecurityDescriptorOwner	RtlGetOwnerSecurityDescriptor
GetSecurityDescriptorSacl	RtlGetSaclSecurityDescriptor
InitializeSecurityDescriptor	RtlCreateSecurityDescriptor
IsValidSecurityDescriptor	RtlValidSecurityDescriptor
SetSecurityDescriptorOwner	RtlSetOwnerSecurityDescriptor

Visualisation d'un descripteur de sécurité

Pour voir les informations de sécurité concernant un objet, procédez comme suit.

1. Commencez par repérer l'entête de l'objet duquel vous souhaitez voir le descripteur de sécurité. Exemple, ici, avec le processus dans lequel est exécuté l'Explorateur Windows (explorer.exe).

```
lkd> !process 0 0 explorer.exe
PROCESS ffffa704a874a080
    SessionId: 2  Cid: 0aec    Peb: 00428000  ParentCid: 1144
    DirBase: 2fb79002  ObjectTable: ffffb9894d997280  HandleCount: 2453.
    Image: explorer.exe

lkd> !object ffffa704a874a080
Object: ffffa704a874a080  Type: (ffffa7049be747a0) Process
    ObjectHeader: ffffa704a874a050 (new version)
    HandleCount: 18  PointerCount: 511721
```

1. Tapez dt nt!_OBJECT_HEADER SecurityDescriptor, suivi de l'adresse de l'entête d'objet, de sorte à afficher la valeur du pointeur de descripteur de sécurité associé à l'objet.

```
lkd> dt nt!_OBJECT_HEADER SecurityDescriptor ffffa704a874a050
```

```
   +0x028 SecurityDescriptor : 0xffffb989`612918e1 Void
```

Les pointeurs de descripteur de sécurité que renferment les entêtes d'objet emploient les trois bits de poids faible comme flags.

```
lkd> !sd 0xffffb989`612918e1 & -8
->Revision: 0x1
->Sbz1    : 0x0
->Control : 0x8814
            SE_DACL_PRESENT
            SE_SACL_PRESENT
            SE_SACL_AUTO_INHERITED
            SE_SELF_RELATIVE
->Owner   : S-1-5-21-1223626255-212531765-3251176030-1001
->Group   : S-1-5-21-1223626255-212531765-3251176030-513
->Dacl    :
->Dacl    : ->AclRevision: 0x2
->Dacl    : ->Sbz1       : 0x0
->Dacl    : ->AclSize    : 0x5c
->Dacl    : ->AceCount   : 0x3
->Dacl    : ->Sbz2       : 0x0
->Dacl    : ->Ace[0]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->Dacl    : ->Ace[0]: ->AceFlags: 0x0
->Dacl    : ->Ace[0]: ->AceSize: 0x24
->Dacl    : ->Ace[0]: ->Mask : 0x001fffff
->Dacl    : ->Ace[0]: ->SID: S-1-5-21-1223626255-212531765-3251176030-1001

->Dacl    : ->Ace[1]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->Dacl    : ->Ace[1]: ->AceFlags: 0x0
->Dacl    : ->Ace[1]: ->AceSize: 0x14
->Dacl    : ->Ace[1]: ->Mask : 0x001fffff
->Dacl    : ->Ace[1]: ->SID: S-1-5-18

->Dacl    : ->Ace[2]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->Dacl    : ->Ace[2]: ->AceFlags: 0x0
->Dacl    : ->Ace[2]: ->AceSize: 0x1c
->Dacl    : ->Ace[2]: ->Mask : 0x00121411
->Dacl    : ->Ace[2]: ->SID: S-1-5-5-0-52064574

->Sacl    :
->Sacl    : ->AclRevision: 0x2
->Sacl    : ->Sbz1       : 0x0
->Sacl    : ->AclSize    : 0x1c
->Sacl    : ->AceCount   : 0x1
->Sacl    : ->Sbz2       : 0x0
->Sacl    : ->Ace[0]: ->AceType: SYSTEM_MANDATORY_LABEL_ACE_TYPE
->Sacl    : ->Ace[0]: ->AceFlags: 0x0
->Sacl    : ->Ace[0]: ->AceSize: 0x14
->Sacl    : ->Ace[0]: ->Mask : 0x00000003
->Sacl    : ->Ace[0]: ->SID: S-1-16-8192
```

ACL

Une liste de contrôle d'accès (ACL, Access Control List) est une collection séquentielle d'énoncés d'acceptation ou d'interdiction s'appliquant à différentes ressources (par exemple fichiers ou clés de registre) en vue de les protéger. Chaque ACL contient des entrées de contrôle d'accès (ACE, Access Control Entry), lesquelles associent une entité de sécurité (un compte d'utilisateur, un groupe de comptes, une entité système, etc.) à une règle définissant les autorisations que la ressource concernée octroie ou refuse.

Il existe deux types de liste de contrôle d'accès, celles dites système (SACL, System Access Control List), et celle dites discrétionnaire (DACL, Discretionary Access Control List). Les DACL permettent de contrôler les accès aux objets et les SACL permettent de contrôler quelles opérations enregistrer dans le cadre de l'audit des accès aux objets.

Tableau 12.6. Fonctions Windows concernant les ACL

GetAclInformation	Retourne des informations concernant une ACL
InitializeAcl	Crée une nouvelle ACL
IsValidAcl	Vérifie la validité d'une ACL

Une ACL se compose des attributs suivants :

- *Niveau de révision (AclRevision)* Numéro de version de l'ACL. Une valeur ACL_REVISION indique une ACL générique. Une valeur ACL_REVISION_DS indique une ACL pour Active Directory. Toutes les ACE utilisées dans un ACL doivent faire mention du même niveau de révision.
- *Taille (AclSize)* Spécifie la taille, en octets, de l'ACL. Cette valeur inclut à la fois la structure ACL et tous ses ACE.
- *Compteur d'ACE (AceCount)* Mémorise le nombre d'ACE que contient un ACL.

ACE

Une entrée de contrôle d'accès (ACE, Access Control Entry) figure dans une ACL et stocke les permissions pour une entité de sécurité donné (un compte d'utilisateur, un groupe de comptes, une entité système), définissant également les propriétés d'audit pour ce compte sur l'objet en question. La combinaison des droits d'accès accordées par les diverses ACE forme l'ensemble des droits d'accès accordées par une ACL.

Dans une DACL, chaque ACE est composée des informations que voici :

- Un SID qui identifie l'ayant droit (utilisateur ou groupe) auquel l'ACE s'applique
- Un masque d'accès qui détermine les différents types d'accès possibles (Lecture, Écriture, etc. - en fonction du type de l'objet),
- Le type de l'ACE, dont il ressort si l'Accès est autorisé ou refusé
- Différents flags définissant les règles d'héritage de l'ACE

Une ACE peut être positive ou négative. Une ACE positive indique que l'utilisateur ou le groupe dispose des autorisations spécifiques alors qu'une ACE négative indique que les autorisations sont refusées. Si l'utilisateur ne figure pas dans l'ACL, directement ou en tant que membre d'un groupe, l'accès est refusé. Notez que, quand le système procède à une vérification des accès, les ACE négatives supplantent automatiquement celles positives. De ce fait, les permissions Refuser ont toujours la priorité sur les permissions Autoriser.

Au sens large, on dénombre six types d'ACE, dont trois sont pris en charge au niveau de tous les objets sécurisables (access allowed, access denied et system audit), et les trois autres (allowed-object, denied-object et system-audit-object) qui sont intelligibles uniquement dans la perspective Active Directory.

Les ACE utilisées dans les DACL ont un certain de flags qui contrôlent et induisent différents comportements vis-à-vis de l'héritage. Certains espaces de noms d'objet ont des *objets conteneur* et des *objets feuille* (qui pour des raisons de commodité sont alors simplement nommés *objets*). Un conteneur peut renfermer d'autres objets conteneur et des objets feuille, qui sont ses *objets enfant*. Exemples de conteneurs : répertoires dans l'espace de noms du système de fichiers, clés dans l'espace de noms du registre, et d'autres. Le tableau qui suit énumère les règles d'héritage qui font surface en tant que flags ACE.

Tableau 12.7. Règles d'héritage ACE

Flag	Règle d'héritage
CONTAINER_INHERIT_ACE	Les objets enfant qui sont des conteneurs, par exemple les répertoires, héritent l'ACE en tant qu'ACE effective.

INHERIT_ONLY_ACE	L'ACE ne s'applique pas à l'objet auquel il est attaché mais peut être hérité par les objets enfants.
NO_PROPAGATE_INHERIT_ACE	Empêche l'ACE d'être héritée par les générations suivantes d'objets.
OBJECT_INHERIT_ACE	Les objets enfant qui ne sont pas des conteneurs, par exemple les fichiers, héritent l'ACE en tant qu'ACE effective. Les objets enfant qui sont des conteneurs héritent l'ACE en tant que ACE inherit-only, sauf si le flag NO_PROPAGATE_INHERIT_ACE est également positionné.

Par exemple, si l'on se place dans une perspective de niveau fichier, les combinaisons suivantes sont envisageables : ce dossier seulement (NO_PROPAGATE_INHERIT_ACE) ; ce dossiers, les sous-dossiers et les fichiers (CONTAINER_INHERIT_ACE et OBJECT_INHERIT_ACE) ; ce dossier et ses sous-dossiers (CONTAINER_INHERIT_ACE) ; ce dossier et les fichiers (OBJECT_INHERIT_ACE) ; les sous-dossiers et les fichiers seulement (INHERIT_ONLY_ACE, CONTAINER_INHERIT_ACE et OBJECT_INHERIT_ACE) ; les sous-dossiers seulement (INHERIT_ONLY_ACE et CONTAINER_INHERIT_ACE) ; les fichiers seulement (INHERIT_ONLY_ACE et OBJECT_INHERIT_ACE).

DACL

Une liste de contrôle d'accès discrétionnaire (DACL, Discretionary Access Control List) gère les permissions sur un objet sécurisé, spécifiant qui est autorisé et à qui il est interdit d'accéder à un objet.

Les DACL contiennent des entrées de contrôle d'accès (ACE) qui fonctionnent comme des enregistrements pour chaque utilisateur ou groupe d'utilisateur désigné par son SID. Si aucune DACL ne figure dans un descripteur de sécurité (DACL null), alors tout le monde tout le monde bénéficie d'un accès complet à l'objet. Si la DACL est vide (c'est-à-dire qu'elle a 0 ACE), alors personne n'a accès à l'objet.

SACL

Une liste de contrôle d'accès système (SACL, System Access Control List) contrôle la génération de messages d'audit lors des tentatives d'accès à un objet sécurité. À l'instar des DACL, les SACL contiennent des ACE qui définissent les règles d'audit pour une ressource donnée.

La capacité à obtenir ou définir la SACL d'un objet est contrôlée par un privilège (SeSecurityPrivilege) dont sont généralement dotés uniquement les administrateurs système

SID (*Security Identifier*)

Pour identifier les ressources et les personnes sur une machine locale ou sur un réseau, Windows repose sur l'emploi de valeurs spéciales, nommées identifiants de sécurité (SID, Security Identifiers), qui sont à la source d'un moyen commode de distinguer sans ambiguïté les entités d'un système. Les utilisateurs et leurs processus, les groupes locaux, les groupes de domaine, les ordinateurs locaux, les domaines et membres de domaine, tout ce qui peut être authentifié par le système d'exploitation a un SID qui lui est associé. L'environnement Microsoft se base sur cette information chaque fois qu'il est nécessaire de réaliser des contrôles d'accès, par exemple pour réaliser une ouverture de session locale ou ouvrir des sessions vers le domaine.

Les SID existent depuis la toute première version de Windows NT et constituent une pierre angulaire importante de la protection de ces systèmes. Les algorithmes liés fonctionnent en lien avec des composants spécifiques de l'autorisation qui, pensés pour fournir un environnement informatique plus sécurisé, agissent dès qu'il est question de vérifier si une entité demandant d'accéder à une ressource a les droits nécessaires pour le faire.

Chaque SID se présente sous forme d'une chaine alphanumérique construite en partie sur le principe de l'imprévisibilité, d'où le caractère unique de chaque représentation. Étant donnée l'étendue que peut couvrir un

SID (il s'agit généralement d'un nombre très grand), couplé au fait que Windows génère des valeurs purement aléatoires au sein de chaque SID, il est virtuellement impossible à Windows d'émettre le même SID deux fois sur des machines ou des domaines de par le monde. Il est plus efficace pour le système d'exploitation d'utiliser le SID, plutôt que le nom, pour identifier un utilisateur (ou toute autre personnification) car les noms, susceptibles déjà de ne pas être uniques, peuvent en outre être modifiés.

Au plus haut niveau d'abstraction, un SID est une valeur numérique de longueur variable qui se compose d'un numéro de révision, d'une valeur autorité identificatrice sur 48 bits et d'un certain nombre de valeurs de valeurs sous-autorité sur 32 bits. Les valeurs individuelles d'un SID correspondent donc à ceci :

- *Révision* Indique la version de la structure de données (format binaire) SID qui est utilisée dans un SID spécifique. La structure employée par Windows jusqu'à ce jour comporte un numéro de révision égal à 1.

- *Autorité identificatrice* Identifie l'agent émetteur du SID, généralement un système local ou un domaine Windows.

- *Sous-autorités* Identifie les sous-autorités de confiance (RID, Relative ID) relatives à l'autorité ayant produit le SID. Les RID désignent des comptes ou groupes à l'intérieur d'un domaine. L'identifiant RID est unique seulement au sein d'un domaine, tandis que le SID, de par ses aspects cumulatifs, est unique dans toutes les installations de Windows.

Les composants d'un SID sont plus faciles à visualiser lorsque les SID sont converties à partir d'un fichier binaire dans un format de chaîne à l'aide de la notation standard, soit S-R-X-Y1-Y2-Yn. Dans cette notation, la lettre S sert à montrer que la série alphanumérique qui suit est un SID, R représente un numéro de révision, X indique la valeur autorité et Y une série de valeurs sous-autorité, où n est le nombre de valeurs. L'exemple suivant est à considérer :

Dans ce SID, le numéro de révision est 1 et la valeur de l'autorité identificatrice est 5 (autorité de sécurité Windows) ; le reste du SID est composé de deux valeurs de sous-autorité : 32 (SECURITY_BUILTIN_DOMAIN_RID) et 544 (DOMAIN_ALIAS_RID_ADMINS). Lorsque vous installez Windows, le programme d'installation émet un SID pour l'ordinateur, de sorte qu'il est impossible pour deux ordinateurs du réseau d'avoir le même SID.

Windows assigne des SID aux comptes locaux de la machine. Lorsque vous créez un compte d'utilisateur ou de groupe, Windows lui assigne un SID unique, lequel reste associé à ce seul compte jusqu'à sa suppression. Si vous renommez un compte, le SID ne change pas, et tous les attributs afférents (droits et autres autorisations) sont préservés. Si vous supprimez un compte , toutes les attributions de sécurité associées à ce compte sont également supprimées. Windows ne réutilisant pas le SID qui était attribué à un compte, même si des comptes partagent le même nom, ils ne partagent pas le même SID. Chaque SID de compte local est basé sur le SID de l'ordinateur source et a un RID à la fin. Les RID des comptes utilisateur et des groupes commencent à 1000 et augmentent de 1 pour chaque nouvel utilisateur ou groupe.

Pour les quelques comptes créés automatiquement lors de l'installation du système, Windows crée des SID qui se composent d'un SID de machine ou de domaine complété par un RID prédéfini. Par exemple, le RID du compte Administrateur est 500 et le RID du compte Invité est 501. Le compte administrateur local d'un ordinateur a comme base le SID machine, auquel s'ajoute le RID 500. Cela donne :

S-1-5-21-2647556361-93858140-2277411872-500

Les valeurs de RID inférieures à 1000 sont destinées à un usage interne de la part du système d'exploitation, qui les utilise dans ce contexte pour identifier des sujets connus. Les comptes d'utilisateurs créés par la suite reçoivent des RID dont la valeur de départ est fixée à 1000.

Enfin, Winlogon créee un SID unique pour chaque session interactive. Le SID d'une session interactive est de la forme S-1-5-5-0, complétée par un RID généré aléatoirement.

Le liste suivante énumère les principales interfaces que fournit Windows en matière de SID.

- AllocateAndInitializeSid Alloue et initialise un SID avec jusqu'à 8 sous-autorités

- CopySid Effectue une copie d'un SID dans un tampon mémoire contrôlé par l'appelant
- CreateWellKnownSid
- FreeSid Supprime la mémoire dévolue à un SID (allouée au préalable via AllocateAndInitializeSid)
- GetLengthSid Retourne la taille en octets d'une structure SID
- IsValidSid Entérine, ou au contraire, sanctionne la validité d'un SID. Les conditions de validation d'une telle structure portent sur le numéro de révision, qui doit être conforme à une valeur donnée, et sur le nombre de sous-autorités, qui doit être inférieur au nombre maximal.
- LookupAccountSid Retourne le nom du compte auquel un SID réfère
- ConvertSidToStringSid Convertit une représentation SID en une chaine de caractères

En pratique : visualisation des SID de compte. L'utilitaire PsGetSid permet de traduire les noms des comptes de machine et d'utilisateur en les SID associés, et vice-versa.

Si vous exécutez PsGetSid sans options, il affiche le SID assigné à l'ordinateur local.

```
c:\>psgetsid

SID for \\lain:
S-1-5-21-495418598-4048141043-142412758
```

Pour connaitre le SID d'un compte d'utilisateur, spécifiez le nom de l'utilisateur comme argument de PsGetSid :

```
c:\>psgetsid

SID for lain\arnaud:
S-1-5-21-495418598-4048141043-142412758-1002
```

Il est également facile de voir la représentation SID de vos comptes avec l'utilitaire Whoami.

Pour afficher le SID de l'utilisateur actuellement connecté sur le système local, saisissez whoami /user.

Afin d'afficher les identificateurs des groupes, saisissez : whoami /groups.

Pour afficher le nom de l'utilisateur actuel, les groupes auxquels il appartient ainsi que les SID et les privilèges de l'utilisateur actuel, saisissez : whoami /all.

Une autre possibilité en ce qui concerne la lecture des SID repose sur l'utilisation de la console WMI, où l'alias useraccount fournit une passerelle vers la gestion des comptes. Pour afficher les noms et les SID des utilisateurs locaux, saisissez la commande wmic useraccount get name,sid. Vous devriez vois quelque chose ressemblant à ce qui suit :

```
C:\>wmic useraccount get name,sid
Name            SID
Administrateur  S-1-5-21-495418598-4048141043-142412758-500
arnaud          S-1-5-21-495418598-4048141043-142412758-1002
Invité          S-1-5-21-495418598-4048141043-142412758-501
```

La raison principale pour Windows d'utiliser des SID, plutôt que des noms, afin de répertorier les diverses entités établies dans le modèle de sécurité réside dans l'impossibilité de donner aux noms un caractère définitif et irréversible, ainsi que de leur faire porter une grande quantité d'informations. Les noms de compte et les noms complets associés à des utilisateurs peuvent par exemple être modifiés, soit par décision de l'administrateur, soit s'il dispose des droits nécessaires pour le faire, par l'utilisateur lui-même. À plus grande échelle, quand un ensemble de machines se partagent des informations d'annuaire (domaine), les noms seuls ne peuvent servir à

faire la différence entre comptes et groupes (un groupe d'utilisateurs est un ensemble de comptes d'utilisateurs) : une machine peut par exemple abriter un compte d'utilisateur x, et une autre machine un groupe homonyme. (Notez que pour éviter toute confusion, Windows interdit que les comptes locaux d'une machine aient le même nom). En outre, du fait de la prise en compte de solutions multilingues, certaines désignations varient d'un système à l'autre. Par exemple, toutes les versions américaines standard de Windows (anglais américain) sont distribuées avec un groupe prédéfini nommé Administrators et dont le SID est S-1-5-32-544. Sur les systèmes en langue française, le même groupe est appelé Administrateurs.

SID (*Security Identifier*)

Un certain nombre de SID couramment employés, appelés SID bien connus, ont des valeurs qui restent constantes sur toutes les installations de Microsoft Windows.

Tableau 12.8. SID bien connus

SID	Groupe	Description
S-1-1-0	Tout le monde	Tous les utilisateurs.
S-1-2-0	Local	Utilisateurs connectés localement au système.
S-1-5-2	Réseau	Groupe qui inclut tous les utilisateurs connectés par le biais d'une connexion réseau.
S-1-5-3	Tâche	Inclut tous les utilisateurs qui ont ouvert une session via une file d'attente de tâches, par exemple les travaux du planificateur de tâches.
S-1-5-6	Service	Groupe qui inclut toutes les entités de sécurité qui se sont connectés en tant que service. les utilisateurs connectés par le biais d'une connexion réseau.
S-1-5-11	Utilisateurs authentifiés	Groupe qui inclut tous les utilisateurs et ordinateurs qui ont ouvert une session authentifiée.
S-1-5-7	Anonyme	Groupe qui inclut tous les utilisateurs connectés de manière anonyme.
S-1-5-18	Système local	Compte de service utilisé par le système d'exploitation.
S-1-5-32-578	Administrateurs Hyper-V	Groupe dont les membres bénéficient d'un accès complet et illimité aux fonctionnalités Hyper-V.

Niveaux d'intégrité

Pour utiles et dignes d'intérêt qu'elles soient, les politiques de contrôle d'accès en œuvre dans les systèmes d'exploitation ne répondent que partiellement aux problématiques auxquelles elles doivent faire face, et d'autant plus à l'échelle des usages modernes, passent à côté de certains aspects pourtant cruciaux, par exemple le fait de tromper l'utilisateur de sorte à l'amener à autoriser l'exécution d'un logiciel malveillant. En basant les moyens de limiter l'accès aux objets uniquement sur l'identité d'un sujet et les habilitations subséquentes (soit tout le propos du contrôle d'accès), il est dans ce contexte difficile de hiérarchiser les actions effectuées depuis un même compte d'utilisateur. Pour ces raisons, Windows met en œuvre le contrôle d'intégrité obligatoire (MIC, Mandatory Integrity Control), une méthode de contrôle d'accès aux ressources qui assure une protection supplémentaire aux objets sécurisables, et ce faisant réduit le risque qu'une exploitation puisse modifier le système ou endommager les fichiers de données des utilisateurs.

Le contrôle d'intégrité obligatoire a été introduit sous Windows Vista et fonctionne de manière complémentaire au mécanisme de permissions existant dans le système d'exploitation. Cette protection est utilisée, entre autre, par le contrôle de compte d'utilisateur (UAC), par le mode protégé d'Internet Explorer (PMIE), et pour l'isolation

des privilèges de l'IHM (UIPI). Au plus haut niveau d'abstraction, il permet au composant gestionnaire de la sécurité (SRM) d'en savoir plus sur la nature d'un appelant. Les principes et algorithmes sous-jacents dérivent du modèle de Biba, ou modèle d'intégrité de Biba, développé par Kenneth J. Biba en 1977.

L'orientation principale recherchée par le mécanisme d'intégrité est de limiter les permissions octroyées aux applications s'exécutant sous un même compte d'utilisateur, donc avec le même profil et des implications analogues en matière de sécurité. Le contrôle d'intégrité intègre dans ce but la notion d'évaluation de la loyauté dans le système d'exploitation. Les éléments dont le degré de loyauté est bas ne peuvent pas modifier les données de degré supérieur. Les éléments dont le degré de loyauté est élevé ne peuvent pas être contraints à faire confiance à des données de degré inférieur.

Lorsqu'un utilisateur ouvre une session, Windows lui assigne un jeton de sécurité. Celui-ci contient une étiquette d'intégrité qui détermine le niveau d'accès auquel lui-même, et par conséquent l'utilisateur, peut prétendre. Les objets à protéger, tels que fichiers, dossiers, canaux, processus, threads, stations fenêtre, clés de Registre et objets de répertoire, possèdent de leur côté des descripteurs de sécurité qui définissent le niveau d'intégrité requis pour l'accès à l'objet.

Au cours d'un contrôle d'accès, avant la vérification du droit d'accès de l'utilisateur par l'intermédiaire de la DACL, Windows vérifie le niveau d'intégrité de l'utilisateur, et le confronte au niveau d'intégrité de l'objet demandé. Si le niveau de l'utilisateur l'emporte sur celui de l'objet (c'est à dire s'il est égal ou supérieur), l'utilisateur sera autorisé à interagir avec l'objet selon les autorisations définies par la DACL. Dans le cas contraire, donc quand un objet à un niveau d'intégrité strictement supérieur à qui le demande, le mécanisme de vérification se réfère alors à un ensemble de politiques standards afin de déterminer les restrictions qui s'appliquent.

La liste ci-après décrit les différents niveaux d'intégrité par ordre croissant :

- *Non approuvé* Niveau attribué par défaut aux processus connectés de manière anonyme.
- *Faible* Niveau attribué par défaut aux processus qui interagissent avec Internet.
- *Moyen* Niveau attribué par défaut aux comptes d'utilisateurs standard et à tout objet non explicitement désigné avec un niveau d'intégrité inférieur ou supérieur.
- *Élevé* Niveau attribué par défaut aux comptes d'administrateurs et aux processus qui demandent à s'exécuter avec des droits d'administration.
- *Système* Niveau attribué par défaut aux services de noyau et aux services principaux Windows.
- *Programme d'installation* Niveau utilisé par les programmes d'installation des logiciels. Les objets dotés du niveau d'intégrité Programme d'installation peuvent installer, modifier et désinstaller tous les autres objets.

Les objets dépourvus d'étiquette d'intégrité sont traités au niveau moyen par le système d'exploitation, ce qui constitue généralement une mesure suffisante pour assurer une protection efficace. Considérez à titre d'exemple le scénario suivant : vous recevez une pièce jointe accompagnant un message de courrier électronique. Lors de l'enregistrement de cette dernière sur le système de stockage, elle est répertorié avec un niveau d'intégrité bas, vu qu'elle provient d'Internet - considéré comme non sûr. Lorsque vous exécutez la pièce jointe, le processus qui en résulte fonctionne au niveau d'intégrité bas, conformément à ce que l'étiquette de l'objet stipule. Etant donné le niveau auquel vos données sont associés (en l'occurence moyen), elles sont de la sorte protégées de toute écriture malintentionnée émanant de la pièce jointe.

Le contrôle d'intégrité obligatoire permet à Windows de contrôler les communications entre les processus en définissant divers niveaux dans lesquels les processus s'exécutent. De ce fait, chaque processus dispose d'une mention d'intégrité, laquelle se situe au niveau de la structure jeton d'accès. Hormis spécification du contraire, un processus hérite du niveau d'intégrité de son parent. Un processus peut donner naissance à un autre processus en lui donnant explicitement un niveau d'intégrité. Les procédés utiles en la matière incluent la duplication du jeton d'accès (fonction Windows DuplicateTokenEx), l'incorporation à ce jeton de nouvelles

informations d'intégrité (SetTokenInformation), et finalement, la création de processus s'exécutant dans un contexte de sécurité spécifique (CreateProcessAsUser).

Pour faciliter la mise à niveau à partir des versions précédentes de Windows, dont les clés de registre et les fichiers n'intègrent pas d'informations d'intégrité, tous les objets ont une côte implicite, laquelle correspond par défaut au niveau moyen. Les concepteurs d'application n'ont donc pas à se soucier de ce problème éventuel.

De nombreux composants dont l'intégrité est élevé (niveau haut ou système) créent des objets de niveau moyen. Loin d'être un paradoxe, ce phénomène s'explique par une raison très simple, relative au contrôle de comptes d'utilisateur (UAC) : si le niveau d'intégrité d'un objet était inconditionnellement le même que celui de l'entité créatrice, un administrateur qui désactive puis réactive UAC se verrait éventuellement déposséder d'un certain nombre de ses droits, incapable auquel cas de modifier les paramètres de registre ou les fichiers crées lors de l'exécution au niveau d'intégrité élevé.

Le noyau Windows assigne automatiquement un niveau d'intégrité à certains types d'objet, dont les processus, threads et jobs, et les jetons d'accès, de sorte à empêcher tout processus exécuté sous le même compte, mais avec un niveau d'intégrité moindre, d'accéder à ces instances ou d'altérer leur comportement. Cela prend en compte, par exemple, l'injection de DLL et d'autres attaques du même style.

Certains dossiers ont une étiquette obligatoire de faible niveau d'intégrité. Un processus à faible niveau d'intégrité peut traiter et modifier les fichiers des dossiers à faible niveau d'intégrité. Ainsi, le dossier Fichiers Internet temporaires contient un dossier appelé Faible, qui est un dossier à faible niveau d'intégrité. Ainsi, si vous téléchargez un logiciel à partir du Web, l'enregistrez dans le dossier Fichiers Internet temporaires, puis l'exécutez, l'application téléchargée s'exécute elle aussi avec le niveau d'intégrité inférieur. L'application n'a pas accès aux données de l'utilisateur, ce qui ajoute une couche de sécurité supplémentaire pour le contenu téléchargé à partir d'Internet.

Si des programmes malveillants sont définis avec un niveau d'intégrité élevé tel que Programme d'installation approuvé ou Système, les comptes Administrateur ne disposent pas des niveaux d'intégrité suffisants pour supprimer ce programme du système. Dans cette éventualité, l'utilisation du droit Modifier un nom d'objet est autorisée pour permettre le changement de nom de l'objet. Toute personne dotée du droit Modifier un nom d'objet (SeRelabelPrivilege) peut ainsi hausser ou diminuer le niveau d'intégrité d'un objet de sorte qu'il puisse être supprimé par du code d'intégrité inférieure.

LSASS (*Local Security Authority Subsystem Service*)

Autre grande figure de la protection logicielle dans Microsoft Windows, le sous-système d'autorité de sécurité locale (LSASS, Local Security Authority Subsystem Service) est un dispositif de sécurité dont les ramifications se répartissent entre tous les services d'authentification et d'autorisation interactives de l'ordinateur. Il est chargé de la stratégie de sécurité du système local, de l'authentification des utilisateurs et de l'envoi des messages d'audit au service de journalisation des événements.

La liste suivante énumère quels sont, dans les grandes lignes, les rôles du composant LSA. Sur le plan technique, c'est le service d'autorité de sécurité locale (Lsasrv, \Windows\System32\Lsasrv.dll), bibliothèque chargée par LSASS, qui implémente la plupart de ces fonctionnalités.

- *Gestion des stratégies de sécurité locales* Il gère et applique les stratégies de sécurité définies pour permettre aux utilisateurs locaux de s'authentifier, et met en oeuvre les mécanismes nécessaires, par exemple, l'identification des entités ayant la permission d'accéder au système et de leurs modalités d'accès, et les paramètres de l'audit de la sécurité du système.

- *Authentification des utilisateurs* Il offre un ensemble de services chargés d'authentifier les comptes accédant à l'ordinateur, incluant les stratégies de mot de passe et les privilèges accordés aux utilisateurs et aux groupes. Il est également utilisé pour traiter les demandes d'authentification via le protocole Kerberos ou le protocole NTLM dans Active Directory.

- *Création des jetons d'accès* Une fois le contrôle d'ouverture de session validé, il est alimenté en informations détaillant l'identité de l'utilisateur, données à partir desquelles il génère un jeton d'accès.

- *Contrôle des stratégies d'audit* Il gère et applique la stratégie d'audit du système local et met à jour le journal d'audit en fonction de cette stratégie lorsqu'il y a lieu de le faire.

L'autorité de sécurité locale se présente sous la forme d'un processus mode utilisateur, exécutant l'image \Windows\System32\Lsass.exe, qui héberge un certain nombre d'autres composants en lien avec l'architecture de sécurité de Windows, tous implémentés en tant que DLL : service d'autorité de sécurité locale (Lsasrv.dll), que nous avons déjà brièvement évoqué, service SAM (Samsrv.dll), serveur Active Directory (Ntdsa.dll), service d'ouverture de session réseau (Netlogon.dll), service de distribution de tickets Kerberos (Kdcsvc.dll), ainsi qu'un ensemble de packages d'authentification (msv1_0.dll et Kerberos.dll).

LSASS, lors de l'ouverture de session, a pour mission de gérer une authentification sécurisée. Ainsi, lorsqu'une entité souhaite se connecter à un système, la demande est relayée à LSASS qui va devoir valider l'authentification demandée. (En interne, cette validation est réalisée, non par LSASS lui-même, mais par l'intermédiaire de composants spécialisés. Voir sur le sujet la section Packages d'authentification). Si la demande est acceptée, autrement dit si le contrôle d'accès a débouché sur une issue favorable, LSASS demande au SRM de procéder à la création d'un jeton d'accès.

En plus de l'authentification des comptes accédant à un ordinateur, LSASS est également chargé de valider les connexions des utilisateurs distants. Ainsi, lorsqu'un utilisateur tente de s'authentifier auprès d'un domaine, un contrôleur de domaine est contacté pendant la phase d'authentification. Celui-ci contrôle le nom et le mot de passe d'ouverture de session et, si le contrôle est positif, renvoie alors, entre autres, la liste des groupes auquel l'utilisateur appartient. Le sous-système local LSASS du poste où l'utilisateur s'est connecté crée à ce moment à ce moment un jeton d'accès qui contient les données d'identification de sécurité de cet utilisateur. Ce jeton contient l'identifiant de sécurité unique (SID) du compte utilisateur ainsi que les SIDs de tous les groupes dont il est membre dans le domaine dans lequel il s'authentifie.

Stratégies de restriction logicielle

Mues essentiellement par l'optique de conférer plus de robustesse aux environnements informatiques contrôlés par Windows - en l'occurence de mieux les armer contre les virus, vers et autres maliciels, les stratégies de restriction logicielle fournissent aux administrateurs un moyen commode d'identifier et de limiter les logiciels susceptibles d'être exécutés sur leurs systèmes.

Utilisées principalement en vue d'empêcher l'utilisation de logiciels non autorisés pour diverses raisons (jeux, applications pair-à-pair, logiciels dont les problèmes sont connus ou reconnus potentiellement dangereux, applets ActiveX, etc.), les stratégies de restriction logicielle, qui font partie des stratégies de groupe, le sont aussi quelquefois afin de respecter des contraintes de licences applicatives, ou d'attribution de droits d'accès à des utilisateurs particuliers et à leurs programmes. Elles peuvent encore se montrer utiles pour résoudre d'éventuels problèmes liés à des utilisateurs malveillants (intentionnellement ou non). Par exemple, si des administrateurs du domaine s'aperçoivent que des virus transitent via les courriers électroniques des utilisateurs, ils peuvent appliquer une stratégie qui n'autorise pas certains types de fichiers à s'exécuter dans les pièces jointes des messages.

Les stratégies de restriction logicielle admettent deux utilisations distinctes, l'un et l'autre mettant l'accent sur différents critères. Dans le premier de ces scénarios, les stratégies de restriction sont utilisées afin d'empêcher les utilisateurs d'exécuter des applications spécifiques. En règle générale, cela concerne surtout les utilisateurs et les ordinateurs types, pour lesquels il n'est pas requis d'exigences très importantes en matière de sécurité. Dans le second scénario, les stratégies de restriction logicielle sont employées de sorte à créer une configuration fortement restreinte, dans laquelle seule l'exécution d'applications clairement identifiées est autorisée. En pratique, cette configuration permet de toucher les comptes d'utilisateurs et d'ordinateurs de faible sécurité.

La fonctionnalité Stratégies de restriction logicielle, si elle constitue un atout certain parmi les technologies de contrôle d'applications intégrées à Windows, ne se substitue pas à l'utilisation de programmes antivirus, pare-feu

ou autres utilitaires du même style. Exercées sans discernement, elles peuvent conduire à un alourdissement de la charge de travail de l'administrateur et du système (en ce qui concerne ce dernier à cause de l'accroissement des opérations nécessaires à l'accomplissement de la tâche demandée), et sérieusement irriter les utilisateurs.

Règles de stratégies et niveaux de sécurité

Sur le plan fonctionnel, chaque stratégie de restriction logicielle est articulée selon plusieurs directions complémentaires, à quoi correspond une règle par défaut, qui s'applique à toutes les applications non identifiées par une règle, et un ensemble d'exceptions à la règle par défaut, qui régissant les modalités d'exécution des logiciels identifiés. A cette infrastructure s'ajoutent trois niveaux de sécurité, ordonnés dans la liste qui suit du plus permissif au plus strict.

- *Non restreint* Permet aux programmes de s'exécuter. Les droits d'accès au logiciel sont déterminés par les droits d'accès de l'utilisateur.
- *Utilisateur Standard* Permet d'exécuter des programmes en tant qu'utilisateur n'ayant pas les droits d'administrateur, mais pouvant accéder aux ressources accessibles aux utilisateurs normaux.
- *Rejeté* Le logiciel ne s'exécute pas, quels que soient les droits d'accès de l'utilisateur.

Lors de la configuration initiale d'une stratégie, l'administrateur commence par déterminer le niveau de sécurité par défaut à laquelle elle est associée. Si le niveau de sécurité par défaut est Non restreint, toutes les applications peuvent s'exécuter, à l'exception de celles faisant l'objet de restrictions instaurées via des règles supplémentaires. Dans un tel schéma, sauf les fichiers expressément interdits, tout le reste est autorisé ; seules s'appliquent par conséquent les permissions fichiers relatives au fichier exécutable et les droits d'accès octroyés à l'utilisateur. Le niveau de sécurité Rejeté, quant à lui, impose une vision plus contraignante des interactions entre logiciels et stratégies, et implique qu'aucun logiciel n'est autorisé à s'exécuter, sauf ceux spécifiquement prévus, désignées en tant que tel par l'intermédiaire de règles spécifiques.

Toute chose ayant ses avantages et inconvénients propres, les niveaux de sécurité implantés sur les stratégies de restriction logicielle ne dérogent pas à la règle ; il est dans tous les cas importants de comprendre dans quelles mesures ces options influent sur l'utilisabilité du système. (Notez que cela vaut également pour l'ensemble des paramètres ayant un impact sur la stratégie de groupe). A cet égard, des erreurs commises au niveau de la conception ou de la mise en œuvre de cette fonctionnalité risquent de mener à un constat d'inefficacité - quand les critères sur lesquels reposent la stratégie sont trop laxistes, ou de créer beaucoup de frustration chez les utilisateurs - critères trop restrictifs.

Exceptions

Une stratégie de restriction logicielle utilise les quatre types de règles suivants pour identifier une application :

- *Règle de hachage* Identifie une application par des caractéristiques uniques de fichiers traduites en un algorithme de hachage. Cet algorithme est tel que deux documents différents ne peuvent avoir la même empreinte, qui est par ailleurs la même quel que soit le nom ou l'emplacement du fichier. Lorsqu'une règle de hachage est créée, une suite d'octets identifiant de façon unique l'exécutable du logiciel est générée. Ainsi, lors de l'exécution d'un logiciel par un utilisateur, le hachage de celui-ci est préalablement comparé aux règles de hachage des stratégies de restriction logicielle. Toute modification du fichier (Service Pack, Hotfix, etc.) entraine par nature un nouveau hachage, ce qui risque de rendre inopérante la stratégie de restriction logicielle.
- *Règle de certificat* Identifie une application en fonction du certificat numérique qui l'accompagne, lequel se présente sous la forme d'une signature définie par un éditeur de sorte à authentifier ledit logiciel. La procédure de certification de logiciel est relativement lourde à mettre en place, mais offre une sécurité très forte. Comme pour les règles de hachage, la certification est indépendante du nom et de l'emplacement du fichier.
- *Règle de chemin d'accès* Identifie une application par le biais d'un chemin d'accès dans le système de fichiers ou d'une localisation dans le Registre. Il est possible d'utiliser aussi bien des chemins UNC que des chemins locaux.

- *Règle de zone* Identifie une application en fonction de la zone réseau à partir de laquelle elle a été téléchargée : Internet, Intranet local, Sites approuvés, Sites sensibles, Ordinateur local. Cette règle ne s'applique qu'aux logiciels installés par Windows Installer.

Conflits de règles

Lorsqu'une stratégie de restriction logicielle contient plusieurs règles, Windows les traite dans l'ordre de la liste précédente. S'il existe, par exemple, une règle de hachage avec un niveau de sécurité Non restreint pour un logiciel qui réside dans un dossier pour lequel une règle de chemin d'accès est assignée avec un niveau de sécurité Rejeté, le programme s'exécute. La règle de hachage ayant une plus haute priorité que la règle de chemin d'accès, elle prend le pas sur cette dernière.

Si des règles conflictuelles s'appliquent au même programme, c'est toujours la plus spécifique qui l'emporte. Par exemple, s'il existe une règle de chemin d'accès pour C:\Windows\ avec un niveau de sécurité Rejeté, mais aussi dans le même temps une règle de chemin d'accès pour C:\Windows\System32\ avec un niveau de sécurité Non restreint, la règle qui a la chemin d'accès le précis est prioritaire. Les logiciels incorporés à C:\Windows\ ne s'exécutent pas, contrairement aux programmes du dossier C:\Windows\System32.

Si deux règles identiques avec des niveaux de sécurité différents sont appliquées à un logiciel, la règle la plus conservatrice est prioritaire. Par exemple, si deux règles de hachage, l'une avec un niveau de sécurité Rejeté, l'autre avec un niveau de sécurité Non restreint, sont appliquées au même logiciel, la règle avec un niveau de sécurité Rejeté est prioritaire et le programme ne s'exécute pas.

Paramètres de stratégies

Plusieurs paramètres de stratégie globaux influencent la façon dont le système interagit avec les stratégies de restriction logicielle.

- *Contrôle obligatoire* Définit si les stratégies de restriction s'appliquent aux bibliothèques (DLL), et si elles s'appliquent ou non aux administrateurs locaux.
- *Types de fichiers désignés* Définit les extensions des fichiers considérés comme étant du code exécutable.
- *Editeurs approuvés* Décide qui peut sélectionner les éditeurs de certificats approuvés.

Configurer les stratégies de restriction logicielle

Les stratégies de restriction logicielle peuvent être définies à l'aide de GPO locaux ou de domaines pour des ordinateurs ou des utilisateurs individuels. Pour configurer des stratégies de restriction logicielle, ouvrez un GPO dans l'Editeur d'objets de stratégie de groupe et sélectionnez le noeud Configuration ordinateur\Paramètres Windows\Paramètres de sécurité\Stratégies de restriction logicielle. Si le noeud ne contient pas d'objets, des stratégies de restriction logicielle n'ont pas été définies dans ce GPO. Pour créer une stratégie de restriction logicielle, cliquez avec le bouton droit de la souris sur Stratégies de restriction logicielle puis faites Nouvelle stratégie de restriction logicielle.

Vous pouvez afficher le niveau de sécurité par défaut, qui définit la manière dont Windows répond aux logiciels qui ne disposent d'aucune exception, en sélectionnant le noeud Niveaux de sécurité sous le noeud Stratégies de restriction logicielle. Trois niveaux de sécurité apparaissent alors : Rejeté, Utilisateur standard et Non restreint. Le niveau par défaut en cours est signalé à l'aide d'une coche sur l'icône. Si aucune modification n'a été apportée, Non restreint sera sélectionné ; tout logiciel peut par conséquent s'exécuter.

Windows Defender

Windows Defender est une fonctionnalité intégrée à Windows de sorte à protéger le système des logiciels malveillants et/ou potentiellement indésirables. Cette application, par ailleurs bâtie sur les fondations du code

obtenu suite au rachat de l'éditeur Giant Software par Microsoft (2004), succède à Microsoft AntiSpyware pour la sécurisation du poste de travail.

Defender est disponible dans toutes les versions de Windows depuis Windows Vista. Dans sa version sous Vista et 7, ledit logiciel est relativement limité, son spectre d'analyse se limitant en grande partie aux logiciels espions. Sous Windows 8 et supérieur, Defender prend en charge un large éventail de technologies, y compris la reconnaissance par signature, l'analyse heuristique et le blocage comportemental. La protection couvre ainsi les virus, les logiciels espions, les logiciels de publicité, les chevaux de Troie et d'autres logiciels du même acabit.

Actifs par défaut, les mécanismes de protection incluent à Windows Defender tentent de protéger l'ordinateur des menaces avant qu'elles aient pu porter atteinte à l'intégrité du système d'exploitation - afin de le compromettre ou de le détourner de certains de ses objectifs. Windows Defender fonctionne à partir de signatures, en utilisant des descriptions qui identifient de manière unique des applications reconnues nuisibles. Il obtient régulièrement de Microsoft de nouvelles signatures afin de pouvoir suivre rapidement l'évolution des différents dangers existants. (Microsoft fournit les mises à jour Windows Defender via Windows Update, via les Mises à jour automatiques et via Windows Server Update Service.)

Windows Defender est muni d'un système de détection dit heuristique pour tenter de débusquer de nouvelles menaces. D'une manière générale, la méthode heuristique consiste à exploiter des connaissances acquises avec l'expérience pour résoudre un problème. Les logiciels de sécurité heuristiques recherchent pour la plupart des schémas, des comportements et des mécanismes que l'on estime nuisibles ou indésirables. La protection en temps réel de Windows Defender surveille les points névralgiques du système d'exploitation dans l'attente de modifications habituellement apportées par des logiciels malveillants. Cette protection analyse tous les fichiers lorsqu'ils sont ouverts et surveille également un certain nombre d'emplacements du système d'exploitation. Si une application tente de modifier l'une des zones protégées du système d'exploitation, Defender demande à l'utilisateur d'effectuer l'action appropriée.

Le moteur anti-malware de Defender se nomme Microsoft Malware Protection Engine et se trouve stocké à l'emplacement c:\Program Files\Windows Defender\MsMpEng.exe. Les services de ce composant peuvent être sont sollicités soit depuis la ligne de commandes (MpCmdRun), soit via une interface graphique (MSASCui.exe). Sur ce sujet, notez que Defender affiche une interface graphique seulement quand des actions spécifiques doivent être menées, suite par exemple à la détection d'une menace dont la suppression nécessiterait le concours de l'utilisateur.

Les utilisateurs visés par Windows Defender se limitent essentiellement aux particuliers. Destiné avant tout à un usage domestique, Defender s'adapte mal à un environnement d'entreprise principalement pour deux raisons : l'intégration minimale de ce logiciel à la Stratégie de groupes et le fait qu'il ne dispose pas de console centrale d'administration et de déploiement. Afin de s'adresser au marché professionnel, Microsoft propose Forefront Client Security, une suite logicielle complète comprenant entre autres antivirus et antispyware.

Paramètrer Windows Defender

Pour configurer Windows Defender, commencez par ouvrir les Paramètres de l'ordinateur (par exemple en appuyant sur les touches Win + I), puis cliquez sur Mise à jour et sécurité. Dans le volet de gauche, effectuez un clic de souris sur le lien Windows Defender. Différents curseurs font alors leur apparition.

- *Protection en temps réel* Fonctionnalités de surveillance des contenus actifs sur la machine locale. La protection en temps réel permet de traiter les menaces éventuelles avant qu'elles ne deviennent un problème.
- *Protection dans le cloud* Fonctionnalités APT (Advanced Threat Protection) basées sur le brassage d'une vaste quantité d'informations de sécurité dans le cloud.
- *Envoi d'un échantillon* Fonctionnalités MAPS (Microsoft Active Protection Service) d'envoi des échantillons à Microsoft.

En dessous de ces options se trouve un encadré qui regroupe les informations de version concernant chacun des dispositifs impliqués dans la défense antimalware des plateformes Windows - dont Defender est sans doute la

représentation la plus visible : version du client anti-programme malveillant, des définitions du système moteur, etc.

Compatibilité avec ELAM

Windows Defender est compatible avec le dispositif de lancement anticipé de programmes anti-malware (ELAM). Les informations de démarrage du pilote correspondant sont stockées dans le Registre sous la clé HKLM\SYSTEM \CurrentControlSet\Services\WdBoot.

Configuration du pilote WdBoot dans le Registre.

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\WdBoot
    SupportElamHive    REG_DWORD    0x0
    DisplayName    REG_SZ    @%ProgramFiles%\Windows Defender\MpAsDesc.dll,-390
    ErrorControl    REG_DWORD    0x1
    Group    REG_SZ    Early-Launch
    ImagePath    REG_EXPAND_SZ    system32\drivers\WdBoot.sys
    Start    REG_DWORD    0x0
    Type    REG_DWORD    0x1
    Description    REG_SZ    @%ProgramFiles%\Windows Defender\MpAsDesc.dll,-400
    SignaturesThumbprint    REG_BINARY    067D5B9DC5627F97DCF3FEFF602A342ED698D2CC
    SignaturesVersion    REG_SZ    1.264 (0) (1.211.2635.0) (1.1.12300.0)
```

Windows Defender emploie son pilote ELAM pour examiner chaque pilote de démarrage et déterminer s'il figure parmi la liste des pilotes de confiance. Le chargeur de démarrage (Winload.exe) utilise les résultats ainsi obtenus dans le but de statuer sur la procédure à suivre, soit initialiser le pilote, ou bien ne rien faire et passer au pilote suivant.

Analyser la station de travail avec Windows Defender

Les procédés de détection de logiciels malveillants intégrés à Windows Defender, s'ils ne sont jamais plus efficaces que dans le mode de protection temps réel dudit logiciel (qui vous avertit dès que des programmes indésirables tentent de s'installer ou de s'exécuter), admettent plusieurs options d'analyse via lesquelles neutraliser les menaces éventuellement déjà présente sur la machine. Dans la pratique, cette solution se traduit par trois niveaux d'analyse, à savoir complète, rapide et personnalisée.

Avec une analyse complète, Windows Defender passe en revue l'ensemble des vecteurs susceptibles d'abriter des logiciels malveillants, et vérifie ainsi scrupuleusement la mémoire, le registre et le système de fichiers. Pour démarrer l'analyse complète, cliquez sur le bouton Options d'analyse puis sélectionnez Analyse complète.

Avec une analyse rapide, Windows Defender examine uniquement les portions du système reconnus pour être utilisés par des logiciels malveillants. Pour démarrer l'analyse rapide, cliquez sur bouton Analyser de la barre d'outils.

Avec une analyse personnalisée, Windows Defender vérifie la mémoire et le registre mais ne contrôle que les points d'intérêt spécifiques que vous avez repéré sur le système de fichiers. Pour démarrer l'analyse personnalisée, cliquez sur le bouton Options d'analyse puis sélectionnez Analyse personnalisée. Ensuite, cliquez sur Sélectionner puis spécifiez les lecteurs à analyser. Enfin, cliquez sur analyser maintenant.

Définitions de virus et autres programmes malveillants

Windows Defender (comme du reste toute stratégie de protection antivirale par signature) identifie les menaces susceptibles de peser sur l'ordinateur à l'aide d'une base de données de définitions qui détaille les caractéristiques de tous les virus, logiciels espions et autres programmes malveillants connus. Chaque définition est unique pour tout logiciel. Les informations situées au niveau de la définition comprennent les noms et chemins d'accès des fichiers dans lesquels le logiciel s'immisce et les changements apportés aux emplacements critiques du système d'exploitation, notamment le Registre Windows.

S'agissant d'un catalogue en perpétuelle évolution, Windows Defender ne peut mener correctement les opérations qui lui sont assignées qu'à la condition de rester constamment alimenté des dernières définitions en date. Par défaut, Windows Defender collabore avec Windows Update pour installer les nouvelles définitions à mesure qu'elles sont publiées. Si les définitions ne sont pas à jour, un bouton Mettre à jour est alors proposé sous l'onglet Accueil ainsi que sous l'onglet Mise à jour. Cliquez sur l'un ou l'autre pour télécharger les fichiers de définition.

Service WinDefend

Le service WinDefend permet de faire fonctionner Windows Defender et assure le lien entre ledit logiciel et le gestionnaire de contrôle des services. Pour vérifier l'état et obtenir des informations en ce qui concerne le service WinDefend, utiliser la commande sc query windefend, comme dans ce qui suit, ou le composant enfichable Services.

```
SERVICE_NAME: windefend
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

Les données de configuration du service WinDefend sont stockées dans le Registre sous HKEY_LOCAL_MACHINE \SYSTEM\CurrentControlSet\Services\WinDefend.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\WinDefend
    DisplayName    REG_SZ    @%ProgramFiles%\Windows Defender\MpAsDesc.dll,-310
    ErrorControl    REG_DWORD    0x1
    ImagePath    REG_EXPAND_SZ    "%ProgramFiles%\Windows Defender\MsMpEng.exe"
    Start    REG_DWORD    0x2
    Type    REG_DWORD    0x10
    Description    REG_SZ    @%ProgramFiles%\Windows Defender\MpAsDesc.dll,-240
    DependOnService    REG_MULTI_SZ    RpcSs
    ObjectName    REG_SZ    LocalSystem
    ServiceSidType    REG_DWORD    0x1
    RequiredPrivileges    REG_MULTI_SZ    SeLoadDriverPrivilege\0SeImpersonatePrivilege\0SeBackupPrivilege
\0SeRestorePrivilege\0SeDebugPrivilege\0SeChangeNotifyPrivilege\0SeSecurityPrivilege\0SeShutdownPrivilege
\0SeIncreaseQuotaPrivilege\0SeAssignPrimaryTokenPrivilege\0SeTcbPrivilege\0SeSystemEnvironmentPrivilege
    FailureActions    REG_BINARY
 80510100000000000100000003000000140000000300000064000000000000006400000000000000640000000
    LaunchProtected    REG_DWORD    0x3
    FailureCommand    REG_SZ    C:\WINDOWS\system32\mrt.exe /EHB /ServiceFailure "CAMP=4.9.10586.0;approximate->
 Engine=1.1.12805.0;AVSIG=1.223.976.0;ASSIG=1.223.976.0" /StartService /Defender /q

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Windefend\Security
```

Automatisation de Windows Defender via PowerShell

Quelques-uns des fonctions de sécurité véhiculées par Windows Defender sont automatisables par le biais de PowerShell. La liste qui suit énumère les commandes les plus courantes employées à cet objet.

- *Add-MpPreference* Affecte les paramètres régissant Windows Defender.
- *Get-MpComputerStatus* Affiche des informations concernant l'état de la protection antimalware sur l'ordinateur.
- *Get-MpPreference* Affiche les paramètres d'analyse et des mises à jour de Windows Defender.
- *Get-MpThreat* Affiche l'historique des menaces détectées.

- *Get-MpThreatCatalog* Affiche les menaces connues à partir du catalogue des signatures.
- *Remove-MpPreference* Supprime des exclusions.
- *Remove-MpThreat* Supprime les menaces détectées.
- *Set-MpPreference* Configure les paramètres d'analyse et des mises à jour de Windows Defender.
- *Start-MpScan* Démarre une analyse sur l'ordinateur.
- *Update-MpSignature* Met à jour les définitions de logiciels malveillants.

Mise en quarantaine et autorisation des programmes

Windows Defender est configuré par défaut de façon à supprimer automatiquement tout programme établie comme un danger pour la station de travail. Les programmes avec un haut taux de risque, mais qui ne représentent pas forcément une menace directe (par exemple un logiciel espion) ou pour lesquels les présomptions accumulés ne suffisent pas pour fonder une condamnation capitale, peuvent quant à eux être placées en quarantaine.

Les éléments mis en quarantaine sur ordre de Windows Defender sont transférés vers un emplacement protégé du système de fichiers (\ProgramData\Microsoft\Windows Defender\Quarantine), où ils ne peuvent s'exécuter sans une approbation spéciale. (Nous reviendrons sur ce sujet un peu plus loin.) Pour voir et gérer les éléments en quarantaine, rendez-vous dans l'onglet Historique, cochez l'option Eléments en quarantaine, puis cliquez sur le bouton Afficher les détails. Vous avez à partir de là la possibilité de décider du sort des éventuels programmes indésirables. - Suppression définitive de tous les programmes indésirables en cliquant sur Supprimer tout.

- Suppression définitive d'un programme particulier en cliquant dessus puis en cliquant sur Supprimer.
- Restauration d'un programme particulier en cliquant dessus puis en cliquant sur Restaurer.

Les éléments autorisés sont des programmes sur lesquels Windows Defender a initialement jeté l'opprobre, mais qu'un administrateur a réhabilité. Vous pouvez voir et gérer les éléments actuellement autorisés en sélectionnant l'option Eléments autorisés à partir de l'onglet Historique, puis en cliquant sur le bouton Afficher les détails. Notez, à titre indicatif général, qu'un élément ne peut être autorisé qu'à partir du moment où il a donné lieu à des soupçons de la part de Windows Defender. Il n'est donc pas possible d'autoriser par avance certains logiciels.

ELAM (Early Launch Anti-Malware)

Le lancement anticipé du logiciel anti-malware (ELAM, Early Launch Anti-Malware) est un mécanisme de sécurité intégré à Windows de sorte à permettre aux programmes conçus pour détecter des maliciels de s'exécuter en amont dans la séquence de démarrage. Un composant engagé dans une telle direction contrôle ainsi l'initialisation des autres pilotes de démarrage jusqu'à ce que le système d'exploitation soit opérationnel. Lorsque le système est démarré avec un environnement d'exécution complet (accès réseau, stockage, etc.), un anti-malware plus évolué peut éventuellement prendre le relais afin de mettre en place les contre-mesures nécessaires.

Un pilote anti-malware de type ELAM doit être signé par Microsoft et le certificat associé doit héberger des attributs d'utilisation avancée de la clé (extended key usage) dotés de la valeur 1.3.6.1.4.1.311.61.4.1 (Pilote de logiciel anti-programme malveillant à lancement anticipé). Cette exigence s'applique sur les plate-formes aussi bien 32 bits que 64 bits.

Désactiver ELAM

Vous pouvez désactiver ELAM à tout moment si vous le souhaitez. Voici la manœuvre.

1. Maintenez enfoncée la touche Maj tout en sélectionnant Redémarrer. Il est également possible d'utiliser la commande shutdown /r /o /t 0 depuis une fenêtre d'invite de commandes ou sur la ligne Exécuter.

2. Cliquez sur Dépannage puis accédez aux Options avancées. Cliquez ensuite sur Paramètres puis sur le bouton Redémarrer.

3. Depuis l'écran intitulé Paramètres de démarrage, utilisez les touches numériques ou les touches de fonctions pour sélectionner l'option correspondant à Désactiver la protection du logiciel anti-programme malveillant à lancement anticipé.

D'ordre général, la possibilité de désactiver ELAM s'inscrit dans la logique de résoudre d'éventuels conflits entre ce composant et un pilote requis pour le démarrage de l'ordinateur. Dans ce cas, vous pouvez neutraliser temporairement ELAM puis supprimer le pilote problématique. Lors du prochain démarrage de Windows, ELAM démarre automatiquement afin de continuer à protéger l'ordinateur.

Situation d'ELAM dans la séquence de démarrage

Pour rappel, lors de la phase de démarrage de la plate-forme, tous les pilotes d'amorçage sont chargés en mémoire par Winload, puis leur séquence d'initialisation est exécutée avant de transférer finalement l'exécution au noyau. Winload recherche ensuite la présence d'un anti-malware conforme au modèle ELAM. S'il y en a un, Winload initialise alors le pilote ELAM.

Par la suite, avant exécution de la séquence d'initialisation de chaque pilote, Winload demande au pilote anti-malware ELAM d'inspecter le pilote s'apprêtant à s'exécuter. Le pilote ELAM classifie chaque image de démarrage selon une nomenclature à trois niveaux : Bon, Mauvais ou Non reconnu (au sens de non répertorié). En fonction de la politique ELAM définie par l'administrateur, le gestionnaire d'E/S - par l'intermédiaire du gestionnaire d'E/S - donne le départ à la routine d'initialisation du pilote, ou ignore cette étape et passe à l'évaluation du pilote suivant.

Fonctionnement interne de ELAM

Au niveau de sa structure interne, le pilote anti-malware ELAM se caractérise essentiellement par deux attributs complémentaires. Il doit être de type SERVICE_BOOT_START (0) et publié sous le groupe Early-Lauch. Il s'exécute sur événement (callback).

Les informations de signature utilisées lors de l'évaluation des images de démarrage par ELAM sont stockées dans le Registre sous HKLM\ELAM.

SmartScreen

Parmi les multiples couches de défense intégrées aux fonctionnalités réseau de Windows, le filtre SmartScreen fournit un système d'alerte anticipée contre les programmes malveillants, les sites Web d'hameçonnage et les fraudes en ligne. Introduite initialement sous Internet Explorer 8, une telle protection équipe à l'heure actuelle un certain nombre de produits phares conçus par Microsoft, dont Edge, Hotmail, Windows (à partir de la version 8), et d'autres.

La visée essentielle des moyens mobilisés par SmartScreen est de protéger les utilisateurs de la malveillance de certains sites, qui récupèrent illicitement des informations personnelles à des fins d'usurpation d'identité, ou diffusent des logiciels malveillants via des attaques d'ingénierie sociale. SmartScreen s'emploie à ce titre à protéger la station de travail par le biais de trois façons complémentaires :

- Lors de la navigation sur Internet, Smartscreen analyse les pages et détermine si elles possèdent des caractéristiques suspectes. Si tel est le cas, Smartscreen affiche une page d'avertissement invitant l'utilisant à proposer un commentaire et à faire preuve de prudence.

- SmartScreen compare systématiquement les liens que tente de visiter l'utilisateur (ou les outils qui le représente, par exemple Outlook pour la visualisation des courriers) à une liste dynamique de sites réputés frauduleux. S'il trouve une correspondance, SmartScreen affiche un avertissement indiquant le blocage du site par mesure de sécurité.

- SmartScreen confronte les fichiers téléchargés à partir du Web à liste de sites confirmés de diffuser des logiciels malveillants, à tout le moins problématiques au niveau de la sécurité. Le gestionnaire de téléchargement identifie dans ce contexte clairement les programmes à haut risque, cela afin que l'utilisateur puisse en toute connaissance de cause supprimer, exécuter ou enregistrer le téléchargement.

Configuration de SmartScreen

Cette sous-section fournit des informations sur le contrôle des paramètres du filtre SmartScreen.

Vous pouvez régler les paramètres SmartScreen dans le Centre de maintenance. Voici la procédure :

1. Ouvrez Centre de maintenance.

2. Cliquez sur Modifier les paramètres Windows SmartScreen.

3. Choisissez la façon dont Windows SmartScreen doit traiter les applications non reconnues.

L'utilisation des fonctionnalités SmartScreen nécessite la connexion à Internet pendant tout le processus d'utilisation. Dans le cas où l'ordinateur n'en dispose pas, un message d'avertissement est affiché afin de souligner d'éventuels problèmes de connectivité du poste.

Contrôles de réputation SmartScreen

Les navigateurs Internet Explorer et Edge mettent les contrôles de réputation SmartScreen au coeur de l'expérience d'utilisation qu'ils proposent, donnant de cette manière la vision la plus complète dudit filtre. Dans cette configuration, lorsque des sites web se trouvent sur le point d'être consultés ou que des programmes téléchargés depuis Internet sont en passe d'être exécutés, SmartScreen recherche en arrière-plan des informations sur leur réputation.

La protection SmartScreen vérifie le niveau de confiance des sites visités auprès d'une liste dynamique de sites signalés comme dangereux. De ce fait, lors d'une tentative d'accès à un site (comprendre avant même la constitution des requêtes réseau idoines), l'adresse demandée est tout d'abord évaluée face à une liste d'entités accrédités - en l'occurrence une liste des domaines ayant pour particularité de générer de hauts volumes de trafic, et dont il est par conséquent difficile de douter de la légitimité. (Généré par Microsoft, ce catalogue est intégré nativement aux navigateurs et mis à jour en continu.) Si le filtre trouve une correspondance, le navigateur reprend le traitement de la demande. S'il n'en trouve aucune, le service Microsoft SmartScreen URS (URL Reputation Service) est alors sollicité dans le but d'enquêter plus avant sur la confiance à accorder au site, et informer si y a lieu des suspicions contre lui. De là, l'utilisateur peut soit se rendre sur sa page de démarrage, revenir au site précédent ou, malgré l'avertissement sous ses yeux, poursuivre la navigation sur le page incriminée. Dans l'éventualité où l'internaute penserait être confronté à un site web hébergeant du contenu hostile, il peut le signaler en cliquant sur Outils, Sécurité puis Signaler un site Web d'hameçonnage.

Les règles établies concernant les fichiers téléchargés sont similaires sur le principe à celles utilisées pour les sites Internet. A la fin du téléchargement d'un fichier exécutable, une vérification sur la réputation de l'application est faite en ligne. Pour ce faire, un ensemble d'informations est envoyé à Microsoft, comme le nom du fichier, son empreinte numérique ou le certificat du signataire du fichier si disponible. A partir de là, SmartScreen Application Reputation cherche l'application, d'une part dans une liste de programmes populaires et, d'autre part, dans une liste de programmes reconnus nuisibles. Si le programme (ou le certificat l'accompagnant) a une réputation bien établie, aucun avertissement ne s'affiche. Si le programme téléchargé n'a aucune réputation établie (il n'est en l'occurrence dans aucune des listes susmentionnées), le filtre avertit que le fichier n'est pas fréquemment

téléchargé. Si le fichier est téléchargé depuis un site déclaré comme malveillant ou bien si le programme est connu pour être malveillant, le filtre SmartScreen le bloque.

A l'issue du téléchargement d'un programme depuis un site web, le fait que ce code exécutable provienne d'Internet est mémorisé à l'aide d'un attribut spécial, encore appelé *identifiant de zone*, associé au fichier téléchargé comme un flux de données alternatif NTFS, dans un fichier nommé Zone.Identifier. Utilisez la commande dir /R ou l'utilitaire Streams de sorte à voir concrètement comment se présentent ces informations.

Authenticode

La technologie Authenticode est un dispositif cryptographique utilisés par les concepteurs et les éditeurs de logiciels afin de signer numériquement toute forme de contenu actif, y compris les images exécutables et les pilotes. La solution ainsi constituée permet de mettre l'accent, de façon générale et non limitative, sur la mise en conformité avec des exigences particulières de qualité et de sécurité, dont l'application a comme but premier le fait d'instaurer un certain niveau de confiance avec l'utilisateur final.

La signature numérique Authenticode garantit de façon sure deux choses :

- *l'authentification de l'origine*, c'est-à-dire la garantie que le composant provient bien du fournisseur qui prétend l'avoir élaboré ;
- *l'intégrité*, c'est-à-dire la garantie que le composant chargé sur la station de travail correspond irrévocablement et en tout point au composant développé par son fournisseur.

L'utilisation la plus répandue d'Authenticode est de s'assurer de la légitimité d'un composant prévue pour Windows (programmes exécutables, pilotes mode noyau, bibliothèques, etc.). Dans ce scénario, la signature numérique Authenticode vise surtout à constituer un moyen pratique de garantir que ledit logiciel est authentique et fiable, autrement dit qu'il n'a pas d'intention malveillante ou n'est pas destine a servir de couverture pour des activités nuisibles ou répréhensibles. De ce point de vue, Authenticode permet aux applications de modifier leur comportement au regard de la signature numérique identifiée (validité de la signature, confiance dans le fournisseur). Par exemple, la signature numérique fait partie des critères employés par la plupart des navigateurs Internet pour vérifier l'authenticité d'un programme et ainsi alerter l'utilisateur dans le cas d'une anomalie de signature. De même, les boîtes de dialogue utilisées par le contrôle de compte utilisateur (UAC) lors de l'exécution d'un programme sont différentes si le programme est signé ou non.

Une autre utilisation de la signature numérique concerne le chargement de pilote noyau : les versions de Windows antérieures à Windows XP affichent un avertissement lorsque l'utilisateur tente d'installer un pilote non signé. Pour les versions 64 bits de Windows et depuis Vista, le noyau exige que le pilote soit signé par une liste restreinte et identifiée d'autorités de certification pour le charger. En dehors de cela, Authenticode est aussi employé de sorte à renforcer la sécurité du canal de distribution des mises à jour de Windows. La signature permet au système d'exploitation qui reçoit une mise à jour de vérifier la légitimité de celle-ci, y compris si elle été livrée par des tiers ou par l'intermédiaire de moyens alternatifs (DVD, clés USB et autres).

Le procédé Authenticode se trouve au coeur du chemin de validation de nombreux composants de Windows, y compris les DLL et autres fichiers centraux du système d'exploitation. C'est notamment de cette manière que la protection des ressources Windows (WRP, Windows Resource Protection) peut intervenir dès lors qu'une modification suspecte est apportée à un fichier, pour auquel cas restaurer une version officiellement déclarée dans le catalogue concerné. Au fur et à mesure que des fichiers système sont ajoutés ou mis à jour, de nouveaux catalogues (fichiers portant l'extension .cat) sont publiés dans le répertoire \system32\catroot.

Couplé au filtre SmartScreen, dont nous parlerons en aval, Authenticode se présente comme un bon moyen pour un éditeur de solutions informatiques de protéger sa marque et mettre en avant sa réputation avec une signature numérique reconnue.

Authenticode a été initialement introduit pour la vérification des contrôles ActiveX sous Internet Explorer, la version 2.0 d'Authenticode arrivant sous Internet Explorer 3. Conçu pour être facilement extensible, Authenticode

est aujourd'hui disponible pour signer de nombreux types de fichiers, dont les paquets logiciels (.cab), les catalogues (.cat), les contrôles de développement (.ctl), les bibliothèques (.dll), les exécutables (.exe), les contrôles OLE (.ocx) et les pilotes de périphériques (.sys). Il est basé sur un ensemble de procédés cryptographique particulièrement riche, incluant le schéma X.509, les spécifications PKCS #7 et PKCS #10, les algorithmes de hachage SHA et MD5.

Protection par Authenticode

Afin de bénéficier de la protection offerte par Authenticode, il est impératif de se munir au préalable d'un certificat de signature compatible avec ladite technologie. Trois façons de procéder conduisent à l'obtention d'un tel document : (1) acheter un certificat à un organisme reconnu monnayant ce genre de services, comme VeriSign ou GlobalSign ; (2) se procurer un certificat auprès d'un département d'entreprise responsable de la création de certificats numérique ; (3) générer un certificat avec l'utilitaire prévu à cet effet, Makecert, sur lequel nous reviendrons plus loin. Dans la perspective Authenticode, un certificat numérique est un fichier auquel est incorporé une paire de clés publique/privée, ainsi des métadonnées permettant de remonter à l'origine d'un composant, par exemple le fournisseur d'applications à qui le certificat a été délivré ou l'autorité de certification qui a émis le certificat. A l'aide des outils Authenticode, le fournisseur calcule à partir de son propre code une empreinte MD5 ou SHA (SHA-1 à partir de Windows Vista, ou SHA-256 à partir de Windows 8), qu'il chiffre au moyen de sa clé privée (algorithme RSA) ; il incorpore ensuite la signature ainsi obtenue (autrement dit le résultat du chiffrement de l'empreinte) et le certificat (contenant la clé publique) au composant, qui protégée de la sorte devient prêt à être distribué ou commercialisé.

Quand une tierce personne entre en possession d'un contenu protégé par Authenticode, toute une série de mécanismes se déroule de façon transparente. Quel que soit le vecteur utilisé (navigateur internet, déploiement sur le poste client, etc.) le système s'en remet au même mode opératoire. Il vérifie d'abord si la signature a bien été émise par une autorité de certification de confiance ; il dispose pour cela de la signature de l'autorité de certification - qui se trouve à l'intérieur même du certificat de signature du fournisseur - ainsi que du certificat de clé publique de cette autorité de certification. Le système contrôle ensuite les dates de validité du certificat et entame le processus de vérification de la signature Authenticode. Il recalcule dans cette optique l'empreinte MD5 ou SHA du composant, puis déchiffre la signature Authenticode au moyen de la clef publique contenue à l'intérieur du certificat de signature. Si les deux valeurs sont égales, la signature du composant est valide.

Autorités de confiance

La liste des autorités de confiance d'un système Windows correspond au magasin de certificats "Autorités de certification racine de confiance", lequel est accessible depuis le composant enfichable Certificats pour l'ordinateur local (certlm.msc).

Anatomie d'une signature numérique Authenticode

Windows intègre plusieurs boites de dialogue qui font passerelle vers les attributs Authenticode d'un fichier. Pour afficher la signature numérique d'un fichier image (.exe, .dll, ou .sys), procédez conformément à ce qui suit.

1. Accédez aux propriétés du fichier dont vous souhaitez voir la signature numérique.

2. Cliquez sur l'onglet Signatures numériques. (Si la signature est endommagée, ou si elle a été externalisée par le biais d'un fichier catalogue, les propriétés du fichier ne contiennent pas l'onglet Signatures numériques).

3. Sélectionnez une entrée parmi celles présentes puis cliquez sur le bouton Détails.

4. Dans la fenêtre Détails de la signature numérique , sélectionnez l'onglet Général et vérifiez que la note en dessous de Informations sur la signature numérique affiche Cette signature numérique est valide.

5. Pour plus d'informations sur la signature numérique, cliquez sur le bouton Afficher le certificat.

6. Cliquez sur l'onglet Chemin d'accès de certification de sorte à voir les relations entre certificats. La hiérarchie des certificats matérialise la chaîne de confiance installée sur le poste.

Généralement, les programmes sont signés avec signtool.exe, qui requiert du signataire un certificat numérique. Cet outil est disponible dans le Platform SDK de Windows ou lors de l'installation de la suite de développement Visual Studio. L'utilitaire sigcheck.exe (SysInternals) permet la vérification d'un binaire passé en paramètre, que la signature soit embarquée ou présente au niveau d'un catalogue de sécurité auxiliaire - ce qui se trouve être le cas pour bon nombre des exécutables distribués par Microsoft.

Le logiciel Process Explorer est capable de vérifier la signature de chaque exécutable en cours d'exécution, ainsi que les bibliothèques chargées par chaque processus. Il faut pour cela activer l'option Verify Image Signatures et ajouter l'affichage de la colonne Verified Signer. De manière analogue, le programme Autoruns permet de vérifier la signature de tous les binaires configurés en démarrage automatique, en activant l'option Verify code signatures. Ces deux logiciels font partie de la suite SysInternals.

Limitations d'Authenticode

Les limitations du mécanisme Authenticode sont essentiellement les mêmes que celles impactant n'importe quelle mesure de protection par signature de code, dont le vol de clef privée et la corruption d'une autorité de certification. Par nature, Authenticode ne garantit absolument rien en ce qui concerne le contenu du composant signé, pour la bonne et simple raison que le fournisseur établit lui-même sa propre signature. Le cas le plus critique est celui des pilotes de périphériques, dans l'éventualité où des vulnérabilités sont exploitables. Ces pilotes vulnérables n'en restent pas moins numériquement signés, et à ce titre jugés de confiance conformément à la politique de sécurité du système d'exploitation.

Kernel Patch Protection (PatchGuard)

Kernel Patch Protection (KPP), connu aussi sous le nom *PatchGuard*, est une fonctionnalité des versions 64 bits de Microsoft Windows dont le but est de protéger quelques éléments clés du noyau contre toute forme de modification en mémoire et, par extension, de rendre Windows moins vulnérable à l'introduction d'un rootkit. Le mécanisme a été pour la première fois introduit dans les éditions x64 de Windows XP et Windows Server 2003.

Afin d'entraver une modification du noyau du système d'exploitation et de ses structures critiques, parmi lesquelles diverses tables de service (SSDT, IDT, GDT), PatchGuard vérifie à intervalle régulier la présence éventuelle d'altérations parmi ces dispositifs. S'il y en a, PatchGuard provoque un arrêt contrôlé du système au travers du Bug Check 0x109 (CRITICAL_STRUCTURE_CORRUPTION).

Étant donné le modèle de privilèges a deux niveaux utilisés par Windows, rien n'empêche théoriquement que du code mode noyau ne vienne interférer avec PatchGuard. Très conscients de ce fait, les concepteurs de ce logiciel l'ont en conséquence doté de toutes sortes de mesures visant à obscurcir les modalités d'implémentation le concernant : où le code réside, quelles opérations il effectue, quelles structures de données l'alimentent, lesquelles ils manipulent, et ainsi de suite. Les acrobaties conceptuelles instaurées à cet égard sont très nombreuses, et certaines d'entre elles particulièrement bien pensées. Rien n'étant définitif, les mécanismes sous-jacents à PatchGuard ont été étudié tant et si bien que plusieurs méthodes de contournement ont été développées.

La liste qui suit passe en revue les composants ou les structures protégés par PatchGuard et décrit brièvement quelques-uns des usages malveillants susceptibles d'en être faits.

- *Modules de base (ntoskrnl.exe, hal.dll, et d'autres)* Altérations apportées au noyau et/ou à la couche d'abstraction matérielle de sorte à en subvertir le fonctionnement. Introduction de portes dérobées parmi l'interface du pilote réseau.

- *Table globale de descripteurs (GDT)* Accès d'un code mode utilisateur à des routines prévues pour être sollicités normalement uniquement depuis de le contexte noyau.

- *Table des descripteurs d'interruptions (IDT)* Exécution d'une routine de service autre que celle légitime pour traiter une certaine interruption. Dissimulation de contenus mémoire par dévoiement des exceptions dues à des fautes de pages.
- *Table des services système (SSDT)* Mise en place de biais lors de la direction des appels système vers un traitement approprié, afin par exemple de cacher des processus ou des fichiers.
- *Registres de configuration processeur (MSR)* Détournement de certaines fonctions spéciales du processeur, y compris le mécanisme de transition noyau/utilisateur.
- *Piles noyau* Installation pour un thread d'une pile noyau contrôlée par l'attaquant.
- *PsInvertedFunctionTable* Prise de contrôle du système durant la gestion de certaines exceptions.
- *Types d'objet* Manipulation des points d'entrée des routines internes appelées par le gestionnaire d'objets à différents points de la durée de vie d'un objet.

Les modules faisant l'objet d'une considération attentive de la part de PatchGuard incluent :

- Ntoskrnl.exe
- Hal.dll
- Ci.dll
- Kdcom.dll
- Pshed.dll
- Clfs.dll
- Ndis.sys
- Tcpip.sys

Windows incorpore un certain nombre de méthodes utilisables en remplacement de celles déjouées par KPP.

- Les pilotes minifiltres de système de fichiers permettent aux logiciels d'intercepter et inspecter toutes les opérations concernant des fichiers, y compris le chargement de fichiers image et de DLL.
- Les notifications de filtrage du registre offrent la possibilité de participer aux activités ayant lieu au sein du registre Windows. Les solutions de sécurité peuvent de la sorte empêcher la modification de portions critiques du registre, déjouer des logiciels malveillants, etc.
- La plateforme de filtrage Windows (WFP, Windows Filtering Platform) procure différents dispositifs utiles pour la collecte et le suivi de données en provenance du réseau.
- Les notifications de processus et de threads autorisent la surveillance du démarrage et de l'arrêt de telles unités fonctionnelles, ainsi que du chargement/déchargement des DLL.
- Le filtrage d'objets rend possible la suppression de droits d'accès accordées aux processus et/ou aux threads. Les applications peuvent ainsi se défendre leurs processus contre certaines opérations.

Code Integrity

Code Integrity est le nom interne donné au mécanisme Windows chargé de vérifier l'intégrité et la provenance d'un code exécutable. Lors de la séquence de démarrage, l'intégrité du code vérifie que les fichiers systèmes

n'ont pas été modifiés et qu'aucun pilote non signé n'est exécuté en mode noyau. Une telle protection concerne notamment le noyau, la couche d'abstraction matérielle et les pilotes d'amorçage.

La façon dont Windows se comporte à l'issue des conclusions de l'intégrité du code dépend des paramètres de stratégie liés à ce niveau qui s'appliquent, par exemple la politique KMCS.

Tous les événements relatifs au chargement des pilotes sous la perspective de l'intégrité du code sont consignés dans un journal spécifique dans Journaux des applications et des services, Microsoft, Windows, CodeIntegrity, Opérationnel.

Les mesures de protection instaurées par l'intégrité du code ne s'appliquent pas uniquement aux fichiers impliqués par l'amorçage. D'autres fonctions plus avancées, par exemple la signature des pages mémoire, sont prises en charge. C'est par ce biais que l'intégrité du code vérifie les fichiers binaires chargés dans les processus protégés et les bibliothèques dynamiques mettant en oeuvre des fonctions cryptographiques.

BitLocker

Sur le plan fonctionnel, BitLocker procède par enregistrement d'empreinte (au sens cryptographique du terme) de diverses parties de l'ordinateur et du système d'exploitation dans la puce TPM. Dans sa configuration d'origine, BitLocker demande au TPM de mesurer l'enregistrement de démarrage principal (MBR), la partition de démarrage active, le secteur de démarrage, le gestionnaire de démarrage Windows et la clé de racine de stockage BitLocker. A chaque démarrage de l'ordinateur, le TPM calcule le hachage SHA1 du code mesuré et le compare à celui qui a été mémorisé lors du démarrage précédent. Si les deux valeurs correspondent, le processus de démarrage continue, autrement il s'arrête. A la fin d'un processus de démarrage réussi, le TPM délivre la clé de racine de stockage à BitLocker, qui déchiffrera les données au fur et à mesure de leur lecture.

BitLocker protège Windows contre les attaques en mode déconnecté. L'attaque en mode déconnecté correspond au scénario dans lequel un tiers mal intentionné ayant un accès physique à un équipement utilise un canal auxiliaire afin de s'emparer des données qui y sont stockés - ici, un système d'exploitation différent de celui pour lequel l'unité de masse a été configuré. Le TPM ne délivrant la clé de racine de stockage qu'à la demande du système ayant initialement créée ladite clé, il est dans cette configuration virtuellement impossible d'accéder au volume protégé.

Droits utilisateur et privilèges

Maintes opérations effectués pour le compte de l'utilisateur, vu qu'elles ne correspondent pas à la notion d'accès et/ou de manipulation d'objets, ne peuvent être autorisées via une stratégie basée sur des permissions individuelles. (Les permissions font ici référence aux droits d'accès à une ressource sécurisable, c'est à dire aux ACL rattachés à ces ressources.) Il s'agit par exemple d'arrêter le système, changer le fuseau horaire, auditer l'ordinateur et les périphériques, etc. De ce fait, dans Windows, la permission d'exécuter ces tâches est octroyée à travers les notions de *droits utilisateur* et de *privilèges*.

Un privilège correspond à la possibilité pour un compte donné (utilisateurs ou groupes d'utilisateurs) d'effectuer une opération spécifique concernant le système, par exemple arrêter l'ordinateur ou changer le fuseau horaire. Un droit utilisateur accorde ou refuse à un compte un certain type d'ouverture de session, par exemple une ouverture de session locale ou une ouverture de session via une connexion de bureau a distance.

Chaque droit d'utilisateur et chaque privilège a un nom de constante et un nom de stratégie de groupe associé. Les noms des constantes sont utilisées lorsque vous faites référence au droit d'utilisateur dans les journaux d'événements.

Paramètre de stratégie de groupe	Nom de constante
Ouvrir une session en tant que tâche	SeBatchLogonRight
Refuser l'ouverture de session en tant que tâche	SeDenyBatchLogonRight

Interdire l'ouverture d'une session locale	SeDenyInteractiveLogonRight
Refuser l'accès à cet ordinateur à partir du réseau	SeDenyNetworkLogonRight
Interdire l'ouverture de session par les services Bureau à distance	SeDenyRemoteInteractiveLogonRight
Refuser l'ouverture de session en tant que service	SeDenyServiceLogonRight
Accéder à cet ordinateur à partir du réseau	SeNetworkLogonRight
Ouvrir une session en tant que service	SeServiceLogonRight

Droits utilisateur

Les droits utilisateur régissent les méthodes grâce auxquelles des utilisateur se voient conférer des moyens d'accès à un système.

En réponse aux demandes d'ouverture de session, LSA compare le type d'ouverture de session avec les droits assignés au compte de l'utilisateur en cours de connexion ; il refuse l'ouverture l'ouverture de session si le compte n'a pas le droit qui autorise le type d'ouverture de session ou, autre éventualité, si le droit lui interdit expressément.

Les applications Windows peuvent ajouter et supprimer des droits dans un compte via les fonctions LsaAddAccountRights et LsaRemoveAccountRights, et elles peuvent prendre connaissance des droits qui sont assignés à un compte via LsaEnumerateAccountRights.

Tableau 12.9. Droits utilisateur

Droit utilisateur	Rôle
Interdire l'ouverture d'une session locale, Permettre l'ouverture d'une session locale	Utilisé pour les ouvertures de session interactives émanant de la machine locale
Accéder à cet ordinateur à partir du réseau, Refuser l'accès à cet ordinateur à partir du réseau	Utilisé pour les ouvertures de session émanant d'une machine distante
Ouvrir une session en tant que tâche, Refuser l'ouverture de session en tant que tâche	Utilisé pour les ouvertures de session de type batch
Refuser l'ouverture de session en tant que service, Ouvrir une session en tant que service	Utilisé par le gestionnaire de contrôle de service quand il démarre un service pour le compte d'un utilisateur donné
Autoriser l'ouverture de session par les services Bureau à distance, Interdire l'ouverture de session par les services Bureau à distance	Utilisé pour les ouvertures de session via les services Bureau à distance

Privilèges

Les privilèges déterminent les autorisations assignées à certains comptes pour effectuer différentes opérations, par exemple arrêter l'ordinateur, se rendre maitre de différents objets, interagir sur les priorités d'ordonnancement, et bien d'autres.

Contrairement aux droits utilisateurs, gérés sous la tutelle d'une entité unique (LSA, ainsi que l'avons vu précédemment), les différents privilèges sont définis par divers composants qui en orchestre l'intégration et la mise en oeuvre. Par exemple, le privilège Déboguer des programmes, à quoi correspond la possibilité pour un processus d'en contrôler un autre, est du ressort du gestionnaire de processus. Autre exemple, le privilège Créer des objets globaux, qui permet à un processus de créer des objets utilisables hors la session de l'utilisateur, est controlé par le gestionnaire d'objet.

Une différence essentielle séparant droits utilisateurs et privilèges tient lieu au fait que seuls ces derniers sont activables et désactivables. Pour qu'une vérification de privilège trouve une issue positive, il faut, d'une part, que le jeton spécifié renferme bel et bien le privilège concerné, et d'autre part, que ce privilège soit activé. Dans l'éventualité où l'une ou l'autre de ces deux conditions fait défaut, le système refuse d'effectuer l'opération demandée. L'idée motrice vis à vis de cette approche est de n'activer des privilèges que quand ils sont réellement indispensables, cela afin qu'un processus ne puisse par inadvertance donner lieu à une opération de sécurité privilégiée.

Les privilèges sont accordés, par les politiques de sécurité, à certains groupes d'utilisateurs. Les privilèges accordés ne sont pas forcément actifs. L'application doit alors appeler AdjustTokenPrivileges.

Compte tenu de leur nature et de leur portée, les privilèges s'appliquent uniquement au niveau local. Il est dès lors tout à fait envisageable (c'est même en réalité presque certain) de disposer de certains privilèges sur une machine, et d'autres privilèges sur d'autres machines, cela à partir du même compte d'utilisateur.

Les processus mode utilisateur peuvent contrôler un jeton de façon à vérifier s'il est détenteur de tel ou tel privilège au moyen des fonctions PrivilegeCheck et LsaEnumerateAccountRights. Coté noyau, les routines SeSinglePrivileCheck et SePrivilegeCheck poursuivent les mêmes finalités. Les fonctions établies pour la prise en charge des privilèges ne tiennent pas compte des droits utilisateur. Par contraste, les fonctions de gestion de droits utilisateur considèrent à la fois les droits utilisateur (ce pour quoi elles sont faites, donc) et les privilèges.

Tableau 12.10. Privilèges

Bit	Privilège	Constante	Droit utilisateur
02	SeCreateTokenPrivilege	SE_CREATE_TOKEN_-NAME	Créer un objet jeton
03	SeAssignPrimaryTokenPrivilege	SE_-ASSIGNPRIMARYTOKEN_-NAME	Remplacer un jeton de niveau processus
04	SeLockMemoryPrivilege	SE_LOCK_MEMORY_-NAME	Verrouiller des pages en mémoire
05	SeIncreaseQuotaPrivilege	SE_INCREASE_QUOTA_-NAME	Changer les quotas de mémoire d'un processus
06	SeUnsolicitedInputPrivilege	SE_UNSOLICITED_-INPUT_NAME	Pas d'information. Obsolète.
06	SeMachineAccountPrivilege	SE_MACHINE_-ACCOUNT_PRIVILEGE	Ajouter des stations de travail au domaine
07	SeTcbPrivilege	SE_TCB_NAME	Agir en tant que partie du système d'exploitation
08	SeSecurityPrivilege	SE_SECURITY_NAME	Gérer le journal d'audit et de sécurité
09	SeTakeOwnershipPrivilege	SE_TAKE_OWNERSHIP_-NAME	Prendre possession de fichiers ou d'autres objets
10	SeLoadDriverPrivilege	SE_LOAD_DRIVER_NAME	Charger et décharger des pilotes de périphériques
11	SeSystemProfilePrivilege	SE_SYSTEM_PROFILE_-NAME	Performance système du profil
12	SeSystemtimePrivilege	SE_SYSTEMTIME_NAME	Modifier l'heure système
13	SeProfileSingleProcessPrivilege	SEPROF_SINGLE_-PROCESS_NAME	Optimiser un processus unique
14	SeIncreaseBasePriorityPrivilege	SEINC_BASE_-PRIORITY_NAME	Augmenter la priorité de planification
15	SeCreatePagefilePrivilege	SE_CREATE_PAGEFILE_-NAME	Créer un fichier d'échange
16	SeCreatePermanentPrivilege	SE_CREATE_-PERMANENT_NAME	Créer des objets partagés permanents
17	SeBackupPrivilege	SE_BACKUP_NAME	Sauvegarder des fichiers et des répertoires
18	SeRestorePrivilege	SE_RESTORE_NAME	Restaurer des fichiers et des répertoires
19	SeShutdownPrivilege	SE_SHUTDOWN_NAME	Arrêter le système
20	SeDebugPrivilege	SE_DEBUG_NAME	Déboguer des programmes
21	SeAuditPrivilege	SE_AUDIT_NAME	Générer des audits de sécurité
22	SeSystemEnvironmentPrivilege	SE_SYSTEM_-ENVIRONMENT_NAME	Modifier les valeurs de l'environnement du microprogramme
23	SeChangeNotifyPrivilege	SE_CHANGE_NOTIFY_-NAME	Contourner la vérification de parcours
24	SeRemoteShutdownPrivilege	SE_REMOTE_-SHUTDOWN_NAME	Forcer l'arrêt à partir d'un système distant

Bit	Privilège	Constante	Droit utilisateur
25	SeUndockPrivilege	SE_UNDOCK_NAME	Retirer l'ordinateur de la station d'accueil
26	SeSyncAgentPrivilege	SE_SYNC_AGENT_NAME	Synchroniser les données du service d'annuaire
27	SeEnableDelegationPrivilege	SE_ENABLE_-DELEGATION_NAME	Permettre à l'ordinateur et aux comptes d'utilisateurs d'être approuvés pour la délégation
28	SeManageVolumePrivilege	SE_MANAGE_VOLUME_-NAME	Effectuer des tâches de maintenance de volume
29	SeImpersonatePrivilege	SE_IMPERSONATE_-NAME	Emprunter l'identité d'un client après l'authentification
30	SeCreateGlobalPrivilege	SE_CREATE_GLOBAL_-NAME	Créer des objets globaux
33	SeIncreaseWorkingSetPrivilege	SE_INC_WORKING_-SET_NAME	Augmenter une plage de travail de processus
34	SeTimeZonePrivilege	SE_TIME_ZONE_NAME	Changer le fuseau horaire
35	SeCreateSymbolicLinkPrivilege	SE_CREATE_-SYMBOLIC_LINK_NAME	Créer des liens symboliques

- **Restaurer des fichiers et des répertoires (SeRestorePrivilege)** Force NTFS à accorder l'accès à tous les fichiers et répertoires lors de la restauration, indépendamment de ce que mentionne sur le sujet le descripteur de sécurité présent. Un utilisateur muni de ce privilège peut remplacer des paramètres du Registre, substituer un fichier du système par un autre fichier de son cru.

- **Gérer le journal d'audit et de sécurité (SeSecurityPrivilege)** Requis pour exécuter un certain nombre de fonctions liées à la sécurité, telles que le contrôle et l'affichage des messages d'audit. Controlé par NtQueryInformationToken.

- **Modifier les valeurs de l'environnement du microprogramme (SeSystemEnvironmentPrivilege)** Exigé par NtQuerySystemEnvironmentValue et NtSetSystemEnvironmentValue pour pouvoir, respectivement, lire et modifier les variables d'environnement emmagasinée dans la mémoire CMOS (ce qui exige une coopération de HAL).

- **Retirer l'ordinateur de la station d'accueil (SeUndockPrivilege)** Controlé par le gestionnaire PnP mode utilisateur lors du retrait de l'ordinateur.

- **Augmenter une plage de travail de processus (SeIncreaseWorkingSetPrivilege)** Obligatoire pour augmenter ou diminuer la taille de la plage de travail d'un processus.

- **Effectuer des tâches de maintenance de volume (SeManageVolumePrivilege)** Exigé pour pouvoir exécuter des tâches de maintenance sur un volume ou un disque, telles que la défragmentation d'un volume existant, la création ou la suppression de volumes, et l'exécution de l'outil Nettoyage de disque. Appliqué par les pilotes de système de fichiers pendant une opération d'ouverture de volume.

- **Modifier un nom d'objet (SeRelabelPrivilege)** Vérifié par le SRM lorsqu'il s'agit d'élever le niveau d'intégrité d'un objet appartenant à un autre utilisateur, ou de rehausser le niveau d'intégrité d'un objet à un niveau supérieur à celui du jeton de l'appelant.

- **Forcer l'arrêt à partir d'un système distant (SeRemoteShutdownPrivilege)** Requis en vue d'arrêter un ordinateur à partir d'un emplacement sur le réseau. Winlogon vérifie que les appelants distants de la fonction InitiateSystemShutdown ont ce privilège.

- **Synchroniser les données du service d'annuaire (SeSyncAgentPrivilege)** Obligatoire pour pouvoir utiliser les services de synchronisation d'annuaire LDAP. Une telle opération est également qualifiée de synchronisation Active Directory.

- **Changer le fuseau horaire (SeTimeZonePrivilege)** Permet aux utilisateurs de définir le fuseau horaire utilisé par Windows pour afficher l'heure locale, laquelle correspond à l'heure système plus le décalage qu'induit

le fuseau horaire. L'heure système est absolue et ne se trouve en rien affectée par un changement de fuseau horaire.

- **Ajouter des stations de travail au domaine (SeMachineAccountPrivilege)** Permet aux utilisateurs d'ajouter des ordinateurs au domaine. Controlé par le gestionnaire des comptes de sécurité (SAM) d'un contrôleur de domaine lors de la création d'un compte de machine dans le domaine.

- **Augmenter la priorité de planification (SeIncreaseBasePriorityPrivilege)** Controlé par le gestionnaire de processus quand un processus tente d'augmenter la priorité d'exécution affectée à d'autres processus. Exigé par NtSetInformationProcess (quand le paramètre ProcessInformationClass vaut ProcessBasePriority ou ProcessPriorityClass) et NtSetInformationThread (quand le paramètre ThreadInformationClass vaut ThreadPriority)

- **Charger et décharger des pilotes de périphériques (SeLoadDriverPrivilege)** Permet à des utilisateurs d'installer et de supprimer des pilotes de périphériques Plug-and-Play. Les pilotes non Plug-and-Play ne sont pas affectés par ce privilège et ne peuvent être installés que par des administrateurs. Controlé par les fonctions de pilote NtLoadDriver et NtUnloadDriver.

- **Modifier l'heure système (SeSystemtimePrivilege)** Obligatoire pour redéfinir le temps chronométré au niveau de l'horloge interne, lequel fait surface en tant que date et heure de l'ordinateur. L'heure système est absolue et n'est pas affectée par un changement de fuseau horaire. Contrôlé par NtSetSystemTime.

- **Créer un fichier d'échange (SeCreatePagefilePrivilege)** Permet la création et la modification d'un fichier d'échange. Controlé par **NtCreatePagingFile**, fonction servant à créer un nouveau fichier de pagination.

- **Créer des liens symboliques (SeCreateSymbolicLinkPrivilege)** Permet à une application exécutée par un utilisateur de créer des liens symboliques. Il est de la sorte possible de rendre visible un fichier ou un dossier à un autre emplacement que celui dans lequel il se situe réellement. L'usage des liens symboliques est limité par défaut pour améliorer la sécurité. Controlé par CreateSymbolicLink.

- **Créer un objet jeton (SeCreateTokenPrivilege)** Permet à des processus des créer des jetons d'accès et de s'en servir de façon à accéder à des ressources locales.

- **Créer des objets partagés permanents (SeCreatePermanentPrivilege)** Permet à des processus de créer des objets dans le gestionnaire d'objets. Controlé par le gestionnaire d'objets quand un processus crée un objet permanent (objet qui n'est pas démantelé quand il n'y a plus de références le concernant). La plupart des composants disposant déjà de ce privilège, il est dès lors relativement rare de devoir l'attribuer spécifiquement. Exigé par le service NtMakePermanentObject et la routine noyau ObCreateObject.

- **Déboguer des programmes (SeDebugPrivilege)** Permet d'ouvrir n'importe quel processus, quel que soit son profil de sécurité. Compte tenu des possibilités auxquelles il donne lieu, le privilège Déboguer des programmes confère aux processus qui le détiennent des leviers d'action normalement hors de leur portée, par exemple accéder à des ressources qui sont, dans l'ordre habituel des choses, protégées, et par conséquent inaccessibles. Controlé par NtOpenProcess, NtOpenThread, NtSetSystemInformation (quand SystemInformationClass vaut SystemVerifierInformation).

- **Verrouiller des pages en mémoire (SeLockMemoryPrivilege)** Permet à des processus de conserver des données dans la mémoire physique, évitant de la sorte que le système ne les remise (pagine) dans la mémoire virtuelle sur disque. Controlé par NtLockVirtualMemory, implémentation noyau de VirtualLock.

- **Arrêter le système (SeShutdownPrivilege)** Permet aux utilisateurs de faire levier sur les procédures d'arrêt et de redémarrage de l'ordinateur, lesquelles sont éventuellement nécessaires à l'activation de certaines options de reconfiguration critique. La possibilité d'arrêter le système devrait être sévèrement restreinte sur des machines agissant en tant que serveur. Sur une station de travail, ce privilège devrait dans l'idéal être attribué dès que nécessaire. Controlé par NtShutdownSystem et NtRaiseHardError qui, dans l'éventualité où l'appelant ne dispose pas du privilège nécessaire, affichent une boite de dialogue d'erreur sur la console interactive.

- **Changer le fuseau horaire (SeTimeZonePrivilege)** Permet aux utilisateurs de définir le fuseau horaire utilisé par Windows pour afficher l'heure locale, laquelle correspond à l'heure système plus le décalage induit par le fuseau horaire. Tous les utilisateurs bénéficient par défaut de ce privilège.

- **Forcer l'arrêt à partir d'un système distant (SeRemoteShutdownPrivilege)** Autorise un utilisateur à arrêter ou redémarrer un ordinateur à distance. Ce privilège, fréquemment attribué aux administrateurs, leur permet de gérer les systèmes en réseau sans nécessiter d'accès physique à chaque machine.

- **Performance système du profil (SeSystemProfilePrivilege)** Autorise l'échantillonnage des performances du système.

- **Remplacer un jeton de niveau processus (SeAssignPrimaryTokenPrivilege)** Permet à des processus de remplacer le jeton associé par défaut à des sous-processus (processus enfants). Les utilisateurs possédant le droit Remplacer un jeton de niveau processus peuvent lancer des processus sous le nom d'un autre utilisateur s'ils connaissent ses informations d'identification. Voir fonction CreateProcessAsUser, laquelle permet de donner lieu à un nouveau processus en spécifiant pour lui un contexte de sécurité.

- **Sauvegarder des fichiers et des répertoires (SeBackupPrivilege)** Permet aux utilisateurs d'outrepasser la sécurité du système de fichier et de sauvegarder l'ensemble des fichiers et dossiers de l'ordinateur. Un utilisateur malintentionné pourrait employer ce privilège dans l'optique, par exemple, de dérober des données ou de copier des données pour les distribuer par la suite.

- **Retirer l'ordinateur de la station d'accueil (SeUndockPrivilege)** Permet de retirer un ordinateur portable d'une station d'accueil via la fonction Éjecter l'ordinateur.

- **Prendre possession de fichiers ou d'autres objets (SeTakeOwnershipPrivilege)** Permet à un utilisateur de prendre possession de tout objet de sécurité, y compris les fichiers et les dossiers, les imprimantes, les clés de registre et les processus. Par défaut, seuls les administrateurs se voient automatiquement accorder la capacité de s'approprier des fichiers et d'autres objets.

- **Générer des audits de sécurité (SeAuditPrivilege)** Permet à des processus d'inscrire des entrées dans le journal de sécurité en vue d'un audit des accès aux objets. Controlé par la fonction **ReportEvent**. Tout logiciel dont le fonctionnement requiert ce droit a en principe deux possibilités, soit l'accorder automatiquement lors du processus d'installation, soit l'accorder à un compte de service spécifiquement prévu à cette fin.

- **Emprunter l'identité d'un client après l'authentification (SeImpersonatePrivilege)** Autorise un compte à emprunter l'identité d'un autre compte. Controlé par le gestionnaire de processus quand un thread veut utiliser un jeton pour faire de l'emprunt d'identité et que le jeton représente un autre utilisateur que le jeton du processus possédant le thread.

Tableau 12.11. Fonctions Windows concernant les privilèges

Fonction	Description
AdjustTokenPrivileges	Modifie l'état (activé ou désactivé) des privilèges possédés par le jeton spécifié
LookupPrivilegeValue	Retourne le LUID associé à un privilège spécifié
LookupPrivilegeName	Retourne le nom d'un privilège à partir du LUID lui étant associé
LookupPrivilegeDisplayName	Retourne le nom affiché pour pour un privilège donné

Pour obtenir une liste complète des privilèges dont dispose ou dont est privé l'utilisateur connecté actuellement, ouvrez une fenêtre d'invite de commandes en mode administrateur et saisissez whoami /priv. Vous devriez voir quelque chose se rapprochant de ce qui suit.

Informations de privilèges ` --

Nom de privilège Description État ========== ========== ========= SeIncreaseQuotaPrivilege

Ajuster les quotas de mémoire pour un processus Désactivé SeSecurityPrivilege Gérer le journal d'audit et de sécurité Désactivé SeTakeOwnershipPrivilege Prendre possession de fichiers ou d'autres objets Désactivé SeLoadDriverPrivilege Charger et décharger les pilotes de périphériques Désactivé SeSystemProfilePrivilege Performance système du profil Désactivé SeSystemtimePrivilege Modifier l'heure système Désactivé SeProfileSingleProcessPrivilege Processus unique du profil Désactivé SeIncreaseBasePriorityPrivilege Augmenter la priorité de planification Désactivé SeCreatePagefilePrivilege Créer un fichier d'échange Désactivé SeBackupPrivilege Sauvegarder les fichiers et les répertoires Désactivé SeRestorePrivilege Restaurer les fichiers et les répertoires Désactivé SeShutdownPrivilege Arrêter le système Désactivé SeDebugPrivilege Déboguer les programmes Désactivé SeSystemEnvironmentPrivilege Modifier les valeurs de l'environnement du microprogramme Désactivé SeChangeNotifyPrivilege Contourner la vérification de parcours Activé SeRemoteShutdownPrivilege Forcer l'arrêt à partir d'un système distant Désactivé SeUndockPrivilege Retirer l'ordinateur de la station d'accueil Désactivé SeManageVolumePrivilege Effectuer les tâches de maintenance de volume Désactivé SeImpersonatePrivilege Emprunter l'identité d'un client après l'authentification Activé SeCreateGlobalPrivilege Créer des objets globaux Activé SeIncreaseWorkingSetPrivilege Augmenter une plage de travail de processus Désactivé SeTimeZonePrivilege Changer le fuseau horaire Désactivé SeCreateSymbolicLinkPrivilege Créer des liens symboliques Désactivé

Secure Boot

Introduit sous Windows 8, Secure Boot permet de vérifier la légitimité des éléments impliqués dans la chaine de démarrage du système d'exploitation, cela afin de garantir que rien dans ledit ensemble ne constitue une menace. La fonction est liée à UEFI, successeur du BIOS, qui ne dispose pas d'une telle sécurité.

Emprunt d'identité

Le mécanisme par lequel Windows permet à une application d'adopter temporairement un autre contexte de sécurité que celui d'origine est dit *emprunt d'identité*. Une telle technique donne de la sorte la possibilité à un thread d'agir au nom d'un autre, et plus important encore, de le faire en s'appuyant sur un jeton d'accès différent de celui du processus dans lequel il s'exécute.

L'emprunt d'identité permet à un thread de faire valoir l'utilisation momentanée d'éléments d'identification (et par extension, d'autorisation) autres que les siens propres ; ce qui se traduit dans une perspective client/serveur par la capacité d'une application serveur à adopter temporairement le profil de sécurité d'un client. Les services utilisent couramment l'emprunt d'identité lors de la validation de l'accès aux ressources. L'application serveur s'exécute à l'aide d'un compte de service, mais lorsque le serveur accepte une connexion cliente, il emprunte l'identité du client et peut ainsi pour le compte de ce dernier effectuer différents types d'opérations, le SRM (du système local ou de l'ordinateur distant) s'occupant des validations d'accès. En règle générale, un serveur, par le biais des droits qui lui conférés via son jeton, a accès à plus de ressources qu'un client et perd donc une partie de ses privilèges durant l'opération. Cependant, l'inverse peut être vrai ; il arrive que le serveur gagne des privilèges pendant l'emprunt d'identité.

Les niveaux d'emprunt d'identité de sécurité régissent le degré auquel un processus serveur peut agir au nom d'un processus client.

- SecurityAnonymous est le niveau le plus restrictif d'emprunt d'identité. Dans cette configuration, le serveur ne peut ni identifier le client ni emprunter son identité. Le serveur peut par contre, en tant qu'alternative, s'appuyer sur propres éléments d'identification pour effectuer les tests d'accès aux ressources auxquelles le client souhaite accéder.

- SecurityIdentification permet au serveur d'employer les éléments d'identification de l'utilisateur pour l'authentifier et autoriser l'accès aux opérations, mais ne peut emprunter son identité.

- SecurityImpersonation permet au serveur d'employer l'identité de l'utilisateur lors de l'accès aux ressources locales de l'ordinateur.

- SecurityDelegation est le niveau le plus permissif d'emprunt d'identité ; le serveur peut emprunter l'identité du client en local et sur les systèmes distants. Le serveur peut de la sorte transmettre l'identité de l'utilisateur à des services distants, qui sont alors en capacité d'authitifier l'utilisateur et effectuer des opérations en empruntant son identité.

Utilisateurs et groupes

Comptes utilisateurs et comptes de groupes

Windows fournit des comptes utilisateurs et des comptes de groupes. Les premiers, essentiellement conçus pour des identités individuelles, servent à identifier et authentifier des utilisateurs sur un ordinateur ou un réseau. Les comptes de groupes, d'un autre coté, visent à simplifier l'administration de plusieurs utilisateurs.

Comptes utilisateurs

Windows prend en charge deux grands types de comptes utilisateurs :

- Les comptes d'utilisateurs locaux, dont les informations résident dans la SAM locale de tout ordinateur.
- Les comptes d'utilisateurs de domaine, sous le contrôle des services d'annuaire (Active Directory).

Tout compte d'utilisateur se compose d'un nom (on parle auquel cas de nom d'ouverture de session) et se voit généralement associé à une méthode pensée pour en assurer la protection (notamment exclure son utilisation par des tiers non autorisés), par exemple mots de passe ou certificats.

Comptes utilisateurs locaux

Chaque ordinateur Windows gère son propre jeu de comptes, ou comptes locaux, qui permettent à des utilisateurs d'ouvrir une session sur le poste. À la création d'un nouveau compte d'utilisateur sur la machine, Windows enregistre les informations dudit compte dans la base locale de sécurité (SAM, Security Account Manager), base sur laquelle il s'appuie pour authentifier tout l'utilisateur qui tente de se connecter, soit localement, soit à partir d'un autre ordinateur.

Au contraire des utilisateurs de domaine, qui par l'intermédiaire l'authentification unique ont la possibilité de se connecter sur n'importe quel ordinateur et accéder à toutes les ressources du domaine, l'utilisateur disposant d'un compte local sur un ordinateur ne peut interagir qu'avec l'ordinateur considéré.

Les comptes locaux par défaut incluent un compte Administrateur local, un compte Invité local et le compte Système.

Comptes utilisateurs prédéfinis

Plusieurs comptes utilisateurs prédéfinis sont intégrés en standard avec Windows, dont Administrateur et Invité.

Compte Administrateur

Le compte prédéfini Administrateur dispose d'un accès complet aux fichiers, répertoires, services et autres ressources. Il appartient au groupe Administrateurs et est désactivé par défaut. Autres caractéristiques de ce compte qui le démarquent des autres : il est impossible de le supprimer, ou de lui enlever son affiliation initiale (comprendre le retirer du groupe Administrateurs).

Dans le cas d'un poste local ou d'un groupe de travail, où chaque ordinateur est géré séparément, les tâches effectuées par le compte administrateur comprennent la gestion des comptes utilisateurs et des groupes locaux,

le déploiement des stratégies de sécurité, l'installation et la configuration de matériels, la régulation de l'accès aux ressources (attribution des autorisations et des droits), etc.

Sur un poste local, le compte Administrateur n'a généralement accès qu'au système local. Bien que les fichiers et répertoires puissent être mis hors de portée du compte Administrateur, ce dernier a la possibilité d'en reprendre le contrôle à tout moment en modifiant leurs autorisations d'accès. Dans un environnement de domaine, le compte Administrateur bénéficie d'un accès complet sur l'ensemble du domaine.

Compte tenu des possibilités auxquels il peut potentiellement donner suite (et des répercussions en matière de sécurité), le compte administrateur prédéfini ne devrait être utilisé que si cela est vraiment nécessaire.

Compte Invité

Le compte Invité est conçu pour autoriser des utilisateurs ayant besoin d'accès ponctuels ou occasionnels.

Le compte Invité est, par défaut, membre des groupes Invités et Invités du domaine. Il est important de noter que le compte Invité, à l'instar des autres comptes nommés, fait également partie du groupe implicite Tout le monde, qui a normalement accès par défaut à certains fichiers et répertoires de l'ordinateur.

Comptes utilisateurs prédéfinis système

En plus des comptes prédéfinis, Windows inclut plusieurs pseudo comptes dont la fonction première est d'épauler le système d'exploitation lors de la réalisation de différentes tâches cruciales.

- *Système local (LocalSystem)* Permet d'exécuter des processus système et de piloter des opérations exigeant un haut niveau d'accès et de contrôle sur le système.
- *Service local (LocalService)* Permet d'exécuter des services ne nécessitant pas de privilèges et de droits d'ouverture de session particulièrement élevés.
- *Service réseau (NetworkService)* Permet d'exécuter des services ne nécessitant pas de privilèges et de droits d'ouverture de session particulièrement élevés, mais devant accéder aux ressources réseau.

Comptes d'utilisateurs de domaine

Les comptes d'utilisateurs de domaine permettent à des utilisateurs de s'authentifier auprès d'un domaine, où les comptes sont enregistrés dans le service d'annuaire Active Directory.

Comptes de groupes

Il peut dans le but de simplifier la gestion des comptes d'utilisateurs être utile de les regrouper en entités plus larges, manipulables comme un ensemble. C'est dans cette optique que Windows fournit des comptes de groupes, ou simplement groupes, qui servent à accorder des autorisations à des profils d'utilisateurs similaires, partant des droits d'accès qui leur reviennent. De ce fait, si un groupe peut accéder à une ressource, chaque utilisateur particulier du groupe en a également la possibilité.

L'utilisateur hérite des droits et des permissions accordés au groupe. Dans l'éventualité où les permissions accordées individuellement à un utilisateur entrent en conflit avec d'autres permissions accordées à un groupe dont l'utilisateur est membre, généralement la permission la plus restrictive a prédominance.

Par convention, les noms de groupes sont typiquement écrits sous la forme plurielle, cela afin d'éviter toute confusion avec un individu.

- *Administrateurs* Regroupe les membres possédant des privilèges d'administration, leur conférant un contrôle presque absolu de l'ordinateur.

- *Opérateurs de sauvegarde* Regroupe les membres ayant le pouvoir de sauvegarder et de restaurer tous les fichiers de l'ordinateur.
- *Invités* Les membres du groupe Invités disposent par défaut des mêmes droits que les membres du groupe Utilisateurs, à l'exception du compte Invité qui dispose d'autorisations restreintes. Par exemple, l'ouverture de session sur la machine locale est autorisée aux membres du groupe Invités, et interdite en ce qui concerne le compte prédéfini Invité.
- *Opérateurs de configuration réseau* Regroupe les membres possédant un certain nombre de privilèges en rapport avec la configuration des interfaces réseau.
- *Utilisateurs* Les membres de ce groupe disposent d'un nombre restreint de privilèges, leur valant un accès limité aux ressources.
- *Utilisateurs avec pouvoir* Les membres de ce groupe peuvent effectuer un certain nombre de tâches administratives (modifier l'heure, créer un compte, autoriser des partages, accéder à certaines portions du registre, installer des programmes, etc.), sans pour autant bénéficier d'un contrôle total sur la machine. Ce groupe existe pour des raisons de compatibilité avec les systèmes antérieurs.
- *Duplicateurs* : Regroupe les membres ayant le pouvoir de répliquer des fichiers entre serveurs dans un domaine. Ce groupe existe en premier lieu en prévision de l'adhésion à un domaine.
- *Utilisateurs du bureau à distance*

Les groupes spéciaux ne sont pas visibles pendant la gestion des groupes, mais résultent de l'état du système à moment donné. Ils sont appelés entités de sécurité intégrée.

- *Tout le monde*
- *Utilisateurs authentifiés*
- *Créateur propriétaire*
- *Réseau*
- *Interactif*

Le tableau suivant dresse la liste des droits par défaut attribués aux groupes d'utilisateurs intégrés.

Tableau 12.12. Droits par défaut des groupes d'utilisateurs intégrés

Groupe	Droits par défaut
Administrateurs	▪ Accéder à cet ordinateur à partir du réseau ▪ Ajuster les quotas de mémoire pour un processus ▪ Arrêter le système ▪ Augmenter la priorité de planification ▪ Autoriser l'ouverture de session par les services Bureau à distance ▪ Changer le fuseau horaire ▪ Charger et décharger des pilotes de périphériques ▪ Contourner la vérification de parcours ▪ Créer des liens symboliques ▪ Créer des objets globaux ▪ Créer un fichier d'échange ▪ Déboguer des programmes

Groupe	Droits par défaut
	▪ Effectuer des tâches de maintenance de volume
	▪ Emprunter l'identité d'un client après authentification
	▪ Forcer l'arrêt à partir d'un système distant
	▪ Gérer le journal d'audit et de sécurité
	▪ Modifier l'heure système
	▪ Modifier les valeurs de l'environnement du microprogramme
	▪ Ouvrir une session en tant que tâche
	▪ Performance système du profil
	▪ Permettre l'ouverture d'une session locale
	▪ Prendre possession de fichiers ou d'autres objets
	▪ Processus unique du profil
	▪ Restaurer les fichiers et les répertoires
	▪ Retirer l'ordinateur de la station d'accueil
	▪ Sauvegarder les fichiers et les répertoires
Opérateurs de sauvegarde	▪ Accéder à cet ordinateur à partir du réseau
	▪ Arrêter le système
	▪ Contourner la vérification de parcours
	▪ Ouvrir une session en tant que tâche
	▪ Permettre l'ouverture d'une session locale
	▪ Restaurer les fichiers et les répertoires
	▪ Sauvegarder les fichiers et les répertoires
Tout le monde	▪ Accéder à cet ordinateur à partir du réseau
	▪ Contourner la vérification de parcours
Utilisateurs du Bureau à distance	▪ Autoriser l'ouverture de session par les services Bureau à distance
Utilisateurs	▪ Accéder à cet ordinateur à partir du réseau
	▪ Arrêter le système
	▪ Augmenter une plage de travail de processus
	▪ Changer le fuseau horaire
	▪ Contourner la vérification de parcours
	▪ Permettre l'ouverture d'une session locale
	▪ Retirer l'ordinateur de la station d'accueil

UAC

Le Contrôle de compte utilisateur (UAC) est un mécanisme intégré à Windows visant à augmenter le niveau de sécurité dudit système en minimisant l'utilisation des permissions et privilèges administratifs, permettant de la sorte de n'octroyer aux utilisateurs et administrateurs que les droits minimaux relatifs aux taches qu'ils accomplissent.

Avec UAC activé, le processus d'ouverture de session pour un administrateur diffère du processus d'ouverture de session pour un utilisateur standard. Par défaut, lorsqu'un utilisateur ouvre une session sur un ordinateur, le

système crée un jeton d'accès pour l'utilisateur, lequel contient des informations sur le niveau d'accès accordé à l'utilisateur. Lorsqu'un administrateur ouvre une session, deux jetons d'accès lui sont assignés : un jeton d'accès administrateur et un jeton d'accès utilisateur standard, dont les privilèges d'administration sont par conséquent expurgés. Le jeton d'accès utilisateur standard est ensuite employé afin de donner lieu au processus du Bureau Windows (Explorer.exe), duquel tous les autres processus lancés par l'utilisateur héritent leur jeton d'accès. Par conséquent, toutes les applications exécutées à partir de là le sont en tant qu'utilisateur standard - y compris si vous avez ouvert une session avec un compte d'admistrateur.

Lorsque Contrôle du compte d'utilisateur cherche à attirer l'attention de l'utilisateur, il le fait par l'intermédiaire d'un voile noir qui couvre tout l'écran. Cette atténuation du Bureau est connue sous l'appellation Bureau sécurisé, qui a lieu sous la forme d'un processus distinct avec lequel ne peut interférer aucune autre application. Il faut ici savoir que si le Bureau sécurisé ne prenait pas en compte ces aspects, un programme malveillant serait alors en mesure d'insérer une boite de dialogue par dessus celle apparentée à UAC, éventuellement avec un message incitant l'utilisateur à laisser le programme continuer. Un tel programme pourrait également enregistrer les frappes du clavier, et ainsi reconstituer le mot de passe demandée. Lorsque le Bureau sécurisé est affiché, vous ne pouvez pas changer de tâche ni cliquer sur les fenêtres du Bureau (fenêtres qui ne sont du reste plus réellement tangibles à ce stade, puisque quand UAC fait appel au Bureau sécurisé, il prend une image instantanée du Bureau, l'assombrit et affiche l'image derrière la boîte de dialogue).

Quand une application requérant une élévation tente de s'ouvrir, UAC évalue l'application puis affiche une invite appropriée. En tant qu'administrateur, l'invite la plus courante est l'invite de consentement. Si vous utilisez un compte standard, lorsqu'un programme nécessite une élévation, c'est alors l'invite d'informations d'identification qui apparait afin de demander des identifiants. Si l'utilisateur peut les fournir et qu'ils correspondent à un administrateur, l'application s'exécute alors au moyen d'un jeton d'accès administrateur.

Pour aider l'utilisateur à mieux évaluer le risque potentiel d'une application, les invites d'élévation UAC s'appuient sur un petit spectre de couleurs connotées symboliquement. Un rapide coup d'oeil à la partie supérieure de la boite de dialogue d'invite permet dans ce contexte de se faire rapidement une idée du type de programme qui requiert une élévation.

Dans la perspective UAC, les opérations qui nécessitent une élévation au statut administrateur incluent toutes celles qui apportent des modifications à la configuration du système. Entrent, par exemple, dans cette catégorie des manoeuvres comme installer ou désinstaller des applications ou des pilotes de périphériques, modifier les paramètres du Pare-feu Windows, ajouter ou supprimer des comptes d'utilisateur, afficher ou modifier des dossiers et fichiers d'un autre utilisateur, etc. Les diverses actions qui requièrent une élévation de statut, et doivent en l'occurence faire l'objet d'une validation de la part de UAC, sont au sein de l'interface Windows affichées avec une icône en forme de bouclier jaune et bleu, ce qui permet en définitive aux utilisateurs de repérer à l'avance quelles modifications ils peuvent ou ne ne peuvent pas effectuer.

Choisir le taux d'intervention du Contrôle du compte d'utilisateur

La couche de contrôle du compte d'utilisateur a lors de son introduction sous Windows Vista donné du fil à retordre à bien des utilisateurs, fatigués en l'occurence de voir ladite protection se manifester constamment. Passés ces débuts difficiles, quelques clics suffisent désormais pour définir le taux d'intervention du Contrôle du compte d'utilisateur. Pour ce faire, accédez à la boite de dialogue Comptes d'utilisateurs puis cliquez sur Modifier les paramètres de contrôle du compte d'utilisateur. Vous aurez alors le choix entre quatre options, chacune correspondant en définitive à un niveau niveau d'alerte plus ou moins élevé.

- *Toujours m'avertir* L'invite du contrôle de compte d'utilisateur est affichée sur le Bureau sécurisé lorsqu'un programme tente d'installer un logiciel ou lorsque vous-même ou un programme tentez de modifier des paramètres de Windows. Si vous ne cliquez pas sur Oui, l'invite du contrôle de compte d'utilisateur refuse automatiquement la demande après 30 secondes. Il s'agit du paramètre le plus sécurisé.

- *M'avertir uniquement quand des applications tentent d'apporter des modifications à mon ordinateur* Vous êtes averti lorsqu'un programme tente d'apporter des modifications à votre ordinateur, y compris des modifications des paramètres de Windows, mais pas quand vous-même effectuez des opérations du même ordre.

- M'avertir uniquement quand des programmes tentent d'apporter des modifications à mon ordinateur (ne pas estomper mon bureau) Ce paramètre est identique à M'avertir uniquement quand des programmes tentent d'apporter des modifications à mon ordinateur, mais vous n'en êtes pas notifié sur le Bureau sécurisé.

- *Ne jamais m'avertir* Cette sélection désactive le contrôle de compte d'utilisateur.

Notez que la dernière des options proposées ci-dessus correspond en terme de sécurité à la pire des solutions.

Exécuter un programme en tant qu'administrateur

Si vous avez besoin, momentanément, pour un programme, de droits et privilèges administratifs, plusieurs méthodes vous sont offertes :

- Cliquez avec le bouton droit sur un programme (ou un raccourci menant à ce programme) puis, dans le menu qui vient de s'ouvrir, cliquez sur Exécuter en tant qu'administrateur. Répondez ensuite à l'invite UAC avec votre consentement ou vos identifiants.

- Démarrez une session d'invite de commandes en tant qu'administrateur puis saisissez un nom de programme, par exemple regedit (Éditeur du Registre) ou taskmgr (Gestionnaire des tâches). Comme vous avez déjà à ce stade franchi l'inspection UAC et du fait que tous les processus auxquels donne lieu l'Invite de commandes sont des processus enfant de cette dernière, vous ne recevez plus d'autres invites UAC. Cette façon de procéder est particulièrement commode dans des situations où il vous faut exécuter en tant qu'administrateur un ensemble de programmes.

- Saisissez le nom du programme à exécuter dans la zone de recherche de démarrage puis validez au moyen de la combinaison de touches Ctrl-Maj-Entrée.

Utilitaire Whoami

Les informations qui s'apparentent au contexte de sécurité de l'utilisateur interactif, incluant le nom de compte, les adhésions à des groupes, l'identificateur de connexion et les privilèges, peuvent être obtenues à l'aide de la commande whoami.

L'outil Whoami peut fonctionner selon différents modes, à quoi correspond les syntaxes que voici :

- whoami [/upn | /fqdn | /logonid]()

- whoami { [/user]() [/groups]() [/claims]() [/priv]() } [/fo format]() [/nh]()

- whoami /all [/fo format]() [/nh]()

Les commutateurs admissibles auprès de la commande sont les suivants.

- */upn* Affiche le nom du compte au format UPN (nom d'utilisateur principal connecté à un domaine).

- */fqdn* Affiche le nom du compte au format FQDN (nom de domaine pleinement qualifié).

- */user* Affiche les informations de l'utilisateur interactif ainsi que l'identificateur de sécurité (SID) qui lui correspond.

- */groups* Affiche les groupes dont fait partie l'utilisateur interactif, les identificateurs de sécurité (SID) et les attributs.

- */priv* Affiche les privilèges de sécurité associés à l'utilisateur.

- */logonid* Affiche l'ID de connexion de l'utilisateur.

- */all* Affiche le nom de l'utilisateur actuel, les groupes auxquels il appartient, les revendications et les privilèges dont il dispose.

- */fo format* Spécifie le format employé pour la présentation de la sortie. Les valeurs valides sont table, list et csv.

- */nh* Spécifie que les en-têtes de colonnes ne doivent pas être affichés dans la sortie. Ce commutateur n'est autorisé qu'avec les formats table et csv.

Session secondaire (RunAs)

Dans l'éventualité où les autorisations accordées au compte interactif ne seraient pas suffisantes pour exécuter pleinement certaines applications, accéder à certaines ressources ou encore réaliser des tâches administratives, Windows offre la possibilité d'ouvrir une session secondaire, qui permet à un utilisateur d'utiliser temporairement le contexte de sécurité d'un autre.

L'ouverture de session secondaire est accessible de deux manières :

- Sélectionnez l'icône de l'application à exécuter, maintenez la touche Shift enfoncée puis effectuez un clic droit. Dans le menu contextuel nouvellement ouvert, choisissez l'option Exécuter en tant qu'autre utilisateur, après quoi une boite de dialogue apparait, demandant quelles informations de sécurité doivent être prises en compte.

- L'autre méthode consiste à exécuter la commande runas dans une invite de commande.

Les fonctionnalités d'ouverture de session secondaire sont mis en oeuvre dans le service Ouverture de session secondaire (seclogon).

La commande Runas permet à un utilisateur d'exécuter des programmes avec des autorisations différentes de celles attribuées à l'ouverture de session.

Un aspect notable de la cmd runas est qu'elle permet la création de jobs, et de le faire ainsi hors de la voie programmatique. Pour en savoir plus sur le sujet, consultez la section Objets job du chapitre Processus et threads.

Configuration des comptes

Windows comporte pas moins de quatre interfaces différentes pour gérer les utilisateurs et les groupes :

- *Comptes d'utilisateurs* Le gestionnaire des comptes d'utilisateur du panneau de configuration constitue la méthode la plus simple pour effectuer des taches courantes. Ouvrez cette version en saisissant control userpasswords dans une invite de commandes.

- *Comptes d'utilisateurs de domaine* Ouvrez cette version en saisissant control userpasswords2 ou netplwiz dans une invite de commandes.

- *Utilisateurs et groupes locaux* Le composant logiciel enfichable Utilisateurs et groupes locaux donne accès à plus de fonctionnalités de gestion de comptes que les programmes susmentionnés, et est plus convivial que les utilitaires accessibles depuis une fenêtre d'invite de commandes.

- *Utilitaires de ligne de commandes* Les sous-commandes user et localgroup intégrées de l'utilitaire net fournissent l'accès le plus large à différentes tâches relatives aux comptes. Quelques unes des options offertes par ces applications font écho à des paramètres indisponibles autrement.

Toutes ces options, plus ou moins simples d'utilisation, permettent à un administrateur de créer, modifier et supprimer des comptes, ainsi que d'interagir sur certaines de leurs propriétés (type de compte par exemple).

Net User

Les options acceptés par la commande net user sont les suivantes :

- */active:{yes | no}* Active ou désactive le compte. (Du moment où un compte est désactivé, l'utilisateur se voit dans l'incapacité d'ouvrir une session ou d'accéder aux ressources de l'ordinateur.)
- */add* Crée un nouveau compte d'utilisateur.
- */comment:"texte"* Permet d'accompagner un commentaire au compte de l'utilisateur.
- */countrycode:nnn* Spécifie le code de pays pour l'implémentation des fichiers de langue et des paramètres de régionalisation pour l'utilisateur. La valeur 000 désigne le code de pays par défaut.
- */delete* Supprime un compte d'utilisateur.
- */expires:{date|never}* Définit la date d'expiration du compte. Le compte expire au début du jour indiqué ; à partir de cette date, l'utilisateur ne peut plus ouvrir une session ni accéder à des ressources sur l'ordinateur, ce jusqu'à ce qu'un administrateur vienne repousser l'échéance avec une nouvelle date d'expiration ultérieure. L'option never, qui se rapporte à un compte n'expirant jamais, est celle par défaut.
- */fullname:"nom"* Spécifie le nom compte de l'utilisateur.
- */logonpasswordchg:{yes|no}* Spécifie si l'utilisateur doit être amené à changer son mot de passe lors de la prochaine ouverture de session. Par défaut, la valeur est no.
- */passwordchg:{yes|no}* Spécifie si l'utilisateur est autorisé à changer son mot de passe. La valeur par défaut est yes.
- */scriptpath:chemin* Emplacement du script d'ouverture de session de l'utilisateur.
- */times:{période|no}* Définit les moments où un compte a la possibilité (comprendre qu'il en a le droit) d'ouvrir une session. La valeur all signifie que l'ouverture de session est permise à tout moment ; une valeur vierge empêche en permanence l'utilisateur d'ouvrir une session.

Stratégies de mots de passe

Les stratégies de mots de passe définissent les exigences relatives aux mots de passe et comprennent les paramètres suivants :

- *Durée de vie maximale du mot de passe* Détermine la durée pendant laquelle les utilisateurs conservent leur mot de passe avant de devoir le modifier. L'objectif, ici, est d'obliger les utilisateurs à renouveler périodiquement leur mot de passe. Par défaut, le délai d'expiration est défini à 42 jours, mais toute valeur comprise entre 0 et 999 est envisageable. La valeur zéro indique que les mots de passe n'expirent jamais.
- *Durée de vie minimale du mot de passe* Prive l'utilisateur de la possibilité de changer son mot de passe pendant un certain nombre de jours. Un tel paramètre permet d'éviter que les utilisateurs reviennent à leur ancien mot de passe immédiatement après qu'un administrateur leur en ait attribué un nouveau, ce qui tend à fragiliser le système.
- *Longueur minimale du mot de passe* Définit le nombre minimum de caractères qu'un mot de passe peut contenir.
- *Nombre de mots de passe antérieurs à conserver* Définit le nombre de mots de passe dont Windows garde trace pour chaque utilisateur, ce afin d'empêcher leur réutilisation. Une valeur nulle (0) pour ce paramètre désactive l'option, ce qui signifie que tous les mots de passe, y compris celui en cours, peuvent être réemployés.

Pour afficher les paramètres relatifs aux mots de passe, saisissez net accounts à une invite de commande. Vous devriez voir quelque chose ressemblant à ce qui suit.

```
C:\>net accounts
Fermeture forcée de la session après expiration ? :                   Jamais
Durée de vie minimale du mot de passe (jours) :                       0
Durée de vie maximale du mot de passe (jours) :                       42
Longueur minimale du mot de passe :                                   0
Nombre de mots de passe antérieurs à conserver :                      Aucune
Seuil de verrouillage :                                               Jamais
Durée du verrouillage (min) :                                         30
Fenêtre d'observation du verrouillage (min) :                         30
Rôle de l'ordinateur :                                                STATION
La commande s'est terminée correctement.
```

Stratégies de verrouillage de compte

Les stratégies de verrouillage de compte régissent la manière dont le système se comporte en cas de tentative de connexion infructueuse (nom d'utilisateur valide mais mot de passe incorrect). Les paramètres instaurés en la matière sont les suivants :

- *Durée du verrouillage* Détermine un délai pendant lequel un compte verrouillé le reste avant d'être déverrouillé automatiquement. La valeur 0 verouille le compte indéfiniment, de sorte que seul un administrateur puisse mettre fin au blocus. La valeur par défaut est de 30 minutes.

- *Seuil de verrouillage* Contrôle le nombre de tentative de connexion invalide possible avant le verrouillage du compte. La valeur par défaut est zéro, ce qui signifie que les échecs de tentatives de connexion ne se voient pas sanctionnés. Toute autre valeur (maximum 999) indique un seuil de verrouillage spécifique.

- *Fenêtre d'observation du verrouillage* Détermine la durée pendant laquelle conserver les informations se rapportant aux tentatives de connexion infructueuses, ce qui permet de mieux différencier les problèmes de sécurité réels de ceux imputables à une simple erreur.

Gérer les sessions des utilisateurs

Pour observer et gérer chacun des utilisateurs ayant une session ouverte sur la station de travail locale, une des méthodes envisageables à cet égard consiste à employer le Gestionnaire de taches. Pour ce faire, sollicitez l'application puis sélectionnez l'onglet Utilisateurs, lequel affiche les sessions des utilisateurs, qu'ils soient locaux ou distants.

En développant l'entrée qui correspond à un utilisateur, il est possible de voir la consommation processeur, mémoire, disque et réseau de chaque application auquel l'utilisateur a donné lieu - et, partant, de chaque processus que la session renferme.

Visualisation des autorisations affectées aux utilisateurs

Accesschk est un programme en mode console qui permet de visualiser les permissions d'accès qu'un utilisateur ou un groupe d'utilisateurs détient sur divers objets, tels que fichiers, dossiers, clés de registre, etc. La syntaxe associée est la suivante :

```
usage: accesschk [-s][-e][-u][-r][-w][-n][-v]-[f <account>,...][[-a]|[-k]|[-m]|[-p [-f] [-t]]|[-h][-o [-t <object type>]][-
c]|[-d]] [[[-l|-L] [-i]]|[username]] <file, directory, event log, registry key, process, service, object>
```

- **c** La requête concerne un service Windows. Pour cibler tous les services, utilisez le caractère générique *.

- **d** Contraint l'analyse à se focaliser exclusivement sur les répertoires.

- **e** N'affiche que les niveaux d'intégrité.
- **k** La requête concerne une clé de registre.
- **o** La requête concerne un objet dans l'espace de noms du gestionnaire d'objets.
- **r** Ne montre que les objets qui permettent un accès en lecture.
- **s** Opère en mode récursif.
- **n** Affiche uniquement les objets qui n'autorisent aucun accès.
- **w** Ne montre que les objets qui permettent un accès en écriture.

Pour tout objet répondant aux critères choisis, Accesschk synthétise les droits d'accès pour chaque utilisateur et groupe mentionné dans l'ACL de l'objet, indiquant R pour les autorisations de lecture, W pour les autorisations d'écriture, et aucun symbole si ces permissions sont absentes.

Droits d'accès et masques d'accès

Tels que définis dans le modèle de sécurité de Windows, *droits d'accès* et *masques d'accès* expriment, pour les premiers un ensemble particulier d'opérations qu'un thread est en mesure d'effectuer sur un objet, pour les seconds un ensemble de droits d'accès pris en charge par un objet (autrement dit ce qu'il est possible de faire dessus). Tous les objets sécurisables du système utilisent un format de masque d'accès. Quand un processus essaie d'ouvrir un handle vers un objet existant, il doit en la circonstance spécifier un ensemble de *droits d'accès désirés*, en clair indiquer ce qu'il veut faire de l'objet. Par exemple, une application qui doit définir et interroger les valeurs d'une clé de registre peut ouvrir la clé en utilisant un masque d'accès pour demander les droits d'accès pour lire (KEY_QUERY_VALUE) et écrire (KEY_SET_VALUE) la clé.

Windows divise la structure commune aux masques d'accès en trois parties distinctes : les droits d'accès standards, qui s'appliquent à tous les types d'objets, les droits d'accès spécifiques, dépendant du type d'objet, et le mappage générique des droits d'accès, qui met en relation les quatre droits d'accès génériques (lire, écrire, exécuter, tout) et les droits d'accès spécifiques au type.

Droits d'accès génériques

Chaque objet Windows sécurisable est titulaire de droits d'accès génériques, lesquels font la corrélation entre un jeu de permissions élémentaires (lire, écrire, exécuter, tout) et différents droits qui dépendent dépendants de la nature de l'objet (comprendre son type). Par exemple, dans le contexte d'un objet fichier, le droit générique GENERIC_READ combine les permissions READ_CONTROL et SYNCHRONIZE, et les droits d'accès spécifiques FILE_READ_DATA, FILE_READ_ATTRIBUTES et FILE_READ_EA.

Comme elles dérivent des types d'objet, la mise en conformité des permissions génériques avec celles standards et celles spécifiques est du ressort de chacun des gestionnaires de l'exécutif ; ainsi, les processus, threads et jobs sont-ils conditionnés à ce niveau par le gestionnaire de processus, les fichiers par le gestionnaire d'E/S, etc.

Tableau 12.13. Droits d'accès normaux

Permission	Rôle principal	Types d'objets concernés	Exemples concrets d'opérations	Note
DELETE	Supprimer un objet	Fichier, processus, thread, mutex, événement, clé de registre	Effacer un fichier, terminer un processus, supprimer un mutex	Peut échouer si l'objet est vérouillé ou protégé par d'autres mécanismes
READ_CONTROL	Consulter les informations	Tous types d'objets sécurisés	Vérifier qui peut accéder à un fichier ou un processus	Ne permet pas de modifier la sécurité

Permission	Rôle principal	Types d'objets concernés	Exemples concrets d'opérations	Note
	sécurité associés à un objet			
SYNCHRONIZE	Synchroniser un thread avec un objet, conformément à son état (signalé ou non)	Thread, processus, événement, mutex	Attendre la fin d'un thread, attendre qu'un mutex soit disponible	
WRITE_DAC	Modifier la liste de contrôle d'accès discrétionnaire associé à un objet	Fichier, clé de registre, processus, mutex, événement	Accorder/Révoquer l'accès à un fichier	Très sensible : peut changer la manière dont un utilisateur interagit avec un objet
WRITE_OWNER	Modifier le propriétaire d'un objet	Fichier, clé de registre, processus	Accéder à la proprité d'un fichier ou d'un processus pour ensuite modifier ses droits	Offre un contrôle complet sur l'objet après prise de possession

Tableau 12.14. Masque d'accès (ACCESS_MASK)

Bits	Signification
0-15	Droits spécifiques
16	Suppression (DELETE)
17	Contrôle en lecture (READ_CONTROL)
18	Accès en écriture à la liste de contrôle d'accès (WRITE_DAC)
19	Accès en écriture du propriétaire (WRITE_OWNER)
20	Synchronisation (SYNCHRONIZE)
24	Accès à la sécurité système
25	Maximum autorisé (MAXIMUM_ALLOWED)
28	Générique pour tout (GENERIC_ALL)
29	Exécution générique (GENERIC_EXECUTE)
30	Ecriture générique (GENERIC_WRITE)
31	Lecture générique (GENERIC_READ)

Le tableau suivant présente les constantes définies en ce qui concerne les autorisations génériques.

Tableau 12.15. Autorisations génériques

Constante	Valeur	Signification
GENERIC_ALL	0x10000000	Accès complet
GENERIC_EXECUTE	0x20000000	Accès en exécution
GENERIC_READ	0x80000000	Accès en lecture
GENERIC_WRITE	0x40000000	Accès en écriture

Pour des raisons pratiques, le fichier Winnt.h définit un petit nombre d'équivalences.

Constante	Valeur	Description
STANDARD_RIGHTS_ALL	0x001F0000	Cumule DELETE, READ_CONTROL, WRITE_DAC, WRITE_OWNER et SYNCHRONIZE.
STANDARD_RIGHTS_EXECUTE	0x00020000	Equivalent à READ_CONTROL.
STANDARD_RIGHTS_READ	0x00020000	Equivalent à READ_CONTROL.
STANDARD_RIGHTS_REQUIRED	0x000F0000	Cumule DELETE, READ_CONTROL, WRITE_DAC et WRITE_OWNER.
STANDARD_RIGHTS_WRITE	0x00020000	Equivalent à READ_CONTROL.

Système à défaillance rapide (fastfail)

Les programmes confrontés à des anomalies susceptibles de mettre en défaut la sécurité ou l'intégrité du système peuvent, de sorte à prévenir des comportements imprévisibles ou des conséquences graves, atteindre prématurément leur fin via l'intrinsèque `__fastfail`, laquelle déclenche une terminaison immédiate de l'exécution en cours (processus appelant) et signale une condition d'erreur spécifique. Une telle méthode assure en cas de situations critiques une réponse rapide, préservant ainsi la fiabilité de fonctionnement de l'ordinateur.

Tableau 12.16. Codes d'erreur fastfail

Définition	Valeur
FAST_FAIL_LEGACY_GS_VIOLATION	0x00000000
FAST_FAIL_VTGUARD_CHECK_FAILURE	0x00000001
FAST_FAIL_STACK_COOKIE_CHECK_FAILURE	0x00000002
FAST_FAIL_CORRUPT_LIST_ENTRY	0x00000003
FAST_FAIL_INCORRECT_STACK	0x00000004
FAST_FAIL_INVALID_ARG	0x00000005
FAST_FAIL_GS_COOKIE_INIT	0x00000006
FAST_FAIL_FATAL_APP_EXIT	0x00000007
FAST_FAIL_RANGE_CHECK_FAILURE	0x00000008
FAST_FAIL_UNSAFE_REGISTRY_ACCESS	0x00000009
FAST_FAIL_GUARD_ICALL_CHECK_FAILURE	0x0000000A
FAST_FAIL_GUARD_WRITE_CHECK_FAILURE	0x0000000B
FAST_FAIL_INVALID_FIBER_SWITCH	0x0000000C
FAST_FAIL_INVALID_SET_OF_CONTEXT	0x0000000D
FAST_FAIL_INVALID_REFERENCE_COUNT	0x0000000E
FAST_FAIL_INVALID_JUMP_BUFFER	0x00000012
FAST_FAIL_MRDATA_MODIFIED	0x00000013
FAST_FAIL_CERTIFICATION_FAILURE	0x00000014
FAST_FAIL_INVALID_EXCEPTION_CHAIN	0x00000015
FAST_FAIL_CRYPTO_LIBRARY	0x00000016
FAST_FAIL_INVALID_CALL_IN_DLL_CALLOUT	0x00000017
FAST_FAIL_INVALID_IMAGE_BASE	0x00000018
FAST_FAIL_DLOAD_PROTECTION_FAILURE	0x00000019
FAST_FAIL_UNSAFE_EXTENSION_CALL	0x0000001A
FAST_FAIL_DEPRECATED_SERVICE_INVOKED	0x000000AB
FAST_FAIL_INVALID_BUFFER_ACCESS	0x0000001C
FAST_FAIL_INVALID_BALANCED_TREE	0x0000001D
FAST_FAIL_INVALID_NEXT_THREAD	0x0000001E
FAST_FAIL_GUARD_ICALL_CHECK_SUPPRESSED	0x0000001F
FAST_FAIL_APCS_DISABLED	0x00000020
FAST_FAIL_INVALID_IDLE_STATE	0x00000021
FAST_FAIL_MRDATA_PROTECTION_FAILURE	0x00000022
FAST_FAIL_UNEXPECTED_HEAP_EXCEPTION	0x00000023
FAST_FAIL_INVALID_LOCK_STATE	0x00000024
FAST_FAIL_GUARD_JUMPTABLE	0x00000025
FAST_FAIL_INVALID_LONGJUMP_TARGET	0x00000026
FAST_FAIL_INVALID_DISPATCH_CONTEXT	0x00000027
FAST_FAIL_INVALID_THREAD	0x00000028
FAST_FAIL_INVALID_SYSCALL_NUMBER	0x00000029
FAST_FAIL_INVALID_FILE_OPERATION	0x0000002A

Définition	Valeur
FAST_FAIL_LPAC_ACCESS_DENIED	0x0000002B
FAST_FAIL_GUARD_SS_FAILURE	0x0000002C
FAST_FAIL_LOADER_CONTINUITY_FAILURE	0x0000002D
FAST_FAIL_GUARD_EXPORT_SUPPRESSION_FAILURE	0x0000002E
FAST_FAIL_INVALID_CONTROL_STACK	0x0000002F
FAST_FAIL_SET_CONTEXT_DENIED	0x00000030
FAST_FAIL_INVALID_IAT	0x00000031
FAST_FAIL_HEAP_METADATA_CORRUPTION	0x00000032
FAST_FAIL_PAYLOAD_RESTRICTION_VIOLATION	0x00000033
FAST_FAIL_LOW_LABEL_ACCESS_DENIED	0x00000034
FAST_FAIL_ENCLAVE_CALL_FAILURE	0x00000035
FAST_FAIL_UNHANDLED_LSS_EXCEPTON	0x00000036
FAST_FAIL_ADMINLESS_ACCESS_DENIED	0x00000037
FAST_FAIL_UNEXPECTED_CALL	0x00000038
FAST_FAIL_CONTROL_INVALID_RETURN_ADDRESS	0x00000039
FAST_FAIL_UNEXPECTED_HOST_BEHAVIOR	0x0000003A
FAST_FAIL_FLAGS_CORRUPTION	0x0000003B
FAST_FAIL_VEH_CORRUPTION	0x0000003C
FAST_FAIL_ETW_CORRUPTION	0x0000003D
FAST_FAIL_RIO_ABORT	0x0000003E
FAST_FAIL_INVALID_PFN	0x0000003F
FAST_FAIL_GUARD_ICALL_CHECK_FAILURE_XFG	0x00000040
FAST_FAIL_CAST_GUARD	0x00000041
FAST_FAIL_HOST_VISIBILITY_CHANGE	0x00000042
FAST_FAIL_KERNEL_CET_SHADOW_STACK_ASSIST	0x00000043
FAST_FAIL_PATCH_CALLBACK_FAILED	0x00000044
FAST_FAIL_NTDLL_PATCH_FAILED	0x00000045
FAST_FAIL_INVALID_FLS_DATA	0x00000046
FAST_FAIL_INVALID_FAST_FAIL_CODE	0xFFFFFFFF

Mesures d'atténuation de niveau processus

- **Forcer la relocalisation des images (ForceRelocateImages)** Assure une distribution aléatoire de l'espace d'adressage (ASLR) y compris aux applications qui n'ont pas originellement été conçues pour bénéficier d'une telle fonctionnalité - laquelle se traduit au niveau de l'image par le commutateur DYNAMICBASE dans l'éditeur de liens.

- **Disallow Win32k System Calls** Désactive toute possibilité d'interagir avec la portion mode noyau du sous-système Windows (Win32k.sys), incluant l'interface utilisateur (user32.dll) et les fonctionnalités graphiques telles que GDI (gdi32.dll) ou DirectX. Tout appel système vers l'un ou l'autre de ces composants est sanctionné par une exception, mettant pour l'occasion fin au processus fautif.

- **Blocage des polices non approuvées (DisableNonSystemFonts)** Empêche le chargement de polices non approuvées, à quoi se ramène toutes celles installées en dehors du répertoire `%windir%\fonts`.

- **Chemin des résolutions des modules (PreferSystem32Images)** Contrôle la manière dont le système recherche les bibliothèques dynamiques (DLL) à charger pour un processus, attribuant dans cette séquence la première position au répertoire `%SystemRoot%\System32`, où résident les DLL système officielles de Windows - place qui dans le scénario conventionnel échoit au repertoire d'origine de l'application.

- **Interdiction des images distantes (NoRemoteImages)** Empêche un processus de charger des modules (fichiers exécutables et bibliothèques dynamiques) depuis un dispositif distant, comme un partage UNC ou un serveur WebDAV.
- **Blocage des modules à faible niveau d'intégrité (NoLowMandatoryLabelImages)** Empêche le processus de charger des modules dont le niveau de confiance en terme de sécurité est inférieur à la moyenne - étiquette d'intégrité faible, par exemple une DLL téléchargée depuis Internet.
- **Vérification stricte des handles** Contrôle de manière approfondie l'utilisation des handles par un processus, s'assurant que chacun est correctement associé à l'objet attendu. Exemple d'usage inappropriée : la mise à l'état signalé (SetEvent) d'un mutex, auquel cas le système génère une exception, mettant ainsi fin au programme incriminé - qui pourrait sinon ignorer l'anomalie.
- **Protection du Flux de Contrôle (Control Flow Guard)** Sécurise les enchaînements appel/retour de fonctions en vérifiant les destinations des instructions de saut (`call` et `jmp`) par rapport à une liste autorisée.
- **CFG strict (StrictMode)** La charge des bibliothèques est restreinte aux seules bibliothèques construites avec la protection du flux de contrôle (CFG) activée.
- **Garde de code arbitraire (ACG, Arbitrary Code Guard)** Empêche un processus d'allouer des pages exécutables, et de modifier les autorisations de celles existantes - notamment par exemple, dans une perspective offensive, afin de les rendre accessibles en écriture.
- **Binaires signés Microsoft (MicrosoftSignedOnly)** Seules les bibliothèques (DLL) dont la signature numérique dérive d'un certificat émanant de Microsoft sont autorisées pour le chargement.
- **Binaires signés Windows Store (StoreSignedOnly)** Seules les bibliothèques (DLL) dont la signature numérique dérive d'un certificat en provenance du magasin d'application Windows Store sont autorisées pour le chargement.
- **Points d'extension (DisableExtensionPoints)** Paralyse le chargement de certains types de modules potentiellement sensibles du point de vue de la sécurité (pour la plupart des composants hérités), y compris méthodes d'entrée (IME, input method editor), crochets logiciels (SetWindowsHookEx), et autres bibliothèques chargées automatiquement (AppInitDlls).
- **Blocage des binaires non-CET (BlockNonCetBinaries)** Empêche l'exécution des modules non compatibles avec les mécanismes de renforcement du contrôle du flux d'exécution (CET, Control-Flow Enforcement Technology).
- **Agrégation des pages (DisablePageCombine)** Empêche le système d'optimiser l'utilisation de la mémoire en fusionnant les pages identiques en RAM.

Tableau 12.17. Attribut MitigationPolicy de GetProcessMitigationPolicy

Valeur	Structure associée
ProcessDEPPolicy	PROCESS_MITIGATION_DEP_POLICY
ProcessASLRPolicy	PROCESS_MITIGATION_ASLR_POLICY
ProcessDynamicCodePolicy	PROCESS_MITIGATION_DYNAMIC_CODE_POLICY
ProcessStrictHandleCheckPolicy	PROCESS_MITIGATION_STRICT_HANDLE_CHECK_POLICY
ProcessSystemCallDisablePolicy	PROCESS_MITIGATION_SYSTEM_CALL_DISABLE_POLICY
ProcessMitigationOptionsMask	ULONG64
ProcessExtensionPointDisablePolicy	PROCESS_MITIGATION_EXTENSION_POINT_DISABLE_POLICY
ProcessControlFlowGuardPolicy	PROCESS_MITIGATION_CONTROL_FLOW_GUARD_POLICY
ProcessSignaturePolicy	PROCESS_MITIGATION_BINARY_SIGNATURE_POLICY
ProcessFontDisablePolicy	PROCESS_MITIGATION_FONT_DISABLE_POLICY
ProcessImageLoadPolicy	PROCESS_MITIGATION_IMAGE_LOAD_POLICY
ProcessRedirectionTrustPolicy	PROCESS_MITIGATION_REDIRECTION_TRUST_POLICY

Valeur	Structure associée
ProcessUserShadowStackPolicy	PROCESS_MITIGATION_USER_SHADOW_STACK_POLICY

Control Flow Guard (CFG)

L'utilitaire dumpbin permet d'obtenir des informations spécifiques sur la manière dont CFG est configuré pour un binaire donné.

```
C:\>dumpbin /headers /loadconfig C:\Windows\System32\smss.exe
Microsoft (R) COFF/PE Dumper Version 14.31.31107.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file C:\Windows\System32\smss.exe

PE signature found

File Type: EXECUTABLE IMAGE

FILE HEADER VALUES
            8664 machine (x64)
               6 number of sections
        360BADB6 time date stamp
               0 file pointer to symbol table
               0 number of symbols
              F0 size of optional header
              22 characteristics
                   Executable
                   Application can handle large (>2GB) addresses

OPTIONAL HEADER VALUES
             20B magic # (PE32+)
           14.30 linker version
              ...
                1 subsystem (Native)
            4160 DLL characteristics
                   High Entropy Virtual Addresses
                   Dynamic base
                   NX compatible
                   *Control Flow Guard*
               ...

  Section contains the following load config:

            00000140 size
                   0 time date stamp
                0.00 Version
                   0 GlobalFlags Clear
                   0 GlobalFlags Set
                  ...
    000000014001A328 Guard CF address of check-function pointer
    000000014001A330 Guard CF address of dispatch-function pointer
    000000014001A368 Guard CF function table
                  3A Guard CF function count
            10C14500 Guard Flags
                       CF instrumented
                       XFG instrumented
                       FID table present
                       Export suppression info present
                       Long jump target table present
                       EH Continuation table present
                  ...
                0000 Code Integrity Flags
```

```
            0000 Code Integrity Catalog
        00000000 Code Integrity Catalog Offset
        00000000 Code Integrity Reserved
000000014001A48C Guard CF address taken IAT entry table
               F Guard CF address taken IAT entry count
0000000000000000 Guard CF long jump target table
               0 Guard CF long jump target count
0000000000000000 Dynamic value relocation table
0000000000000000 Hybrid metadata pointer
0000000000000000 Guard RF address of failure-function
0000000000000000 Guard RF address of failure-function pointer
        00000000 Dynamic value relocation table offset
            0000 Dynamic value relocation table section
            0000 Reserved2
0000000000000000 Guard RF address of stack pointer verification function pointer
        00000000 Hot patching table offset
            0000 Reserved3
0000000000000000 Enclave configuration pointer
0000000000000000 Volatile metadata pointer
000000014001A358 Guard EH continuation table
               3 Guard EH continuation count
000000014001A338 Guard XFG address of check-function pointer
000000014001A340 Guard XFG address of dispatch-function pointer
000000014001A348 Guard XFG address of dispatch-table-function pointer
000000014001A350 CastGuard OS determined failure mode
0000000000000000 Guard memcpy function pointer

Guard CF Function Table

      Address
      --------
     X 0000000140001170
     X 0000000140001230
     X 0000000140001830
     X 0000000140001840
     X 0000000140001880
     ...

Guard CF Address Taken IAT Entry Table

      Address
      --------
       0000000140019D18
       0000000140019D30
       0000000140019D50
       0000000140019D60
       0000000140019D70
       ...

Guard EH Continuation Table

      Address
      --------
       000000014000239B
       00000001400030C7
       0000000140010B80
```

...

- **IMAGE_GUARD_CF_INSTRUMENTED (0x00000100)** Le module est protégé par CFG.
- **IMAGE_GUARD_CFW_INSTRUMENTED (0x00000200)**
- **IMAGE_GUARD_CF_FUNCTION_TABLE_PRESENT (0x00000400)** Le module contient des informations sur les fonctions du programme protégées par CFG.

- **IMAGE_GUARD_SECURITY_COOKIE_UNUSED (0x00000800)** Le module en question n'intègre pas de protection contre les dépassements de tampon (commutateur `/GS`).
- **IMAGE_GUARD_PROTECT_DELAYLOAD_IAT (0x00001000)** Le module a une table d'importation (IAT) en mode de chargement différé en lecture seule.
- IMAGE_GUARD_DELAYLOAD_IAT_IN_ITS_OWN_SECTION (0x00002000) _
- IMAGE_GUARD_CF_EXPORT_SUPPRESSION_INFO_PRESENT (0x00004000)
- **IMAGE_GUARD_CF_ENABLE_EXPORT_SUPPRESSION (0x00008000)** Le module permet la suppression des exportations.
- IMAGE_GUARD_CF_LONGJUMP_TABLE_PRESENT (0x00010000)
- IMAGE_GUARD_RF_INSTRUMENTED (0x00020000)
- IMAGE_GUARD_RF_ENABLE (0x00040000)
- IMAGE_GUARD_RF_STRICT (0x00080000)
- IMAGE_GUARD_RETPOLINE_PRESENT (0x00100000)

User Interface Privilege Isolation (UIPI)

Pour renforcer la sécurité et la stabilité du système d'exploitation en limitant les interactions non autorisées entre différentes applications et processus, Windows intègre l'Isolation des Privilèges de l'Interface Utilisateur (UIPI, *User Interface Privilege Isolation*). Un tel mécanisme impose des restrictions au sein du sous-système Windows, et prive en l'occurrence les processus de niveau d'intégrité inférieur de la possibilité d'envoyer des messages aux fenêtres appartenant à des processus de niveau d'intégrité supérieur.

Autres dispositifs que UIPI vient court-circuiter : les hooks de fenêtre (API SetWindowsHookEx). Concrètement, cela signifie, par exemple, qu'un processus utilisateur standard ne peut pas enregistrer les frappes au clavier qu'un autre utilisateur effectue dans une application dotée de privilèges administratifs. De la même manière, et pour des raisons relativement identiques, les hooks de journalisation sont également restreints.

Tous les messages de fenêtre envoyés d'un processus vers un autre (sous-entendu un processus avec des privilèges supérieurs) ne sont pas systématiquement bloqués. En règle générale, les messages dont émanent des E/S lecture, comme `WM_GETTEXT`, sont autorisés, tandis que les E/S écriture, tels que `WM_SETTEXT`, sont stoppés.

Chapitre 13. Réseau

BITS (Background Intelligent Transfer Service)

Background Intelligent Transfer Service (BITS), soit service de transfert intelligent en arrière-plan, est un composant des systèmes d'exploitation Microsoft Windows dont le trait le plus caractéristique réside dans une utilisation intelligente de la bande passante.

Modèle OSI

Pour aider les différents constructeurs du secteur informatique à standardiser, et de la sorte faire mieux coopérer, leurs équipements (logiciel ou matériel), l'Organisation internationale de normalisation (ISO, International Organization for Standardization) décide en 1984 d'un standard de communication entre ordinateurs : le *modèle de référence pour l'interconnexion des systèmes ouverts* (OSI , Open System Interconnection).

L'objectif premier du modèle OSI consiste en l'établissement d'un cadre général pour la création de normes ultérieures cohérentes et fiables. Il tient en cela plus de l'approche descriptive que du strict schéma normatif. Les vocabulaires, concepts, règles, méthodologies et notions qui en découlent demeurent néanmoins essentiels, tellement importants en réalité qu'ils orientent une vaste majorité des discussions et échanges d'idées dans le domaine. De plus,

Le modèle OSI comporte 7 couches. Chaque couche du modèle communique avec une couche adjacente (celle du dessus ou celle du dessous). Chaque couche utilise ainsi les services des couches inférieures et en fournit à celle de niveau supérieure.

- *Couche application* Gère le transfert d'informations entre deux applications réseau, y compris des fonctions telles que les contrôles de sécurité et l'identification des machines.
- *Couche présentation* Se charge de la représentation des informations que les entités s'échangent (mise en forme - notamment pour ce qui concerne les problèmes de fin de ligne : CR-LF, LF seul, et ainsi de suite, compression, chiffrement, etc). Masque l'hétérogénéité des formes de codage utilisées par les différents systèmes.
- *Couche session* Assure le transport de données entre deux applications distantes. Comparativement à la couche transport, la couche session intégre des services supplémentaires comme la synchronisation des échanges, ce qui ouvre la voie à la possibilité de les interrompre et les reprendre.
- *Couche transport* Assure une transmission de bout en bout des informations entre deux équipements.
- *Couche réseau* Gère la façon dont les informations sont acheminés de l'émetteur au récepteur. En plus de contrôler la route empruntée par les paquets, cette couche assure également l'interconnection entre réseaux hétérogènes. Elle doit à cet effet être capable de modifier le format d'adresse d'une trame, de redimensionner les paquets, d'assurer l'interopérabilité de traitement entre protocoles différents, etc.
- *Couche liaison de données* Fournit les moyens fonctionnels et procéduraux pour le transfert de données entre des systèmes immédiatement adjacents, ainsi que les dispositifs nécessaires à la détection (et éventuellement la correction) des erreurs susceptibles d'être apparu au niveau de la couche physique.
- *Couche physique* Transmet des flots de bits au câble réseau ou à un autre support de transmission physique.

Commandes *net*

La suite de commandes réseau *net* offre différentes possibilités très pratiques pour l'administration des réseaux et serveurs Windows. Pour afficher la liste des commandes disponibles, tapez net /? dans l'invite de commande. Pour obtenir de l'aide en ce qui concerne une commande en particulier, entrez net help suivi du nom de la commande, par exemple net help start.

La liste des commandes et une brève description de chacune d'entre elles est donnée ci-dessous. Certaines de ces commandes sont des doublons de fonctions accessibles autrement : Shell réseau (netsh.exe), Contrôleur de services (sc.exe), Configuration IP (ipconfig.exe), pour ne citer que quelques approches.

- net accounts
- net computer
- *net config*
- net continue
- net file
- net group
- *net help* Fournit des informations concernant les commandes et les messages d'erreur.
- *net helpmsg* Explique pourquoi une erreur est survenue et fournit des informations relatives à la résolution d'un problème.
- *net localgroup* Crée des groupes locaux ou en modifie la liste des membres
- net name
- *net pause* Met en pause un service en cours d'exécution.
- *net print* Affiche des informations concernant les files d'attente d'impression et gère les tâches d'impression.
- *net send* Permet d'envoyer des messages instantanées NetBIOS à des ordinateurs.
- net session
- net share
- net start
- net statistics
- net stop
- *net time* Affiche l'heure ou synchronise l'horloge par rapport à celle d'un serveur de temps Microsoft Windows.
- *net use* Associe une lettre de lecteur locale à un partage réseau distant (dossier, imprimante), par exemple net use z: \\Serveur\Public. La commande net use sans paramètre permet de visualiser les connexions réseau en cours.
- net user
- *net view* Affiche les resources ou ordinateurs du réseau.

Netsh

Netsh (pour *network shell* en anglais) est un logiciel utilitaire qui permet d'afficher et de modifier la configuration de l'ordinateur et de gérer divers composants réseau, par exemple un serveur DHCP.

Dans la perspective Netsh, les différents environnements réseau avec lesquels ledit logiciel permet une interaction sont appelés des *contextes*, qui constituent ainsi des sous-ensembles de configuration. De même,

chaque contexte peut comporter des sous-contextes, et ainsi de suite. Un contexte est désigné par une chaîne alphabétique spécifique. Pour opérer dans un contexte donné (et sous-contextes éventuels), il suffit d'indiquer successivement les noms de contextes à la suite de la commande NETSH, ou à chaque invite de commande si on opère en mode interactif. La liste qui suit passe en revue les principaux contextes.

Tableau 13.1. Contextes de la commande Netsh

Contexte	Fonction
AAAA	Permet de gérer la base de données d'authentification, d'autorisation, de gestion des comptes et d'audit utilisée par IAS (Internet Authentication Service), qui est l'implémentation Microsoft de RADIUS (Remote Authentication Dial-In User Service)
Bridge	Permet de gérer les fonctionnalités de pont sur la machine.
DHCP	Permet de gérer des serveurs DHCP.
Firewall	Permet de gérer le Pare-feu Windows
Interface	Permet de gérer les cartes réseau de la machine
RAS	Permet de gérer l'accès à une machine à distance (RAS, Remote Access Server)
WINS	Permet de gérer le service Windows de nommage Internet (WINS, Windows Internet Name Service)

Résolution de noms

Dans un réseau IP, la résolution de nom est le procédé par lequel un nom basé sur des caractères, tel que www.microsoft.com ou MyComputer, est converti en une adresse numérique, telle que 192.168.1.1. Windows, comme beaucoup d'autre système d'exploitation, font appel à de la résolution de noms en vue de faciliter la communication avec les autres ordinateurs. De cette manière, au lieu de recourir à de longues séries de chiffres difficilement mémorisables, les utilisateurs emploient un nom familier pour accéder à une machine distante.

Windows fournit plusieurs méthodes de résolution de noms, incluant le service de nommage Internet (WINS, Windows Internet Name Service), le système de nom de domaine (DNS, Domain Name System), le protocole de résolution des noms de pairs (PNRP, Peer Name Resolution Protocol), le fichier d'hôtes (hosts) et le fichier lmhosts. Ces services et leurs modalités sont traités dans les sections qui suivent.

WINS (Windows Internet Name Service)

Le service Windows de nommage Internet (WINS, Windows Internet Name Service) gère le mappage entre les noms NetBIOS et les adresses IP employés par les applications TCP/IP basées sur NetBIOS.

Composants de mise en réseau Windows

- La plate-forme de filtrage Windows (WFP, Windows Filtering Platform) est une architecture disposant d'API utiles à tout logiciel se basant sur l'inspection des flux de communication (pare-feu, anti-virus et et autres solutions de protection). WFP permet aux applications d'interagir de façon poussée avec le traitement des paquets à différents niveaux de la pile réseau Windows. Les données réseau peuvent de la sorte être suivies, filtrées, et éventuellement modifiées, avant d'atteindre la destination qui leur a été donnée.

- Les pilotes WFP (Windows Filtering Platform) sont des pilotes en mode noyau qui étendent les fonctionnalités de Windows en matière de filtrage réseau.

- TDI (Transport Driver Interface) est une interface générale entre les applications et presque tous les protocoles de la couche transport (TCP, UDP, etc).

- Les clients TDI sont des pilotes hérités qui implémentent généralement la portion mode noyau d'une API réseau.

- Les transports TDI, également appelés transports ou pilotes NDIS sont des pilotes de protocole en mode noyau. Ils acceptent les IRP émanant de clients TDI et traitent les demandes que ces IRP représentent.

- La bibliothèque de fonctions NDIS (Ndis.sys) cache aux pilotes de carte réseau la complexité de l'équipement physique et fait office d'interface entre la couche 2 (liaison) et 3 (réseau) du modèle OSI.

- Les pilotes de miniport NDIS sont des pilotes noyau responsables de l'interfaçage des transports TDI vers les adaptateurs réseau associés. Les pilotes de miniport NDIS ne manipulent pas d'IRP, mais enregistrent auprès du système différents points d'entrée (pointeurs sur fonction) vers les routines NDIS appropriées. Ils ne communiquent pas non plus directement avec le matériel, sollicitant à cet effet les fonctions de la bibliothèque NDIS, lesquelles débouchent sur les fonctions HAL correspondantes.

Windows Filtering Platform (WFP)

La plateforme de filtrage Windows (WFP, Windows Filtering Platform) défini un ensemble de primitives et d'appels systèmes permettant la collecte, à plus forte raison l'analyse, de données en provenance du réseau.

DFS (Distributed File System)

Les noms UNC, s'ils demeurent un moyen commode et effectivement éprouvé de désigner des ressources sur un réseau, perdent de leur attrait dès lors que plusieurs serveurs de fichiers peuvent offrir le même contenu (les noms au format UNC incluent explicitement le nom du serveur, limitant par nature leur utilisation à un seul). Afin de prendre en compte cette complexité, Windows supporte un système de fichiers répartis (DFS, Distributed File System), lequel offre aux administrateurs la possibilité de créer une vue hiérarchique des partages de fichiers cohérente sur l'ensemble des sites d'un réseau, qui plus est accessible facilement.

Voici les différents éléments dont se compose DFS :

- *Espace de noms* La représentation virtuelle des dossiers partagées au sein de l'organisation. Le chemin d'accès d'un espace de noms est similaire à un chemin UNC pointant vers un dossier partagé, par exemple \\Serveur1\Public\Documents\Textes. Dans cet exemple, le dossier partagé Public et ses sous-dossiers Documents et Textes sont tous hébergés sur Serveur1.

- *Serveur d'espaces de noms* Un serveur d'espaces de noms héberge un espace de noms. Il peut s'agir d'un serveur autonome, d'un serveur membre du domaine ou d'un contrôleur de domaine.

- *Racine de l'espace de noms* La racine de l'espace de noms constitue le point de départ de l'espace de noms.

- *Dossier* Un conteneur dans un espace de noms qui fournit aux clients un contenu réel, ou éventuellement redirige les clients vers une cible de dossier.

- *Cible de dossier* Un emplacement dans lequel sont stockées des données et du contenu.

DFS se fonde sur deux technologies complémentaires : espaces de noms DFS et réplication DFS.

Les espaces de noms DFS permettent de regrouper des dossiers partagés situés à différents emplacements du réseau (comprendre sur différents serveurs) et de les présenter sous forme d'une arborescence virtuelle unifiée. Les avantages d'une manière de procéder sont nombreux, quelques-uns hautement significatifs comme l'amélioration de la disponibilité des données, un meilleur équilibrage de la charge et la simplification de la migration des données.

La réplication DFS correspond à un moteur de réplication multi-maître qui prend en charge l'ajustement des transferts. Elle s'appuie pour ce faire sur un protocole nommé Compression différentielle à distance (RDC, Remote Differential Compression), qui se prête particulièrement bien aux réseaux dont la bande passante est limitée. RDC détecte les insertions, les suppressions et les réorganisations de données dans les fichiers, ce qui permet à la réplication DFS de ne propager que les les portions modifiées d'un fichier. Les activités de réplication en lien avec DFS sont enregistrées dans le journal DFS Replication, dont l'emplacement par défaut est \SystemRoot\System32\Winevt\Logs\DfsReplication.evtx.

L'utilitaire Système de fichiers DFS (Dfsutil.exe) gère le Système de fichiers répartis et affiche les informations s'y rapportant.

Active Directory

L'annuaire actif (AD, Active Directory) est l'implémentation sous Windows des services du protocole léger d'accès d'annuaire (LDAP, Lightweight Directory Access Protocol).

Un certain nombre d'environnements de programmation permettent d'accéder de façon conviviale aux objets et services Active Directory :

- ADSI (Active Directory Service Interface) permet d'accéder à Active Directory en présentant les objets enregistrés dans l'annuaire comme des objets COM (Component Object Model). Des passerelles avec de nombreux langages de programmation sont disponibles, y compris Visual Basic, C/C++, WSH, et d'autres.
- L'API LDAP C rassemble des API de bas niveau permettant une interface avec LDAP. Microsoft prend en charge les API LDAP C sur toutes les plates-formes Windows.
- La prise en charge MAPI est assurée de sorte à garantir la compatibilité avec Microsoft Exchange et avec les applications clientes du carnet d'adresses Outlook.

TDI

Les transports TDI implémentent généralement l'ensemble des protocoles qu'il leur revient de mettre en oeuvre. Par exemple, le pilote TCP/IP (Tcpip.sys) prend en charge TCP, UDP, IP, ARP, ICMP et IGMP.

Remote Differential Compression (RDC)

La synchronisation de fichiers entre serveurs et sites, tout particulièrement lorsque ces fichiers sont volumineux et/ou que la bande passante est limitée, peut s'avérer une procédure complexe, onéreuse à la fois en temps et en ressources machine. Aussi, pour prendre en compte ces aspects, Windows implémente RDC (Remote Differential Compression), un protocole de compression dont la qualité première est d'optimiser les transferts réseau.

RDC se compose d'un jeu d'interfaces sur lequel des applications peuvent prendre appui en vue de déterminer si des fichiers ont été modifié et, le cas échéant, détecter quelles portions de quels fichiers se sont vus être l'objet de changements. RDC gère les insertions, les suppressions et les réorganisations des données dans des fichiers quel que soit leur type.

L'endroit le plus visible où apparait RDC (du moins le plus perceptible, dans la mesure où très peu de logiciels tierce partie utilisent ladite fonction) est la réplication DFS, qui est conduite par ce biais de la manière la plus la plus efficiente possible. RDC étant capable de marquer toutes sortes de modifications, DFS propage uniquement les changements opérés au sein d'un fichier, et non le fichier entier.

RDC n'est pas utilisé sur les fichiers de taille inférieure à 64 Ko.

Winsock

Winsock offre notamment les fonctionnalités suivantes :

- Prise en charge des E/S disperser/rassembler (scatter/gather) et asynchrones
- Inclusion de mécanismes permettant aux applications de négocier une qualité de service avec le réseau (s'il en intègre les protocoles), permettant de la sorte aux applications de satisfaire des exigences de latence et de bande passante.

- Indépendance vis à vis du protocole de transport. Les applications basées sur Winsock peuvent ainsi supporter des protocoles autres que ceux que Windows reconnait nativement.
- Support de la communication multipoint, laquelle forme de diffusion permet à une source unique de transmettre simultanément vers plusieurs récepteurs.
- Conformité à l'architecture WOSA.

Vue de manière abstraite, l'architecture Winsock se décompose en 3 couches logiques distinctes :

- L'interface de programmation Winsock sert de point d'entrée aux concepteurs d'applications.
- L'interface de prestation de service normalise l'accès à différents protocoles (sans toutefois en implémenter aucun).
- Les fournisseurs de service communiquent avec les pilotes de protocoles mode noyau. A l'instar des API concernant tubes nommées et boites aux lettres, Winsock s'intègre au modèle d'E/S de Windows et enracine la prise en charge des sockets au niveau du système de fichiers (comprendre, en définitive, que ce sont des fichiers qui sont employés pour représenter les sockets). En interne, cette façon de procéder nécessite le concours du Pilote de fonction auxiliaire pour WinSock (AFD, Ancillary Function Driver), hébergé sous \Windows\\System32\Drivers\Afd.sys, et sollicité dans ce contexte par Msafd.dll.

Chapitre 14. Système d'E/S

Ce chapitre décrit la structure et les composants internes du système d'E/S de Windows, qui est le composant de l'exécutif en charge des entrées et des sorties émanant des pilotes de périphérique, des systèmes de fichier et des dispositifs apparentés à la mise en cache.

Composants du système d'E/S

La communication et la concertation entre les différents services rattachés à l'ordinateur est une problématique cruciale pour tout système d'exploitation. Les périphériques d'entrée/sortie varient tellement dans leur fonctionnement et leur dynamique que des méthodes très différentes sont requises pour les contrôler. Ces méthodes constituent le *sous-système d'entrée/sortie* du noyau, lequel a pour objectif d'isoler les composants de la couche moyenne et supérieure de la complexité de la gestion des dispositifs d'entrée/sortie. Dans la perspective Windows, le système d'E/S se compose de plusieurs entités qui gèrent ensemble les périphériques, tant matériels (physiques) que logiciels (virtuels ou logiques), et fournissent aux applications une couche d'abstraction (des interfaces) vers ces périphériques.

Voici dans les grandes lignes quelles fonctionnalités prend en charge le système d'E/S :

- Généralisation de l'interface entre le système et les périphériques.
- Prise en charge de la gestion de l'alimentation, afin le système et les divers matériels et périphériques qui constituent l'ordinateur puissent adapter leur consommation électrique en fonction de leur charge de travail.
- Uniformisation de la dénomination des périphériques.
- Prise en charge de multiples système de fichiers installables, incluant FAT, VFAT, CDFS, UDF et NTFS.
- Chargement et déchargement dynamiques des pilotes de périphérique.
- Prise en charge de Plug and Play qui permet qui permet au système de localiser et installer automatiquement les pilotes des matériels nouvellement détectés.
- Prise en charge de WMI (Windows Management Instrumentation), en vue de permettre la gestion, le suivi et le diagnostic des pilotes via des applications et scripts WMI.

Le système d'E/S de Windows, étant donné la diversité des fonctions qu'il est amené à exercer, s'articule autour de plusieurs composants de l'exécutif et d'un certain nombre de services tiers.

- Le gestionnaire d'E/S est le centre opérationnel du système d'E/S. Il définit l'infrastructure de gestion et de mise en service des pilotes de périphérique, et relie à ce ce titre les applications et les composants système aux périphériques.
- Le gestionnaire PnP collabore de façon étroite avec le gestionnaire d'E/S pour gérer l'ajout ou la suppression de périphériques matériels.
- Le gestionnaire d'alimentation, lui aussi, collabore étroitement avec le gestionnaire d'E/S pour guider le système, ainsi que les divers pilotes de périphérique qui prennent en charge les standards en matière d'options d'alimentation, dans des transitions d'état d'alimentation.
- Le registre sert de base de données stockant la description des périphériques matériels fondamentaux attachés au système, ainsi que les paramètres d'initialisation et de configuration des pilotes.
- La couche d'abstraction matérielle (HAL, Hardware Abstraction Layer) isole les pilotes des spécificités du processeur (notamment vis à vis des interruptions) en offrant des fonctions qui masquent les différences structurelles et fonctionnelles entre les plateformes reconnues par Windows.

- Les fichiers INF (extension .inf) sont des fichiers d'installation de pilote.

Gestionnaire d'entrée/sortie

À la fois pierre angulaire et fondation du système d'E/S de Windows, le gestionnaire d'E/S a pour rôle premier la définition du cadre selon lequel les requêtes d'E/S sont, au premier lieu conçus, au second distribués aux pilotes de périphérique.

Le gestionnaire d'E/S convertit les requêtes qu'il reçoit en un format standard appelé paquet de requête d'E/S (IRP, I/O Request Packet). Un IRP est une structure de données dont la sémantique regroupe l'ensemble des données logiquement apparentées à une requête d'E/S, et cela dans de multiples domaines : représentation, gestion, suivi, et enfin, achèvement (étape où l'opération a fini d'être traitée).

Outre la création et la suppression des IRP, le gestionnaire d'E/S fournit du code commun aux différent pilotes que ces derniers sollicitent pour mener à bien leur traitement des E/S. Le regroupement des taches communes dans le gestionnaire d'E/S permet de simplifier, et surtout alléger, les divers pilotes. Ainsi, le gestionnaire d'E/S offre une fonction par laquelle donner lieu à une requête d'E/S, une autre qui permet à un pilote de passer le relais à d'autres pilotes, et ainsi de suite.

Une dimension essentielle des services d'E/S rendus visibles par le gestionnaire d'E/S est d'être flexibles, permettant de la sorte aux sous-systèmes d'environnement, tels Windows et POSIX, d'implémenter par dessus ces interfaces leurs primitives spécifiques, et partant leurs propres fonctionnalités d'E/S.

En plus des fonctions usuelles d'ouverture, de fermeture, de lecture et d'écriture, le gestionnaire d'E/S intègre plusieurs fonctionnalités avancées, telles les E/S asynchrones, les E/S directes, les E/S bufferisées, les E/S rapides et les E/S de type disperser/rassembler (scatter/gather).

Matériel d'entrée/sortie

Les ordinateurs pilotent de nombreuses sortes de périphériques, lesquels, pour la plupart, relèvent de l'une ou l'autre des catégories générales que voici : dispositifs de stockage (disques, bandes), de transmission (cartes réseau), et d'interface utilisateur (écran, clavier, souris). D'autres dispositifs sont davantage spécialisés, par exemple joystick ou manette de jeu. Indépendamment de cette disparité, quelques concepts seulement sont centraux afin de comprendre la façon dont Windows relie dispositifs d'E/S et logiciels, y compris port, bus et contrôleur.

Un périphérique communique avec un ordinateur en envoyant des informations sur une liaison physique (un câble) ou à travers les airs, par l'intermédiaire d'un point de connexion externe (ou port). Lorsque plusieurs périphériques se partagent un ensemble de lignes, la connexion est appelée un bus, par exemple le bus système (ou bus interne), qui relie le microprocesseur à la mémoire vive, ou le bus d'extension (ou bus d'entrées/sorties), qui relie le microprocesseur aux connecteurs d'entrées/sorties et aux connecteurs d'extension.

Un contrôleur est un composant électronique en lien avec un port, un bus ou un périphérique. Les contrôleurs les plus simples (contrôleur de port série, par exemple) sont composés d'une puce unique (voire d'une partie d'une puce) intégrée sur la carte mère. Les contrôleurs plus complexes (contrôleur de bus SCSI) existent généralement en tant qu'extension, sous forme de carte enfichée dans l'ordinateur, et contiennent éventuellement un processeur, un microcode et un certain volume de mémoire propre. Le contrôleur, en vue d'assurer les interactions auxquelles il est susceptible de prendre part, rend visible un certain nombre de registres pour les données et les signaux de contrôle. Le processeur communique avec le contrôleur en lisant et écrivant des combinaisons de bits dans ces registres, à l'aide essentiellement d'instructions spéciales qui commanditent des transfert à l'adresse d'un port d'entrée/sortie. L'instruction d'entrée/sortie (par nature générique) demande aux lignes du bus de sélectionner le périphérique concerné, puis de transférer des bits vers où depuis l'un des registres. Le contrôleur de périphérique peut également reconnaître les entrées/sorties projetées en mémoire (*memory mapped*), auquel cas les registres apparentés au périphérique sont représentés dans l'espace d'adressage du processeur. Certains systèmes exploitent les deux techniques. Par exemple, un

contrôleur graphique standard possède des ports d'entrée/sortie pour le contrôle de base, mais expose aussi une vaste zone de représentation en mémoire pour le contenu à l'écran.

Un port d'entrée/sortie comporte en général quatre registres : un d'état, un de contrôle, et deux autres, l'un concernant les entrées de données, l'autre les sorties.

- Le *registre d'état* contient des bits servant à représenter des états particuliers, indiquant par exemple si la dernière commande a pris fin, si de l'information est prête à être lu dans le registre d'entrée de données ou si une erreur de périphérique s'est produite au cours du traitement.
- Le *registre de contrôle* a trait aux caractères directionnel, temporel et événementiel des échanges. Un certain bit du registre de contrôle d'un port série permet par exemple de choisir une communication full-duplex ou half-duplex, un autre active la vérification de l'intégrité des données (contrôle de parité), d'autres sélectionnent l'une des vitesses gérés par le port.
- Le processeur lit le registre d'*entrée de données* en vue de récupérer les informations émanant d'un périphérique (disque, réseau, clavier, etc.).
- Le processeur écrit dans le registre de *sortie de données* pour émettre des données vers un périphérique.

Un processeur, pour dialoguer avec les périphériques avec lesquels il est susceptible de le faire, a à sa disposition trois méthodes :

- En questionnant de façon continue un périphérique jusqu'à ce qu'il soit prêt à accepter de nouvelles opérations (polling).
- En donnant à un périphérique l'occasion de l'avertir quand il est prêt (interruption).
- En établissant une communication directe entre deux périphériques (DMA).

IRP (I/O Request Packet)

Une large majorité des requêtes qui circulent entre les divers composants du système d'E/S sont représentées au moyen de structures de données spéciales, dites paquet de requête d'E/S (IRP, I/O Request Packet), qui contrôlent le traitement d'une opération d'E/S à chaque étape de sa prise en charge. (Ainsi que vous le verrez par la suite, toutes les requêtes d'E/S ne donnent pas automatiquement lieu à une représentation IRP. Le mécanisme Fast I/O, par exemple, n'utilise pas les IRP.)

Chaque structure IRP est sur le plan fonctionnel un modèle autonome, cela au sens qu'elle contient toutes les données dont le système a besoin pour traiter une requête d'E/S. Certaines parties de la structure renferment des informations communes à tous les pilotes dans une pile de pilotes participant à une opération d'E/S. D'autres parties contiennent des informations spécifiques à un pilote particulier au sein de la pile.

Quand un thread appelle un service d'E/S (autrement dit un service qui donne lieu directement ou indirectement à une E/S), le gestionnaire d'E/S construit un IRP pour représenter l'opération en cours de déroulement dans le système d'E/S. Il passe ensuite au pilote concerné un pointeur vers l'IRP et se défait du paquet quand l'opération d'E/S est terminée. De son coté, le pilote reçoit un IRP, effectue l'opération spécifiée par cet intermédiaire, et repasse l'IRP au gestionnaire d'E/S, soit pour finalisation soit afin de le relayer à un autre pilote pour complément de traitement.

Les variables noyau en lien avec les IRP incluent IopLargeIrpLookasideList, IopSmallIrpLookasideList, et IopMediumIrpLookasideList.

- IRP_MJ_CLOSE Correspond à la fermeture d'un handle pour un objet fichier. Les demandes émises dans ce scénario résultent du service NtClose, appelé depuis la fonction CloseHandle.
- IRP_MJ_CREATE Correspond à la création ou à l'ouverture d'un handle pour un objet fichier. Les demandes émises dans ce scénario résultent du service NtCreateFile, appelé depuis la fonction CreateFile.

- IRP_MJ_CREATE_NAMED_PIPE Correspond à la création d'un tube nommé. Les demandes émises dans ce scénario résultent du service NtCreateNamedPipeFile.
- IRP_MJ_CREATE_MAILSLOT Les demandes émises dans ce scénario résultent du service NtCreateMailslotFile.
- IRP_MJ_DEVICE_CONTROL Correspond à une opération de contrôle d'E/S. Les demandes émises dans ce scénario résultent du service NtDeviceIoControlFile, appelé depuis la fonction DeviceIoControl.
- IRM_MJ_DIRECTORY_CONTROL Voir fonctions FindFirstFile et FindNextFile.
- IRP_MJ_FLUSH_BUFFERS Correspond à une vidange de la mémoire utilisée comme système de cache. Les demandes émises dans ce scénario résultent des fonctions FlushConsoleInputBuffer, FlushFileBuffers, PurgeComm et d'autres.
- IRP_MJ_LOCK_CONTROL Les demandes émises dans ce scénario résultent de la fonction NtLockFile.
- IRP_MJ_DIRECTORY_CONTROL Les demandes émises dans ce scénario résultent de la fonction NtNotifyChangeDirectoryFile.
- IRP_MJ_INTERNAL_DEVICE_CONTROL Les demandes émises dans ce scénario résultent de la fonction IoBuildDeviceIoControlRequest.
- IRP_MJ_QUERY_INFORMATION Correspond à une demande d'informations concernant un fichier ou un handle pour un fichier. Les demandes émises dans ce scénario résultent du service NtQueryInformationFile, appelé depuis les fonctions GetFileInformationByHandle, GetFileSize, GetCompressedFileSize, GetFileTime et d'autres.
- IRP_MJ_QUERY_EA Correspond à une demande d'informations concernant les attributs étendus associés à un fichier. Les demandes émises dans ce scénario résultent du service NtQueryEaFile.
- IRP_MJ_SET_EA Correspond à la modification des attributs étendus se rapportant à un fichier. Les demandes émises dans ce scénario résultent du service NtSetEaFile.
- IRP_MJ_CLEANUP Les demandes émises dans ce scénario résultent de la routine IopCloseFile.
- IRP_MJ_SET_VOLUME_INFORMATION Les demandes émises dans ce scénario résultent du service NtSetVolumeInformationFile.
- IRP_MJ_READ Correspond à une opération d'E/S lecture concernant un fichier ou un périphérique. Les demandes émises dans ce scénario résultent du service NtReadFile, appelé depuis les fonctions ReadFile et ReadFileEx.
- IRP_MJ_SET_INFORMATION Correspond à la modification des informations concernant un fichier ou un handle pour un fichier. Les demandes émises dans ce scénario résultent du service NtSetInformationFile, appelé depuis les fonctions SetFileAttributes, SetEndOfFile, SetFilePointer, SetFileTime et d'autres.
- IRP_MJ_SHUTDOWN Correspond à une procédure d'extinction de l'ordinateur. Les demandes émises dans ce scénario résultent du service NtShutdownSystem, appelé depuis les fonctions ExitWindows(Ex) et InitiateSystemShutdown.
- IRP_MJ_WRITE Correspond à une opération d'E/S écriture concernant un fichier ou un périphérique. Les demandes émises dans ce scénario résultent du service NtWriteFile, appelé depuis les fonctions WriteFile et WriteFileEx.
- IRP_MJ_QUERY_SECURITY Correspond à la lecture de la protection (descripteur de sécurité) d'un fichier, d'un répertoire, d'un volume ou d'un périphérique. Les demandes émises dans ce scénario résultent de la routine IopGetSetSecurityObject.

- IRP_MJ_SET_SECURITY Correspond à la modification de la protection (descripteur de sécurité) d'un fichier, d'un répertoire, d'un volume ou d'un périphérique. Les demandes émises dans ce scénario résultent de la routine IopGetSetSecurityObject.

```
lkd> dt nt!_IRP
   +0x000 Type             : Int2B
   +0x002 Size             : Uint2B
   +0x004 AllocationProcessorNumber : Uint2B
   +0x006 Reserved         : Uint2B
   +0x008 MdlAddress       : Ptr64 _MDL
   +0x010 Flags            : Uint4B
   +0x018 AssociatedIrp    : <unnamed-tag>
   +0x020 ThreadListEntry  : _LIST_ENTRY
   +0x030 IoStatus         : _IO_STATUS_BLOCK
   +0x040 RequestorMode    : Char
   +0x041 PendingReturned  : UChar
   +0x042 StackCount       : Char
   +0x043 CurrentLocation  : Char
   +0x044 Cancel           : UChar
   +0x045 CancelIrql       : UChar
   +0x046 ApcEnvironment   : Char
   +0x047 AllocationFlags  : UChar
   +0x048 UserIosb         : Ptr64 _IO_STATUS_BLOCK
   +0x050 UserEvent        : Ptr64 _KEVENT
   +0x058 Overlay          : <unnamed-tag>
   +0x068 CancelRoutine    : Ptr64     void
   +0x070 UserBuffer       : Ptr64 Void
   +0x078 Tail             : <unnamed-tag>
```

La liste qui suit énumère les attributs d'importance parmi la structure IRP.

- **Processeur d'allocation (AllocationProcessorNumber)** Numéro du processeur sur lequel l'IRP a été alloué. Ce champ permet au noyau d'optimiser le traitement des requêtes dans un environnement multiprocesseur, en favorisant une gestion localisée.

- **IRP principal (AssociatedIrp.MasterIrp)** Pointeur vers un IRP principal (maître) dans les cas où un traitement d'E/S s'est vu décomposé en plusieurs sous-opérations. Lorsqu'un pilote de haut niveau (souvent un pilote de système de fichiers) doit gérer une opération complexe, il peut décomposer cette requête en plusieurs sous-requêtes IRP (par exemple, une lecture fragmentée sur différents volumes). L'IRP principal garde en la circonstance une trace de ces IRP de second niveau.

- **Compteur d'IRP secondaires (AssociatedIrp.IrpCount)** Nombre d'IRP qui secondent un IRP principal dans le cadre d'un traitement d'E/S subdivisé en plusieurs opérations (et, partant, plusieurs sous-requêtes d'E/S).

- **Tampon système (AssociatedIrp.SystemBuffer)** Adresse virtuelle d'un tampon dans l'espace mémoire du noyau sur lequel système et pilotes peuvent s'appuyer pour lire ou écrire des données lors de certaines requêtes. Ce champ est principalement utilisé lorsque l'E/S est prise en charge dans le mode tamponné, c'est-à-dire lorsque les données utilisateur sont copiées dans un tampon intermédiaire en mémoire noyau avant ou après l'opération d'E/S proprement dite.

- **Indicateur d'annulation (Cancel)** Indique si l'IRP a été marqué comme annulé. Ce champ est actualisé lorsqu'une opération d'E/S commence de faire l'objet d'une procédure d'annulation (`IoCancelIrp`), généralement à la demande d'un thread auxiliaire ou du système (par exemple en cas de fermeture de handle). Les pilotes, en consultant une telle valeur, peuvent de la sorte éviter de traiter une requête devenue caduque.

- **Indice de l'emplacement de pile courant (CurrentLocation)** Indice désignant la position de la pile actuellement active dans un IRP, utilisé pour déterminer le pilote en cours de traitement. Chaque fois que l'IRP est transmis à un pilote plus bas dans la chaîne, cet indice est décrémenté.

- **Pointeur emplacement de pile courant (CurrentStackLocation)** Pointeur vers l'emplacement de pile actuellement actif dans un IRP, permettant au pilote courant d'accéder à sa propre section de la requête pour

en lire ou modifier les paramètres. Une telle valeur est automatiquement mise à jour à chaque transmission de l'IRP à un niveau inférieur de la chaîne.

- **IRQL lors de l'annulation (CancelIrql)** Niveau d'interruption (IRQL) en vigueur au moment où un IRP a été marqué comme annulé. Ce champ est principalement utilisé en interne par le noyau afin de garantir que la routine d'annulation s'exécute au niveau approprié. Voir routines IoCancelIrp et IoAcquireCancelSpinLock.

- **Routine d'annulation (CancelRoutine)** Pointeur vers une éventuelle fonction définie par un pilote, appelée automatiquement si l'IRP venait à être annulé tandis qu'il se trouve toujours dans une file d'attente de traitement. Cette routine permet de démanteler les ressources mobilisées pour une requête et d'en assurer la fin de manière contrôlée. Une valeur NULL indique que l'IRP n'est pas annulable. Voir fonction Windows CancelIo ; service NtCancelIoFile ; routine IoSetCancelRoutine.

- **Attributs (Flags)** Masque binaire servant à entériner les conditions et le mode de traitement d'une requête IRP, en particulier les modalités de transfert entre le mode utilisateur et l'espace noyau, ainsi que diverses options influençant le comportement de l'IRP, comme la stratégie de mise en cache, la nature des opérations attendues sur l'objet sous-jacent ou le caractère synchrone ou asynchrone de la requête. Voir tableau plus loin.

- **État d'achèvement (IoStatus)** Structure au travers de laquelle système et pilotes consignent le résultat final d'une opération d'E/S (réussite, échec ou attente), accompagné d'éventuelles informations complémentaires, par exemple le nombre d'octets transférés.

- **MDL (MdlAddress)** Pointeur vers une structure décrivant la plage de mémoire impliquée dans une opération d'E/S, typiquement en mode direct. Dans un tel cas, le système alloue une MDL afin de représenter les pages de l'espace utilisateur concernées, permettant ainsi aux pilotes d'accéder aux données sans avoir à recourir à une copie intermédiaire.

- **Statut En attente (PendingReturned)** Valeur d'état employée de sorte à indiquer qu'un pilote a marqué l'IRP comme en attente parce que du traitement supplémentaire était requis. Voir routine IoMarkIrpPending.

- **Mode d'exécution du demandeur (RequestorMode)** Indique le mode d'exécution (noyau ou utilisateur) du thread dont émane une opération d'E/S, permettant ainsi de distinguer rapidement les requêtes internes au système de celles émises par des applications. Initialisée lors de la création de l'IRP, une telle valeur est par la suite employée par les pilotes de sorte à adapter leur comportement, notamment en matière de validation des pointeurs, ceux provenant du mode utilisateur faisant l'objet d'une vigilance accrue, et devant en l'occurrence être rigoureusement contrôlés avant tout accès.

- **Taille (Size)** Taille totale en octets de la structure IRP elle-même. Ce champ est principalement utilisé à des fins de validation interne, pour s'assurer que la structure a bien été allouée ou manipulée correctement.

- **Nombre de piles (StackCount)** Nombre total d'emplacements de pile (*I/O stack locations*) alloués pour un IRP, correspondant au nombre de pilotes susceptibles d'intervenir dans une chaîne de traitement donnée. Une telle valeur est fixée lors de la création de l'IRP.

- **Type d'identification IRP (Type)** Valeur prédéfinie (habituellement `IO_TYPE_IRP`) qu'emploient les routines d'E/S pour vérifier qu'un pointeur censé désigner un IRP correspond bel et bien à la structure attendue. Cela permet d'une part de déterminer rapidement les erreurs de pointeur (par exemple un pointeur invalide ou corrompu), d'autre part d'écarter de protéger le système contre des comportements indéfinis.

- **Événement utilisateur (UserEvent)** Pointeur vers un objet événement éventuellement fourni par une application dès lors qu'elle initie une requête d'E/S en mode asynchrone. À l'achèvement de l'opération, le système signale automatiquement l'événement, permettant ainsi à l'appelant d'être notifié sans blocage.

- **Tampon utilisateur (UserBuffer)** Champ référant un tampon mémoire fourni par une application lorsqu'une requête d'E/S transite par le mode utilisateur. Sa signification dépend du mode d'accès configuré pour le périphérique concerné. En mode tamponné, cet attribut, lorsqu'il est renseigné par le système, n'a pas de

valeur autre qu'informative, les pilotes préférant généralement accéder aux données par un autre moyen (via `AssociatedIrp.SystemBuffer`). En mode direct, il peut être employé pour retourner un pointeur vers les données mappées dans l'espace utilisateur après traitement. En mode ni direct ni tamponné, le pilote accède directement à ce pointeur, par extension au contenu qu'il véhicule, mais doit le faire avec prudence, le système n'offrant en la circonstance aucune garantie quant à la validité de la région mémoire pointée.

Tableau 14.1. Bits de contrôle dans un IRP (IRP.Flags)

Valeur	Symbole	Signification
0x0001	IRP_NOCACHE	L'opération d'E/S ne doit pas utiliser de cache.
0x0002	IRP_PAGING_IO	L'E/S est effectuée dans le cadre d'un accès mémoire paginé.
0x0002	IRP_MOUNT_-COMPLETION	Indique qu'il s'agit d'un IRP de complétion de montage (utilise la même valeur que IRP_PAGING_IO).
0x0004	IRP_SYNCHRONOUS_-API	L'IRP provient d'une API synchrone côté utilisateur.
0x0008	IRP_ASSOCIATED_IRP	L'IRP est associé à un IRP maître (fragment d'une opération plus large).
0x0010	IRP_BUFFERED_IO	L'E/S est effectuée en mode tamponné.
0x0020	IRP_DEALLOCATE_-BUFFER	Le tampon noyau doit être libéré après traitement.
0x0040	IRP_INPUT_OPERATION	L'opération implique une lecture de données vers le noyau.
0x0040	IRP_SYNCHRONOUS_-PAGING_IO	Opération d'E/S paginée synchronisée (même valeur que IRP_INPUT_-OPERATION).
0x0080	IRP_CREATE_-OPERATION	L'IRP correspond à une opération de création ou d'ouverture d'objet.
0x0100	IRP_READ_OPERATION	L'IRP représente une lecture.
0x0200	IRP_WRITE_OPERATION	L'IRP représente une écriture.
0x0400	IRP_CLOSE_OPERATION	L'IRP signale une fermeture d'objet (handle).
0x0800	IRP_DEFER_IO_-COMPLETION	La complétion de l'E/S doit être différée.
0x1000	IRP_OB_QUERY_NAME	L'IRP sert à interroger le nom d'un objet.
0x2000	IRP_HOLD_DEVICE_-QUEUE	L'IRP doit retenir la file d'attente de périphérique (empêche d'autres requêtes d'être envoyées).

Tableau 14.2. Résumé des objets d'E/S (IRP Type)

ID	Type	Structure
1	IO_TYPE_ADAPTER	ADAPTER_OBJECT
2	IO_TYPE_CONTROLLER	CONTROLLER_OBJECT
3	IO_TYPE_DEVICE	DEVICE_OBJECT
4	IO_TYPE_DRIVER	DRIVER_OBJECT
5	IO_TYPE_FILE	FILE_OBJECT
6	IO_TYPE_IRP	IRP
7	IO_TYPE_MASTER_ADAPTER	-r
8	IO_TYPE_OPEN_PACKET	OPEN_PACKET
9	IO_TYPE_TIMER	IO_TIMER
10	IO_TYPE_VPB	VPB
11	IO_TYPE_ERROR_LOG	IO_ERROR_LOG_ENTRY
12	IO_TYPE_ERROR_MESSAGE	IO_ERROR_LOG_MESSAGE
13	IO_TYPE_DEVICE_OBJECT_EXTENSION	DEVOBJ_EXTENSION
18	IO_TYPE_APC	KAPC
19	IO_TYPE_DPC	KDPC
20	IO_TYPE_DEVICE_QUEUE	KDEVICE_QUEUE
21	IO_TYPE_EVENT_PAIR	KEVENT_PAIR
22	IO_TYPE_INTERRUPT	KINTERRUPT

23	IO_TYPE_PROFILE	KPROFILE

Emplacements de pile IRP

Sur le plan structurel, toute requête d’E/S (IRP) se compose de deux éléments principaux : un en-tête fixe et un ou plusieurs emplacements de pile. La partie fixe regroupe des informations générales sur la requête, telles que son type, son statut, des pointeurs vers les données auxquelles elle a trait, ou encore certains indicateurs de contrôle. Chaque emplacement de pile (*stack location*) abrite quant à lui des paramètres spécifiques au traitement de la requête pour un pilote donné.

```
lkd> dt _IO_STACK_LOCATION
nt!_IO_STACK_LOCATION
   +0x000 MajorFunction    : UChar
   +0x001 MinorFunction    : UChar
   +0x002 Flags            : UChar
   +0x003 Control          : UChar
   +0x008 Parameters       : <unnamed-tag>
   +0x028 DeviceObject     : Ptr64 _DEVICE_OBJECT
   +0x030 FileObject       : Ptr64 _FILE_OBJECT
   +0x038 CompletionRoutine : Ptr64     long
   +0x040 Context          : Ptr64 Void

lkd> dt _IO_STACK_LOCATION Parameters.
nt!_IO_STACK_LOCATION
   +0x008 Parameters  :
      +0x000 Create      : <unnamed-tag>
      +0x000 CreatePipe  : <unnamed-tag>
      +0x000 CreateMailslot : <unnamed-tag>
      +0x000 Read        : <unnamed-tag>
      +0x000 Write       : <unnamed-tag>
      +0x000 QueryDirectory : <unnamed-tag>
      +0x000 NotifyDirectory : <unnamed-tag>
      +0x000 NotifyDirectoryEx : <unnamed-tag>
      +0x000 QueryFile   : <unnamed-tag>
      +0x000 SetFile     : <unnamed-tag>
      +0x000 QueryEa     : <unnamed-tag>
      +0x000 SetEa       : <unnamed-tag>
      +0x000 QueryVolume : <unnamed-tag>
      +0x000 SetVolume   : <unnamed-tag>
      +0x000 FileSystemControl : <unnamed-tag>
      +0x000 LockControl : <unnamed-tag>
      +0x000 DeviceIoControl : <unnamed-tag>
      +0x000 QuerySecurity : <unnamed-tag>
      +0x000 SetSecurity : <unnamed-tag>
      +0x000 MountVolume : <unnamed-tag>
      +0x000 VerifyVolume : <unnamed-tag>
      +0x000 Scsi        : <unnamed-tag>
      +0x000 QueryQuota  : <unnamed-tag>
      +0x000 SetQuota    : <unnamed-tag>
      +0x000 QueryDeviceRelations : <unnamed-tag>
      +0x000 QueryInterface : <unnamed-tag>
      +0x000 DeviceCapabilities : <unnamed-tag>
      +0x000 FilterResourceRequirements : <unnamed-tag>
      +0x000 ReadWriteConfig : <unnamed-tag>
      +0x000 SetLock     : <unnamed-tag>
      +0x000 QueryId     : <unnamed-tag>
      +0x000 QueryDeviceText : <unnamed-tag>
      +0x000 UsageNotification : <unnamed-tag>
      +0x000 WaitWake    : <unnamed-tag>
      +0x000 PowerSequence : <unnamed-tag>
      +0x000 Power       : <unnamed-tag>
      +0x000 StartDevice : <unnamed-tag>
      +0x000 WMI         : <unnamed-tag>
```

```
    +0x000 Others      : <unnamed-tag>
```

- **Object périphérique (DeviceObject)** Pointeur vers l'objet périphérique auquel est associé un IRP, ce qui dans la pratique permet d'identifier le périphérique logique ou matériel que la requête vise à atteindre à ce niveau de la pile.

- **Masque de contrôle (Flags)** Jeu de modificateurs, chacun visant à influer sur le comportement général d'une requête, par exemple si elle est synchrone ou asynchrone.

- **Code de fonction majeur (MajorFunction)** Indique la nature fondamentale d'une opération d'E/S qu'un pilote est invité à traiter : ouverture/fermeture d'un fichier ou d'un périphérique, lecture de données, commandes de gestion Plug and Play, événement d'alimentation, etc. Chaque type d'opération correspond généralement à une fonction de rappel propre au pilote, initialisée dans sa table de fonctions lors de son enregistrement auprès du gestionnaire d'E/S.

- **Code de fonction mineur (MinorFunction)** Précise, le cas échant, une variation spécifique au sein d'une catégorie générale d'opérations.

- **Paramètres (Parameters)** Conteneur pour un ensemble d'attributs associés à un type particulier d'opérations. Par exemple, dans le cas d'une E/S lecture (code de fonction majeur `IRP_MJ_READ`), seule la sous-structure `Parameters.Read` contient des informations valides, telles que la position du fichier ou le nombre d'octets à traiter. Pour une E/S écriture (`IRP_MJ_WRITE`), il s'agira de `Parameters.Write`, tandis que pour une commande de gestion de périphérique (`IRP_MJ_DEVICE_CONTROL`), c'est en la circonstance `Parameters.DeviceIoControl` qui sera utilisé, et ainsi de suite selon la nature de l'opération demandée.

Visualisation des routines de ventilation d'un pilote

Pour voir la liste des fonctions qu'un pilote a définies pour ses routines de ventilation, utilisez la commande `!drvobj` du débogueur noyau.

```
lkd> !drvobj kbdclass 2
Driver object (ffff950b32af96c0) is for:
 \Driver\kbdclass

DriverEntry:   fffff80515e71010
DriverStartIo: 00000000
DriverUnload:  00000000
AddDevice:     fffff80515e6fae0

Dispatch routines:
[00] IRP_MJ_CREATE                      fffff80515e639b0 +0xfffff80515e639b0
[01] IRP_MJ_CREATE_NAMED_PIPE           fffff80581af4570 nt!IopInvalidDeviceRequest
[02] IRP_MJ_CLOSE                       fffff80515e64620 +0xfffff80515e64620
[03] IRP_MJ_READ                        fffff80515e62920 +0xfffff80515e62920
[04] IRP_MJ_WRITE                       fffff80581af4570 nt!IopInvalidDeviceRequest
[05] IRP_MJ_QUERY_INFORMATION           fffff80581af4570 nt!IopInvalidDeviceRequest
[06] IRP_MJ_SET_INFORMATION             fffff80581af4570 nt!IopInvalidDeviceRequest
[07] IRP_MJ_QUERY_EA                    fffff80581af4570 nt!IopInvalidDeviceRequest
[08] IRP_MJ_SET_EA                      fffff80581af4570 nt!IopInvalidDeviceRequest
[09] IRP_MJ_FLUSH_BUFFERS               fffff80515e65c20 +0xfffff80515e65c20
[0a] IRP_MJ_QUERY_VOLUME_INFORMATION    fffff80581af4570 nt!IopInvalidDeviceRequest
[0b] IRP_MJ_SET_VOLUME_INFORMATION      fffff80581af4570 nt!IopInvalidDeviceRequest
[0c] IRP_MJ_DIRECTORY_CONTROL           fffff80581af4570 nt!IopInvalidDeviceRequest
[0d] IRP_MJ_FILE_SYSTEM_CONTROL         fffff80581af4570 nt!IopInvalidDeviceRequest
[0e] IRP_MJ_DEVICE_CONTROL              fffff80515e6ef40 +0xfffff80515e6ef40
[0f] IRP_MJ_INTERNAL_DEVICE_CONTROL     fffff80515e6fab0 +0xfffff80515e6fab0
[10] IRP_MJ_SHUTDOWN                    fffff80581af4570 nt!IopInvalidDeviceRequest
[11] IRP_MJ_LOCK_CONTROL                fffff80581af4570 nt!IopInvalidDeviceRequest
[12] IRP_MJ_CLEANUP                     fffff80515e64260 +0xfffff80515e64260
[13] IRP_MJ_CREATE_MAILSLOT             fffff80581af4570 nt!IopInvalidDeviceRequest
```

```
[14] IRP_MJ_QUERY_SECURITY              fffff80581af4570 nt!IopInvalidDeviceRequest
[15] IRP_MJ_SET_SECURITY                fffff80581af4570 nt!IopInvalidDeviceRequest
[16] IRP_MJ_POWER                       fffff80515e631a0 +0xfffff80515e631a0
[17] IRP_MJ_SYSTEM_CONTROL              fffff80515e70a50 +0xfffff80515e70a50
[18] IRP_MJ_DEVICE_CHANGE               fffff80581af4570 nt!IopInvalidDeviceRequest
[19] IRP_MJ_QUERY_QUOTA                 fffff80581af4570 nt!IopInvalidDeviceRequest
[1a] IRP_MJ_SET_QUOTA                   fffff80581af4570 nt!IopInvalidDeviceRequest
[1b] IRP_MJ_PNP                         fffff80515e61240 +0xfffff80515e61240
```

Visualisation des IRP en instance d'un thread

```
lkd> !process 0 0 services.exe
PROCESS ffff950b3342b0c0
    SessionId: none  Cid: 02f4    Peb: a5a0b84000  ParentCid: 027c
    DirBase: 2660a002  ObjectTable: ffffd181ebf697c0  HandleCount: 443.
    Image: services.exe

lkd> .process ffff950b3342b0c0
Implicit process is now ffff950b`3342b0c0

lkd> !thread
THREAD ffff950b34046080  Cid 104c.1038  Teb: 000000d5e9eb8000 Win32Thread: ffff950b35374e80 WAIT: (WrUserRequest) UserMode
 Non-Alertable
    ffff950b36e12840  QueueObject
IRP List:
    ffff950b338c9530: (0006,0358) Flags: 00060800  Mdl: 00000000
    .
    .
    .

lkd> !irp ffff950b338c9530
Irp is active with 2 stacks 1 is current (= 0xffff950b338c9600)
 No Mdl: No System Buffer: Thread ffff950b34046080:  Irp stack trace.
     cmd  flg cl Device   File     Completion-Context
>[IRP_MJ_FILE_SYSTEM_CONTROL(d), N/A(0)]
            5 e1 ffff950b3237aa60 ffff950b37435dc0 fffff80512fb5150-ffff950b36654980 Success Error Cancel pending
        \FileSystem\Npfs
   Args: 00000000 00000000 00110008 00000000
 [IRP_MJ_FILE_SYSTEM_CONTROL(d), N/A(0)]
            5  0 ffff950b3234c2f0 ffff950b37435dc0 00000000-00000000
        \FileSystem\FltMgr
   Args: 00000000 00000000 00110008 00000000

lkd> dx -r1 -nv (*((nt!_IO_STACK_LOCATION *) 0xffff950b338c9600))
(*((nt!_IO_STACK_LOCATION *)0xffff950b338c9600))                 : IRP_MJ_FILE_SYSTEM_CONTROL / 0x0 for Device for
 "\FileSystem\Npfs" [Type: _IO_STACK_LOCATION]
    [+0x000] MajorFunction    : 0xd [Type: unsigned char]
    [+0x001] MinorFunction    : 0x0 [Type: unsigned char]
    [+0x002] Flags            : 0x5 [Type: unsigned char]
    [+0x003] Control          : 0xe1 [Type: unsigned char]
    [+0x008] Parameters       [Type: <unnamed-tag>]
    [+0x028] DeviceObject     : 0xffff950b3237aa60 : Device for "\FileSystem\Npfs" [Type: _DEVICE_OBJECT *]
    [+0x030] FileObject       : 0xffff950b37435dc0 : "\dotnet-diagnostic-4172" - Device for "\FileSystem\Npfs" [Type:
 _FILE_OBJECT *]
    [+0x038] CompletionRoutine : 0xfffff80512fb5150 : 0xfffff80512fb5150 [Type: long (__cdecl*)(_DEVICE_OBJECT *,_IRP
 *,void *)]
    [+0x040] Context          : 0xffff950b36654980 [Type: void *]
```

Gestionnaire de périphériques

L'emplacement central pour le contrôle de l'état des périphériques installés, ainsi que pour la configuration des pilotes associés, est le Gestionnaire de périphériques.

Pour ouvrir le gestionnaire de périphériques, appliquez l'une des techniques suivantes : - Depuis une invite de commandes, saisissez devmgmt.msc.

- Depuis le menu technique (touche Windows + X au clavier ou en effectuant en clic droit sur l'icône Windows, sélectionnez Gestionnaire de périphériques.
- Depuis le Panneau de configuration, ouvrez Système et cliquez sur le lien Gestionnaire de périphériques dans le volet gauche de la fenêtre.
- Depuis la console Gestion de l'ordinateur (compmgmt.msc), sélectionnez Gestionnaire de périphériques, sous Outils système.

Outre les informations de base, la boite de dialogue des propriétés d'un périphérique donné peut comporter plusieurs onglets personnalisés. Pour le pilote de classe de stockage de masse USB (Usbstor.sys), par exemple, un onglet personnalisé nommé Gestion de l'alimentation permet de contrôler les états d'alimentation des périphériques associées.

Pour afficher des informations complémentaires sur un pilote installé, cliquez sur le bouton Détail du pilote. La boîte de dialogue qui s'ouvre alors procure différentes informations concernant tous les fichiers associées. Sélectionnez un nom de fichier dans la liste de façon à obtenir les détails sur celui-ci dans la partie inférieure de la boite de dialogue.

Afficher les périphériques déconnectés. Par défaut, le Gestionnaire de périphériques n'affiche que les périphériques actuellement connectés au système. Pour contourner ce problème et rendre visible tous les périphériques, y compris les périphériques virtuels (IPsec, NetBIOS, etc.), procédez comme suit.

1. Ouvrez une fenêtre d'invite de commandes.
2. À l'invite de commandes, tapez la ligne suivante, puis appuyez sur Entrée : set devmgr_show_nonpresent_devices=1.
3. Saisissez devmgmt.msc pour ouvrir le Gestionnaire de périphériques,
4. Cliquez ensuite sur Affichage dans la barre de menu puis sur Afficher les périphériques cachés. Les périphériques déconnectés du système, mais dont les pilotes sont installés, apparaissent en grisé.

Pilotes de périphériques

Pour masquer les originalités et les aspects spécifiques des divers périphériques, Windows est structuré de telle façon à utiliser des sous-programmes spécialisés dits pilotes de périphériques (*device drivers*), lesquels offrent une interface uniforme d'accès au sous-système d'E/S du noyau.

Chaque pilote Windows peut avoir sous son contrôle un ou plusieurs périphériques d'entrées/sorties, mais un pilote peut également prendre en charge des aspects qui ne sont apparentés à aucun périphérique spécifique, y compris par exemple le chiffrement de flux de données ou l'interaction avec des structures de données du noyau.

En règle générale, chaque pilote se charge de traduire des commandes intelligibles dans une perpective fichier (par exemple, lire ou écrire) en des commandes compréhensibles pour le périphérique, et inversement.

Une large majorité des pilotes de périphériques sont écrits en C. Certains sont écrits en C++, bien qu'il n'existe parmi les environnements de programmation de pilotes aucun support spécifique à ce langage. En ce qui concerne l'assembleur, son utilisation est fortement déconseillée, compte tenu d'une part de la complexité qu'il introduit (tant au niveau de la conception que de la maintenance), et de la difficulté qu'il apporte dans l'éventualité d'un portage entre différentes architectures matérielles.

Les pilotes de périphérique sous Windows ne manipulent pas directement le matériel, mais sollicitent les fonctions d'une couche logicielle spéciale nommée HAL (Hardware Abstraction Layer), laquelle s'occupe de

la gestion et du pilotage des mécanismes dépendants du matériel et de l'architecture. Les pilotes peuvent ainsi, moyennant bon usage des routines HAL, être portables au niveau source sur les diverses architectures processeur reconnues par Windows, et être portables au niveau binaire au sein de la même famille d'architecture.

Les pilotes sont exécutés en mode noyau ou en mode utilisateur. Les pilotes mode noyau bénéficient des privilèges exceptionnellement haut, et ont à ce titre de grandes responsabilités concernant la fiabilité, la sécurité et la robustesse du système d'exploitation. Fonctionnant à un niveau de privilège moindre, les pilotes mode utilisateur tendent à minimiser certains aspects critiques de la gestion des périphériques. Séparés du noyau, de tels pilotes ne peuvent en cas de défaillance altérer la mémoire noyau, ou provoquer des défaillances à l'échelle du système. Ils ont par contre, du fait de nombreux changements de contexte, pour travers des performances plus faibles que leurs homologues en mode noyau.

Si les pilotes de périphériques vont généralement de pair avec avec des pièces spécifiques de matériel informatique, certains programmeurs les utilisent surtout comme moyen d'accéder à des fonctions ou structures de données internes du système d'exploitation qui ne sont pas accessibles depuis le mode utilisateur. C'est notamment le cas de divers utilitaires de niveau système qui, pourtant sans lien avec le matériel, combinent un pilote de périphérique, qui réalise ce à quoi est destinée le logiciel et invoque à ce titre un certain nombre de routines noyau, et une application Windows, qui en tant que dispositif de dialogue homme-machine rend visible à l'utilisateur les fonctionnalités du logiciel.

Types de pilotes

Il existe moult types de pilotes, que l'on peut diviser en plusieurs catégories de base :

- *Pilote d'amorçage* Pilote dont la présence est indispensable pour amorcer le système.
- *Pilote d'imprimante* Pilote qui traduit les opérations graphiques indépendantes du périphérique (exprimées sous forme de requêtes GDI) en requêtes spécifiques à l'imprimante. Ces commandes sont ensuite généralement envoyées à un pilote de port, tel le pilote de port parallèle (Parport.sys) ou le pilote de port imprimante USB (Usbprint.sys).
- *Pilote de système de fichiers* Pilote qui accepte les requêtes d'E/S portant sur des fichiers et y répond en émettant ses propres requêtes, destinées aux périphériques de stockage ou, le cas échant, aux pilotes réseau.
- *Pilote de système de fichiers local* Pilote qui gère les volumes directement connectés à l'ordinateur.
- *Pilote de système de fichiers réseau* Pilote qui permet un accès aux volumes appariés à des ordinateurs distants.
- *Pilote graphique* Pilote de périphérique d'affichage ou d'impression du sous-système Windows, qui traduit les requêtes GDI (Graphics Device Interface) en requêtes spécifiques au périphérique de sortie (carte vidéo ou imprimante).
- *Pilote de port* Pilote qui implémente le traitement d'une requête d'E/S spécifique à un type de port d'E/S (ou bus), par exemple SCSI ou IDE.
- Les pilotes Plug and Play collaborent avec le matériel et s'intègrent au gestionnaire d'alimentation et au gestionnaire PnP de Windows.
- Les pilotes non Plug and Play, appelées aussi extensions noyau, étendent les fonctionnalités du système en fournissant des accès mode utilisateur aux services mode noyau. Ces pilotes ne s'intègrent pas au gestionnaire PnP, ni au gestionnaire d'alimentation.
- *Pilote de bus* Pilote qui dessert un contrôleur de bus, un adaptateur, un pont ou tout autre périphérique ayant des périphériques enfant. Exemples de bus : PCMCIA, PCI, USB, FireWire et ISA. Un pilote de bus prend en charge la détection et le signalement au gestionnaire PnP des différents périphériques attachés au bus qu'il supervise.

- *Pilote de fonction* Pilote principal d'un périphérique, pour lequel il fournit l'interface opérationnelle.
- *Pilote de classe* Type de pilote mode noyau qui implémente les fonctionnalités communes (traitement des E/S notamment) pour une certaine classe de périphériques (disque, bande, CD-ROM, USB, et d'autres).
- *Pilote de miniport* Type de pilote mode noyau qui associe une requête d'E/S générique à un type de port d'un type d'adaptateur.
- *Pilote WDM* Pilote de périphérique qui adhère au modèle de pilote Windows (WDM, Windows Driver Model).
- *Pilote WDF* Pilote de périphérique conforme à l'infrastructure de pilote Windows (WDF, Windows Driver Foundation).
- *Pilote KMDF* Pilote de périphérique qui se définit par son adhésion à KMDF, l'un des deux sous-ensembles de WDF.
- *Pilote UMDF* Pilote de périphérique qui se définit par son adhésion à UMDF, l'un des deux sous-ensembles de WDF.

Structure d'un pilote

Au plus haut niveau d'abstraction, un pilote se compose d'un ensemble de routines qui sont sollicitées de façon à traiter les diverses étapes d'une requête d'E/S.

- **Routine d'initialisation** Le gestionnaire d'E/S exécute la routine d'initiation d'un pilote, nommée en principe DriverEntry, lorsqu'il charge ledit composant dans le système d'exploitation. Le comportement typique d'une telle fonction est de donner lieu à l'ensemble des opérations en ce qui concerne les initialisations globales requises, par exemple la déclaration au gestionnaire d'E/S des autres routines que le pilote déploie.
- **Routine de service d'interruption** Dans l'éventualité où un périphérique dont le pilote est responsable génère une interruption, le dispatcher d'interruptions intégré au noyau transfère le contrôle à cette routine. Dans le modèle d'E/S de Windows, les ISR sont exécutées au niveau DIRQL, ce qui les conduit en vue de ne pas bloquer inutilement les autres interruptions de niveau inférieur à effectuer le minimum de traitement. Une ISR met en file un appel DPC, à quoi correspond une exécution au niveau DPC/dispatch, de façon à accomplir le reste des travaux nécessaires.
- **Routine de déchargement** Une routine de déchargement libère les ressources système utilisées par le pilote afin que le gestionnaire d'E/S soit en mesure de les supprimer de la mémoire. En général, la routine de déchargement procède au démantèlement des ressources acquises dans la routine d'initialisation.
- **Routine de notification d'arrêt du système** Cette routine permet au pilote d'initier différentes opérations de nettoyage au moment de l'arrêt du système.
- **Routines de ventilation (dispatch)** Les routines de ventilation constituent les principaux vecteurs opérationnels que fournit un pilote de périphérique. Le gestionnaire d'E/S, quand il est amené à faire levier dans le cadre d'une opération d'E/S, génère un IRP et envoie une notification au pilote via l'une de ses routines de ventilation.
- **Routine d'ajout de périphérique** Le gestionnaire PnP envoie une notification de pilote via cette routine chaque fois qu'il détecte un équipement associé au pilote. Au sein de cette routine, un pilote alloue généralement un objet périphérique pour afin de représenter l'équipement.
- **Routine de démarrage d'E/S** Un pilote peut, le cas échéant, définir une routine de démarrage d'E/S pour sérialiser les requêtes d'E/S lors d'un transfert de données vers ou depuis un périphérique. Dans une telle configuration, le gestionnaire d'E/S veille à ce que que le pilote ne traite qu'un seul IRP à la fois, en respectant strictement l'ordre d'arrivée des requêtes dans la file. En pratique, ce mode de traitement séquentiel

s'impose pour certains matériels ne supportant pas l'exécution concurrente, comme les claviers ou autres périphériques série.

- **Routine DPC de traitement d'interruption** Une routine DPC prend en charge la majeure partie des opérations impliqués par le traitement d'une interruption de périphérique, ce après exécution de l'ISR. La routine DPC est exécutée à un niveau interruption inférieur à celui de l'ISR.

- **Routine d'annulation d'E/S** Si une opération d'E/S peut être annulée, un pilote peut le cas échéant définir une ou des routines d'annulation. Quand le pilote reçoit un IRP pour une requête d'E/S pouvant être prise en charge dans un tel scénario, il assigne une routine d'annulation à l'IRP. Si un thread ayant émis une requête d'E/S se termine avant que la requête soit achevée ou s'il annule l'opération (avec la fonction Cancello, par exemple), le gestionnaire d'E/S exécute la routine d'annulation de l'IRP s'il y en a une d'assignée. La routine d'annulation a pour principale attribution de libérer les ressources qui ont été acquises pour l'IRP, mais aussi de terminer l'IRP avec un état Annulé.

- **Routines d'achèvement d'E/S** Un pilote empilé par dessus un autre pilote peut avoir des routines d'achèvement d'E/S qui lui signaleront que le pilote de niveau inférieur a fini de traiter un IRP. Par exemple, le gestionnaire d'E/S appelle la routine d'achèvement d'E/S d'un pilote de système de fichiers après qu'un pilote de périphérique a fini de transférer des données vers ou depuis un fichier. La routine d'achèvement informe le pilote de système de fichiers du succès, de l'échec ou de l'annulation de l'opération, lui permettant ainsi d'effectuer des tâches de nettoyage.

- **Routines de journalisation d'erreur** En cas de dysfonctionnement majeur, par exemple un bloc du disque devenu soudainement illisible, les routines de journalisation d'erreur consignent les circonstances de l'incident et en informent le gestionnaire d'E/S. Ce dernier se charge alors de transférer les données recueuillis vers un fichier spécial dédié à l'enregistrement des erreurs, assurant de la sorte la traçabilité de l'événement à des fins de diagnostic ou d'analyse ultérieure.

- **Routines d'aiguillage rapide** Dans le schéma fonctionnel classique, lorsqu'une application manipule (lit ou écrit) un fichier, le gestionnaire d'E/S donne lieu à une représentation de la requête (IRP), transmet celle-ci au pilote de système de fichiers, qui la traite en accédant éventuellement au disque. Ce mécanisme, adapté à de nombreux scénarios, présente toutefois sur le plan des performances un certain cout. Or, quand le gestionnaire de cache est impliqué (par exemple, si les données sont déjà en mémoire), il peut être opportun de court-circuiter ce schéma. C'est dans ce contexte qu'interviennent les routines d'aiguillage rapide, des points d'entrée alternatifs fournis par les pilotes (essentiellement de systèmes de fichiers) sur lesquels s'appuie le noyau (via le gestionnaire de mémoire ou le gestionnaire de cache) de sorte à accéder directement aux données en mémoire, sans recourir à l'acheminement standard par IRP.

Informations de pilote

Chaque pilote hébergé par un ordinateur exécutant Windows a un fichier de pilote de associé. Suivez la procédure ci-dessous pour voir l'emplacement du fichier de pilote et autres détails :

1. Ouvrez le Gestionnaire de périphériques. Celui-ci affiche une liste des périphériques installés, classés par défaut selon les différents types auxquels ils appartiennent.

2. Effectuez un clic secondaire sur le périphérique à gérer, puis sélectionnez l'option Propriétés pour ouvrir la boite de dialogue des propriétés du périphérique.

3. Dans l'onglet Pilote, cliquez sur Détails du pilote pour afficher la boîte de dialogue Détails des fichiers du pilote, laquelle répertorie des informations (fichiers de pilote, fournisseur, version, etc.) concernant un pilote.

Objet pilote

Un objet pilote représente un pilote individuel du système.

La commande dt du débogueur noyau vous permet d'examiner en détail le format de la structure de données sous-jacente aux différents objets de type pilote.

```
lkd> dt nt!_DRIVER_OBJECT ffffe70244078df0
   +0x000 Type             : 0n4
   +0x002 Size             : 0n336
   +0x008 DeviceObject     : 0xffffe702`44fe7030 _DEVICE_OBJECT
   +0x010 Flags            : 0x92
   +0x018 DriverStart      : 0xfffff803`69a30000 Void
   +0x020 DriverSize       : 0x38a000
   +0x028 DriverSection    : 0xffffe702`42675d10 Void
   +0x030 DriverExtension  : 0xffffe702`44078f40 _DRIVER_EXTENSION
   +0x038 DriverName       : _UNICODE_STRING "\FileSystem\Ntfs"
   +0x048 HardwareDatabase : 0xfffff803`d7fb7dd0 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
   +0x050 FastIoDispatch   : 0xfffff803`69ac9d00 _FAST_IO_DISPATCH
   +0x058 DriverInit       : 0xfffff803`69d160f0     long  +fffff80369d160f0
   +0x060 DriverStartIo    : (null)
   +0x068 DriverUnload     : (null)
   +0x070 MajorFunction    : [28] 0xfffff803`69bb8ea0     long  +fffff80369bb8ea0
```

- *Objet périphérique (DeviceObject)* Pointeur vers l'objet périphérique que le pilote a créé en vue de représenter un périphérique. Ce champ est automatiquement mis à jours lors de l'appel à IoCreateDevice. Il est possible d'utiliser cet attribut, combiné au champ NextDevice de la structure DEVICE_OBJECT, pour découvrir une liste de tous les objets périphérique auxquels un pilote a donné lieu.

- *Routine d'initialisation (DriverInit)* Adresse de la routine d'initialisation du pilote. Voir routine IopLoadDriver.

- *Nom du pilote (DriverName)* Nom par lequel se fait connaitre le pilote auprès des autres composants du système d'exploitation. Voir routine IopLoadDriver.

- *Taille virtuelle du pilote (DriverSize)* Taille qu'occupe le pilote une fois chargé dans l'espace d'adressage virtuel du système. Correspond au champ SizeOfImage de la structure IMAGE_OPTIONAL_HEADER. Voir routine IopLoadDriver.

- **Adresse de démarrage (DriverStart)** Adresse de base en mémoire où l'image du pilote a été chargée par le système.

- *Routine de démarrage d'E/S (DriverStartIo)* Adresse de la routine de démarrage d'E/S du pilote.

- **Routine de déchargement (DriverUnload)** Pointeur vers une fonction invoquée lors de la cessation d'activité d'un pilote, permettant d'exécuter les opérations nécessaires à la libération des ressources qu'il a mobilisées pour son exécution : mémoire allouée, fichiers ouverts, requêtes d'E/S en attente, etc. Voir routine IopLoadDriver.

- **Base de données du matériel (HardwareDatabase)** Chemin du registre Windows où sont stockées les informations relatives à la configuration matérielle du système. Cette base de données centralise les descriptions des périphériques, leurs identifiants, ainsi que divers paramètres nécessaires à leur initialisation et à leur gestion par le noyau et les pilotes. Voir HKLM\Hardware\Description\System. Voir variable noyau CmRegistryMachineHardwareDescriptionSystemName.

- **Routage rapide des E/S (FastIoDispatch)** Pointeur vers une structure regroupant l'ensemble des pointeurs de fonction que le pilote a éventuellement spécifié pour gérer les opérations d'entrée/sortie via le chemin rapide (mécanisme Fast I/O), permettant un traitement optimisé des requêtes.

- **Indicateurs d'état (Flags)** Ensemble de bits de contrôle caractérisant l'état et les propriétés du pilote, y compris des renseignements sur la prise en charge de fonctionnalités spécifiques, la présence d'une routine de déchargement ou encore l'appartenance à une catégorie particulière de pilotes, tels que les pilotes hérités ou les pilotes intégrés au noyau. Voir DRVO_LEGACY_DRIVER, DRVO_UNLOAD_INVOKED.

- **Taille (Size)** Taille de la structure `DRIVER_OBJECT`, comptabilisée en nombre d'octets. Voir routine IopLoadDriver.

- Type d'objet *Type* Type de l'objet tel qu'il est défini dans la perspective du gestionnaire d'E/S, ici IO_TYPE_DRIVER.
- *Fonctions de pilote (MajorFunction)* Tableau de pointeurs vers des routines à appeler en réponse aux différentes requêtes d'E/S que le pilote reçoit.

Tableau 14.3. DRIVER_OBJECT Flags

Constante	Valeur
DRVO_UNLOAD_INVOKED	0x01
Indique que la routine de déchargement du pilote a été appelée.	DRVO_LEGACY_DRIVER
0x02	Identifie un pilote hérité (non-PnP).
DRVO_BUILTIN_DRIVER	0x04
Indique un pilote intégré au noyau, qui ne peut pas être déchargé (ex. : pilotes critiques du système).	DRVO_REINIT_REGISTERED
0x08	Signale que le pilote a enregistré une fonction de réinitialisation (fonction `IoRegisterDriverReinitialization`).
DRVO_INITIALIZED	0x10
Le pilote a terminé son initialisation.	DRVO_BOOTREINIT_REGISTERED
0x20	Indique que le pilote a enregistré une routine de réinitialisation à l'amorçage (fonction `IoRegisterBootDriverReinitialization`).
DRVO_LEGACY_RESOURCES	0x40
Utilisé par les pilotes hérités en vue de caractériser l'utilisation de ressources matérielles directes.	DRVO_BASE_FILESYSTEM_DRIVER
0x80	Marque un pilote de base de système de fichiers (ex. : `ntfs.sys`, `fastfat.sys`). Ces pilotes gèrent les systèmes de fichiers fondamentaux.

Objets périphérique

Lorsqu'il s'agit de représenter un équipement physique (comme un disque dur ou un clavier) ou logique (tel un volume de fichiers), le système emploie à cet effet un *objet périphérique*. Ce dernier reflète les caractéristiques propres à l'appareil concerné (exigences d'alignement mémoire, organisation de ses files d'attentes, etc.) et sert de cible aux opérations d'E/S qui lui sont sont destinées.

Quand un pilote crée un objet périphérique, il peut facultativement assigner un nom au périphérique. Un nom, comme à l'accoutumée en pareil contexte, octroie à l'objet une place au sein de l'espace de noms du gestionnaire d'objets, le rendant ainsi accessible via un chemin symbolique. Un pilote peut soit définir explicitement un nom soit laisser le gestionnaire d'E/S en générer un automatiquement. Par convention, les objets périphérique nommés sont placés dans le répertoire `\Device`, à la racine de l'espace de noms.

Pour permettre aux applications d'accéder à un périphérique, un lien symbolique situé dans le répertoire `\GLOBAL??` doit pointer vers son nom réel au sein de la hiérarchie `\Device`. Les pilotes non PnP (comme ceux de système de fichiers) emploient généralement des noms fixes (par exemple `\Device\HarddiskVolume1`), faciles à repérer. Les pilotes PnP, quant à eux, s'appuient sur des identifiants pseudo-aléatoires (GUID), plus souples dans un environnement dynamique, où le matériel peut apparaître ou disparaître à tout moment.

```
lkd> dt nt!_DEVICE_OBJECT
   +0x000 Type             : Int2B
   +0x002 Size             : Uint2B
   +0x004 ReferenceCount   : Int4B
   +0x008 DriverObject     : Ptr64 _DRIVER_OBJECT
   +0x010 NextDevice       : Ptr64 _DEVICE_OBJECT
   +0x018 AttachedDevice   : Ptr64 _DEVICE_OBJECT
   +0x020 CurrentIrp       : Ptr64 _IRP
```

```
+0x028 Timer            : Ptr64 _IO_TIMER
+0x030 Flags            : Uint4B
+0x034 Characteristics  : Uint4B
+0x038 Vpb              : Ptr64 _VPB
+0x040 DeviceExtension  : Ptr64 Void
+0x048 DeviceType       : Uint4B
+0x04c StackSize        : Char
+0x050 Queue            : <unnamed-tag>
+0x098 AlignmentRequirement : Uint4B
+0x0a0 DeviceQueue      : _KDEVICE_QUEUE
+0x0c8 Dpc              : _KDPC
+0x108 ActiveThreadCount : Uint4B
+0x110 SecurityDescriptor : Ptr64 Void
+0x118 DeviceLock       : _KEVENT
+0x130 SectorSize       : Uint2B
+0x132 Spare1           : Uint2B
+0x138 DeviceObjectExtension : Ptr64 _DEVOBJ_EXTENSION
+0x140 Reserved         : Ptr64 Void
```

- *Exigence d'alignement (AlignmentRequirement)* Spécifie l'alignement exigé lors des transferts DMA. Voir routine GetDmaAlignment.

- *Characteristics* Voir routine IoCreateDevice (paramètre DeviceCharacteristics), constantes FILE_REMOVABLE_MEDIA, FILE_READ_ONLY_DEVICE, FILE_FLOPPY_DISKETTE, FILE_WRITE_ONCE_MEDIA, FILE_REMOTE_DEVICE, FILE_DEVICE_IS_MOUNTED, FILE_VIRTUAL_VOLUME, FILE_AUTOGENERATED_DEVICE_NAME.

- *Verrou de périphérique (DeviceLock)* Voir routine IoCreateDevice.

- *Type de périphérique (DeviceType)* Type de matériel sous-jacent au périphérique que l'objet périphérique représente, par exemple un disque, un clavier, une carte vidéo, etc. Voir service NtQueryVolumeInformationFile (FsInformationClass = FileFsDeviceInformation), routine IoCreateDevice (paramètre DeviceType), constantes FILE_DEVICE_CD_ROM, FILE_DEVICE_DFS, FILE_DEVICE_DISK, FILE_DEVICE_NETWORK_FILE_SYSTEM et FILE_DEVICE_VIRTUAL_DISK.

- *Extension de périphérique (DeviceExtension)*

- *Compteur de référence (ReferenceCount)* Utilisé par le gestionnaire d'E/S pour suivre le nombre de handles ouverts sur un périphérique, évitant ainsi au système de décharger un pilote qui encadre son fonctionnement.

- *Minuterie (Timer)* Voir routine IoInitializeTimer.

Tableau 14.4. DEVICE_OBJECT Characteristics

Constante	Valeur	Description
FILE_AUTOGENERATED_DEVICE_NAME	0x00000080	-
FILE_CHARACTERISTIC_CSV	0x00010000	Le périphérique est un volume partagé de cluster (CSV, Cluster Shared Volume).
FILE_CHARACTERISTIC_PNP_DEVICE	0x00000800	-
FILE_DEVICE_IS_MOUNTED	0x00000020	Le périphérique a un point de montage qui lui est associé. Voir service NtQueryVolumeInformationFile (FsInformationClass = FileFsDeviceInformation).
FILE_DEVICE_SECURE_OPEN	0x00000100	-
FILE_FLOPPY_DISKETTE	0x00000004	Le périphérique est un lecteur de disquette.
FILE_REMOVABLE_MEDIA	0x00000001	Le périphérique prend en charge les supports amovibles.
FILE_REMOTE_DEVICE	0x00000010	Le périphérique est accessible via une connexion réseau.

FILE_READ_ONLY_DEVICE	0x00000002	Le périphérique est protégé contre l'écriture.
FILE_VIRTUAL_VOLUME	0x00000040	-
FILE_WRITE_ONCE_MEDIA	0x00000008	Le périphérique prend en charge les supports en écriture unique.

Tableau 14.5. DEVICE_OBJECT Flags

Constante	Valeur	Description
DO_BUFFERED_IO	0x00000004	Indique l'utilisation des E/S bufferisées.
DO_DEVICE_HAS_NAME	0x00000040	Voir routine IoCreateDevice (paramètre DeviceName).
DO_DEVICE_INITIALIZING	0x00000080	-
DO_DIRECT_IO	0x00000010	-
DO_EXCLUSIVE	0x00000008	Voir routine IoCreateDevice (Exclusive = TRUE)
DO_POWER_INRUSH	0x00004000	-
DO_POWER_PAGABLE	0x00002000	-

Types de périphériques

Le gestionnaire d'E/S de Windows classe les périphériques en différents types, et demandent aux pilotes qui en prennent en charge de spécifier une valeur de type de périphérique appropriée lors de l'appel à IoCreateDevice. En interne, la routine IoCreateDevice utilise le type de périphérique fourni pour initialiser l'attribut DeviceType de la structure DEVICE_OBJECT.

Tableau 14.6. DEVICE_OBJECT DeviceType

Constante	Valeur
FILE_DEVICE_8042_PORT	00000027
FILE_DEVICE_ACPI	00000032
FILE_DEVICE_BATTERY	00000029
FILE_DEVICE_BEEP	00000001
FILE_DEVICE_BUS_EXTENDER	0000002a
FILE_DEVICE_CD_ROM	00000002
FILE_DEVICE_CD_ROM_FILE_SYSTEM	00000003
FILE_DEVICE_CHANGER	00000030
FILE_DEVICE_CONTROLLER	00000004
FILE_DEVICE_DATALINK	00000005
FILE_DEVICE_DFS	00000006
FILE_DEVICE_DFS_FILE_SYSTEM	00000035
FILE_DEVICE_DFS_VOLUME	00000036
FILE_DEVICE_DISK	00000007
FILE_DEVICE_DISK_FILE_SYSTEM	00000008
FILE_DEVICE_DVD	00000033
FILE_DEVICE_FILE_SYSTEM	00000009
FILE_DEVICE_FIPS	0000003a
FILE_DEVICE_FULLSCREEN_VIDEO	00000034
FILE_DEVICE_INPORT_PORT	0000000a
FILE_DEVICE_KEYBOARD	0000000b
FILE_DEVICE_KS	0000002f
FILE_DEVICE_KSEC	00000039
FILE_DEVICE_MAILSLOT	0000000c

FILE_DEVICE_MASS_STORAGE	0000002d
FILE_DEVICE_MIDI_IN	0000000d
FILE_DEVICE_MIDI_OUT	0000000e
FILE_DEVICE_MODEM	0000002b
FILE_DEVICE_MOUSE	0000000f
FILE_DEVICE_MULTI_UNC_PROVIDER	00000010
FILE_DEVICE_NAMED_PIPE	00000011
FILE_DEVICE_NETWORK	00000012
FILE_DEVICE_NETWORK_BROWSER	00000013
FILE_DEVICE_NETWORK_FILE_SYSTEM	00000014
FILE_DEVICE_NETWORK_REDIRECTOR	00000028
FILE_DEVICE_NULL	00000015
FILE_DEVICE_PARALLEL_PORT	00000016
FILE_DEVICE_PHYSICAL_NETCARD	00000017
FILE_DEVICE_PRINTER	00000018
FILE_DEVICE_SCANNER	00000019
FILE_DEVICE_SCREEN	0000001c
FILE_DEVICE_SERENUM	00000037
FILE_DEVICE_SERIAL_MOUSE_PORT	0000001a
FILE_DEVICE_SERIAL_PORT	0000001b
FILE_DEVICE_SMARTCARD	00000031
FILE_DEVICE_SMB	0000002e
FILE_DEVICE_SOUND	0000001d
FILE_DEVICE_STREAMS	0000001e
FILE_DEVICE_TAPE	0000001f
FILE_DEVICE_TAPE_FILE_SYSTEM	00000020
FILE_DEVICE_TERMSRV	00000038
FILE_DEVICE_TRANSPORT	00000021
FILE_DEVICE_UNKNOWN	00000022
FILE_DEVICE_VDM	0000002c
FILE_DEVICE_VIDEO	00000023
FILE_DEVICE_VIRTUAL_DISK	00000024
FILE_DEVICE_WAVE_IN	00000025
FILE_DEVICE_WAVE_OUT	00000026

Requête d'E/S destinée à un pilote mono sphère

Le traitement d'un E/S synchrone adressée à un pilote unique s'effectue en plusieurs étapes, dont voici le déroulement général.

1. Une application en mode utilisateur invoque une fonction de l'API Windows émettrice d'E/S (par exemple `ReadFile`), passant à son tour le relais à un service natif (ici, `NtReadFile`) qui entraine une transition contrôlée vers le mode noyau.

2. Le noyau, en s'appuyant sur le handle de fichier transmis lors de l'appel initial (handle qui est lié à un objet fichier, lui-même rattaché à un pilote), identifie le pilote impliqué par la requête.

3. Le gestionnaire d'E/S alloue un IRP pour représenter la requête et l'envoie à la routine d'aiguillage appropriée du pilote, via la fonction système `IoCallDriver`. Dans ce cas précis, c'est la routine associée à une opération d'E/S lecture (`IRP_MJ_READ`) qui sera sollicitée. Chaque type de requête majeur correspond à un index dans le tableau des routines d'aiguillage du pilote.

4. Le pilote reçoit l'IRP. En règle générale, cet appel effectué dans le contexte du thread appelant, autrement dit celui dont émane l'opération initiale. Le pilote s'exécute donc dans le même thread et avec le même contexte (pile, accès mémoire, etc.), ce qui est tout particulièrement important pour l'accès aux tampons qui accompagnent l'E/S.

5. Différents contrôles de validation sont mis en oeuvre (si le gestionnaire d'E/S en a à ce stade déjà effectués quelques-uns, il ignore le sens précis de la requête) : vérification des tampons d'entrée (dans le cas d'une requête d'E/S lecture) et de sortie (E/S écriture), approbation ou rejet du code de contrôle fourni, etc. Si l'une de ces vérifications échoue (paramètres invalides, taille incorrecte, code non reconnu…), le pilote termine immédiatement l'IRP en appelant `IoCompleteRequest` avec un statut d'échec, puis retourne sans plus de délai.

6. Le pilote notifie le gestionnaire d'E/S (toujours avec `IoCompleteRequest`) pour l'informer avoir conduit la requête jusqu'à son terme, après quoi le thread revient en mode utilisateur. L'appel à l'API Windows initiale (`ReadFile` dans notre exemple) se conclut alors, en retournant à l'appelant un indicateur de succès ou d'échec.

E/S rapides (*Fast I/O*)

Les opérations de lecture et d'écriture peuvent si les circonstances s'y prêtent être traitées au sein d'une procédure spéciale basée sur des E/S dites rapides (Fast I/O), lesquelles permettent de contourner la génération d'IRP et, à la place, d'aller directement vers le pilote de système de fichiers ou vers le gestionnaire de cache. Une telle méthode offre deux deux avantages majeurs ; d'une part la réduction du nombre de phases requises pour mener à bien une requête d'E/S, d'autre part la diminution des coûts inhérents à la création d'un IRP.

Les services d'E/S rapides imposent trois exigences fondamentales : (1) que le fichier soit présent dans le cache (La première lecture ou écriture dans un fichier ne peut dès lors donner lieu à une E/S rapide), (2) que la requête soit synchrone, et (3) que le bloc de données demandé ne contient aucune plage d'octets verrouillée (voir entrée FastIoCheckIfPossible de la table Fast I/O).

Seuls les pilotes de système de fichiers et les pilotes de transport réseau ont généralement l'utilité des E/S rapides. Un pilote déclare ses points d'entrée Fast I/O en les enregistrant dans une structure pointée par l'attribut FastIoDispatch de son objet pilote.

Visualisation des routines Fast I/O d'un pilote. La commande !drvobj du débogueur noyau permet d'énumérer les services d'E/S rapides qu'un pilote met en oeuvre. La sortie suivante montre les points d'entrée de la table Fast I/O du pilote du système de fichiers NTFS.

```
lkd> !drvobj \filesystem\ntfs 2
Driver object (ffffda8e1a211b40) is for:
 \FileSystem\Ntfs

.
.
.
Fast I/O routines:
FastIoCheckIfPossible                   fffff8052eb9e020 Ntfs!NtfsFastIoCheckIfPossible
FastIoRead                              fffff8052ea8e080 Ntfs!NtfsCopyReadA
FastIoWrite                             fffff8052ea5cb00 Ntfs!NtfsCopyWriteA
FastIoQueryBasicInfo                    fffff8052eac50d0 Ntfs!NtfsFastQueryBasicInfo
FastIoQueryStandardInfo                 fffff8052eac2de0 Ntfs!NtfsFastQueryStdInfo
FastIoLock                              fffff8052eac6160 Ntfs!NtfsFastLock
FastIoUnlockSingle                      fffff8052eac6b40 Ntfs!NtfsFastUnlockSingle
FastIoUnlockAll                         fffff8052eb9d300 Ntfs!NtfsFastUnlockAll
FastIoUnlockAllByKey                    fffff8052eb9d5c0 Ntfs!NtfsFastUnlockAllByKey
ReleaseFileForNtCreateSection           fffff8052e9b3670 Ntfs!NtfsReleaseForCreateSection
FastIoQueryNetworkOpenInfo              fffff8052eac4cb0 Ntfs!NtfsFastQueryNetworkOpenInfo
AcquireForModWrite                      fffff8052e9b4c20 Ntfs!NtfsAcquireFileForModWrite
MdlRead                                 fffff8052ea5b6a0 Ntfs!NtfsMdlReadA
MdlReadComplete                         fffff80529cfeca0 nt!FsRtlMdlReadCompleteDev
PrepareMdlWrite                         fffff8052ea5aae0 Ntfs!NtfsPrepareMdlWriteA
MdlWriteComplete                        fffff8052a2a81c0 nt!FsRtlMdlWriteCompleteDev
```

```
FastIoQueryOpen                         fffff8052eac4940 Ntfs!NtfsNetworkOpenCreate
ReleaseForModWrite                      fffff8052e9b5a40 Ntfs!NtfsReleaseFileForModWrite
AcquireForCcFlush                       fffff8052e998690 Ntfs!NtfsAcquireFileForCcFlush
ReleaseForCcFlush                       fffff8052e9b5610 Ntfs!NtfsReleaseFileForCcFlush

lkd> dt nt!_DRIVER_OBJECT FastIoDispatch ffffda8e1a211b40
   +0x050 FastIoDispatch : 0xfffff805`2ea1ea60 _FAST_IO_DISPATCH

lkd> dt nt!_FAST_IO_DISPATCH 0xfffff805`2ea1ea60
   +0x000 SizeOfFastIoDispatch : 0xe0
   +0x008 FastIoCheckIfPossible : 0xfffff805`2eb9e020     unsigned char  Ntfs!NtfsFastIoCheckIfPossible+0
   +0x010 FastIoRead       : 0xfffff805`2ea8e080     unsigned char  Ntfs!NtfsCopyReadA+0
   +0x018 FastIoWrite      : 0xfffff805`2ea5cb00     unsigned char  Ntfs!NtfsCopyWriteA+0
   +0x020 FastIoQueryBasicInfo : 0xfffff805`2eac50d0     unsigned char  Ntfs!NtfsFastQueryBasicInfo+0
   +0x028 FastIoQueryStandardInfo : 0xfffff805`2eac2de0     unsigned char  Ntfs!NtfsFastQueryStdInfo+0
   +0x030 FastIoLock       : 0xfffff805`2eac6160     unsigned char  Ntfs!NtfsFastLock+0
   +0x038 FastIoUnlockSingle : 0xfffff805`2eac6b40     unsigned char  Ntfs!NtfsFastUnlockSingle+0
   +0x040 FastIoUnlockAll  : 0xfffff805`2eb9d300     unsigned char  Ntfs!NtfsFastUnlockAll+0
   +0x048 FastIoUnlockAllByKey : 0xfffff805`2eb9d5c0     unsigned char  Ntfs!NtfsFastUnlockAllByKey+0
   +0x050 FastIoDeviceControl : (null)
   +0x058 AcquireFileForNtCreateSection : (null)
   +0x060 ReleaseFileForNtCreateSection : 0xfffff805`2e9b3670     void  Ntfs!NtfsReleaseForCreateSection+0
   +0x068 FastIoDetachDevice : (null)
   +0x070 FastIoQueryNetworkOpenInfo : 0xfffff805`2eac4cb0     unsigned char  Ntfs!NtfsFastQueryNetworkOpenInfo+0
   +0x078 AcquireForModWrite : 0xfffff805`2e9b4c20     long  Ntfs!NtfsAcquireFileForModWrite+0
   +0x080 MdlRead          : 0xfffff805`2ea5b6a0     unsigned char  Ntfs!NtfsMdlReadA+0
   +0x088 MdlReadComplete  : 0xfffff805`29cfeca0     unsigned char  nt!FsRtlMdlReadCompleteDev+0
   +0x090 PrepareMdlWrite  : 0xfffff805`2ea5aae0     unsigned char  Ntfs!NtfsPrepareMdlWriteA+0
   +0x098 MdlWriteComplete : 0xfffff805`2a2a81c0     unsigned char  nt!FsRtlMdlWriteCompleteDev+0
   +0x0a0 FastIoReadCompressed : (null)
   +0x0a8 FastIoWriteCompressed : (null)
   +0x0b0 MdlReadCompleteCompressed : (null)
   +0x0b8 MdlWriteCompleteCompressed : (null)
   +0x0c0 FastIoQueryOpen  : 0xfffff805`2eac4940     unsigned char  Ntfs!NtfsNetworkOpenCreate+0
   +0x0c8 ReleaseForModWrite : 0xfffff805`2e9b5a40     long  Ntfs!NtfsReleaseFileForModWrite+0
   +0x0d0 AcquireForCcFlush : 0xfffff805`2e998690     long  Ntfs!NtfsAcquireFileForCcFlush+0
   +0x0d8 ReleaseForCcFlush : 0xfffff805`2e9b5610     long  Ntfs!NtfsReleaseFileForCcFlush+0
```

Plug and Play

L'appellation Plug and Play correspond à un ensemble de spécifications utilisé pour désigner des périphériques conçus dans l'optique d'être reconnus rapidement et automatiquement par le système d'exploitation, dès le branchement du matériel, et a fortiori sans redémarrage de l'ordinateur. Windows offre une prise en charge assez complète du Plug and Play, mais compte pour le faire pleinement sur le matériel et ses pilotes.

La procédure d'installation de matériels Plug and Play est généralement très simple. Lorsqu'un périphérique PnP est connecté à l'ordinateur, Windows le détecte, installe tous les pilotes nécessaires, et détermine pour lui quels sont les paramètres adéquats en matière de gestion des ressources (IRQ, E/S, mémoire et DMA). Il s'assure ensuite qu'aucun conflit ne puisse éclater entre le nouveau matériel et les autres périphériques Plug and Play.

Lorsque vous connectez un périphérique Plug and Play pour la première fois, Windows lit la balise d'identification Plug and Play contenue dans un microprogramme du matériel. Il confronte ensuite les informations obtenues à une liste maitre des balises correspondantes extraite des fichiers d'information du dossier \Windows\Inf. S'il trouve un pilote avec une balise correspondante, il déploie alors le ou ou les fichiers appropriés. Dans le cas contraire, l'Assistant Nouveau matériel prend le relais.

Gestionnaire Plug and Play

Le gestionnaire Plug and Play (PnP) est le principal composant permettant à Windows de reconnaitre les modifications de la configuration matérielle et de s'y adapter en conséquence. Le Gestionnaire offre les fonctionnalités que voici :

- *Détection automatique des périphériques* Le gestionnaire PnP détecte automatiquement les périphériques installés ; ce processus inclut l'énumération des périphériques reliées au système pendant le démarrage, ainsi que la détection de l'ajout ou de la suppression de périphériques pendant l'exécution.

- *Arbitrage des ressources* Un autre rôle que le gestionnaire PnP remplit est l'allocation des ressources matérielles, ce qu'il fait en collectant de prime abord les exigences en la matière (interruptions, mémoire d'E/S, registres d'E/S, et d'autres ressources spécifiques aux bus) des différents périphériques raccordés à un bus d'E/S de l'ordinateur. Par extension vis à vis de ces aspects, le gestionnaire PnP, en plus de l'affectation, est également le garant bon arbitrage des ressources. Les périphériques pouvant être ajoutés au système au cours meme de son fonctionnement, le gestionnaire PnP doit être capable de ré assigner les ressources pour répondre aux besoins des périphériques ajoutés de façon dynamique. La prolifération des périphériques branchables à chaud, et notamment des périphériques de stockage externe, justifie d'autant plus le support de ressources configurables dynamiquement.

- *Identification et chargement des pilotes* Une fois un nouveau périphérique détecté, le gestionnaire d'E/S détermine, à partir de l'identification du périphérique, si un pilote capable de gérer le périphérique est d'ores et déjà présent sur le système. Si cette éventualité se confirme, il sollicite le gestionnaire d'E/S dans l'optique de charger ledit pilote. Si aucun pilote approprié n'est trouvé, le gestionnaire PnP mode noyau en informe le gestionnaire PnP mode utilisateur, lequel prend alors le relais pour attirer l'attention de l'utilisateur sur le problème, lui demandant alors de localiser un pilote convenable.

Installation de pilotes

Dans l'éventualité où le gestionnaire PnP rencontre un périphérique pour lequel aucun pilote n'est disponible dans la configuration actuelle du système, il s'appuie alors sur le gestionnaire PnP mode utilisateur pour guider le processus d'installation. (Le gestionnaire PnP mode utilisateur est mis en oeuvre dans \Windows \System32\Umpnpmgr.dll et est exécuté en tant que service dans le processus Services.exe.)

Niveau de prise en charge Plug and Play

Lors même que Windows vise à fournir des fonctionnalités Plug and Play complètes, le niveau de prise en charge du Plug and Play par un périphérique dépend à la fois du périphérique lui-même, mais aussi des pilotes qui le caractérisent auprès du système d'exploitation. Si le périphérique prend en charge Plug and Play mais que le pilote ne le fait pas (ou inversement), le périphérique s'identifie alors comme non compatible Plug and Play.

Pour que la stratégie PnP fonctionne à son plein potentiel, les périphériques et les pilotes doivent supporter le standard PnP. Les normes industrielles pour l'identification et l'énumération des périphériques attachés aux bus forment dans ce contexte la base de la prise en charge par Windows du Plug and Play. Par exemple, la norme USB définit la façon dont les périphériques d'un bus USB s'auto identifient. La prise en charge du Plug and Play par Windows fonctionne de manière optimale sur les ordinateurs utilisant l'interface ACPI. A défaut d'un tel mécanisme, Windows prend également en charge les périphériques Plug and Play sur les ordinateurs plus anciens disposant d'options de gestion de l'alimentation APM ou d'un BIOS Plug and Play.

Par définition, tout périphérique USB ou PCMCIA est compatible Plug and Play, tout comme le sont pratiquement tous les périphériques PCI et PCIe. Les appareils connectés à un port série ou parallèle peuvent ne pas être totalement compatibles Plug and Play, et les périphériques hérités de bus ISA, totalement dépourvues de telles fonctions, ne peuvent par conséquent pas être gérés de cette manière.

Pilotes Plug and Play

Les pilotes qui gèrent la procédure Plug and Play permettent au système de reconnaître le nouveau périphérique et de procéder aux modifications de la configuration matérielle sans nécessiter, ou presque, l'intervention de l'utilisateur. Pour être compatible, un pilote doit à cet égard implémenter une routine de ventilation Plug and Play, une routine de ventilation de gestion d'alimentation et une routine d'ajout de périphérique.

Pilotes non Plug and Play

Un périphérique qui n'est pas compatible avec le Plug and Play est un périphérique qui n'intègre pas la détection automatique, et requiert de ce fait une configuration manuelle. Avant le Plug and Play, les ressources étaient configurées sur le matériel au moyen de différents micro-interrupteurs et cavaliers, et les pilotes de périphériques étaient assignés aux ressources via des fichiers de paramètres, ou quelque chose d'approchant. Avant d'installer du nouveau matériel, l'utilisateur était en conséquence tenu étudier les ressources système et deviner ou se rappeler de quels périphériques exigeaient quelles ressources.

En l'absence de fonctionnalité Plug and Play, comme le système d'exploitation ne sait pas où se trouve physiquement le matériel, certaines opérations ne peuvent dès lors être accomplies. En revanche, si l'on installe un pilote Plug and Play qui assure un minimum de compatibilité avec le périphérique, le pilote peut au moins implémenter pour le périphérique l'assignation des ressources dirigés par le gestionnaire PnP.

Parmi les pilotes qui ne sont pas compatibles avec le Plug and Play figurent les pilotes hérités. Bien que ces pilotes puissent éventuellement fonctionner sous des versions récentes de Windows, la nature de ceux-ci prive le gestionnaire PnP de toute marge de manoeuvre vis à vis de la configuration dynamique du matériel. Par exemple, supposons qu'un périphérique puisse utiliser les plages de mémoire A et B et que, lors du démarrage, le gestionnaire PnP lui assigne la plage A. Si l'on connecte ultérieurement un périphérique qui ne peut utiliser que la plage A, le gestionnaire PnP ne peut pas solliciter le pilote du premier périphérique en vue de s'adapter aux circonstances nouvelles, ce qui impliquerait le concernant d'utiliser la plage B. En plus de cela, les pilotes hérités compromettent les fonctionnalités de mise en veille ou d'hibernation d'une machine.

Dans la configuration par défaut, le Gestionnaire de périphériques présente des informations sur tous les périphériques Plug and Play installés. Pour afficher les dispositifs non Plug and Play (à quoi correspond bien évidement les périphériques qui ne prévoient pas une telle procédure), saisissez devmgmt.msc pour ouvrir le Gestionnaire de périphériques, et choisissez Afficher les périphériques cachés dans le menu Affichage. Les périphériques précédemment dissimulés apparaissent alors sous la branche Pilotes non Plug and Play.

Objet fichier

La commande dt du débogueur noyau donne une vue concrète de la structure de données sous-jacente à l'objet fichier :

```
lkd> dt nt!_FILE_OBJECT
   +0x000 Type             : Int2B
   +0x002 Size             : Int2B
   +0x008 DeviceObject     : Ptr64 _DEVICE_OBJECT
   +0x010 Vpb              : Ptr64 _VPB
   +0x018 FsContext        : Ptr64 Void
   +0x020 FsContext2       : Ptr64 Void
   +0x028 SectionObjectPointer : Ptr64 _SECTION_OBJECT_POINTERS
   +0x030 PrivateCacheMap  : Ptr64 Void
   +0x038 FinalStatus      : Int4B
   +0x040 RelatedFileObject : Ptr64 _FILE_OBJECT
   +0x048 LockOperation    : UChar
   +0x049 DeletePending    : UChar
   +0x04a ReadAccess       : UChar
   +0x04b WriteAccess      : UChar
   +0x04c DeleteAccess     : UChar
   +0x04d SharedRead       : UChar
   +0x04e SharedWrite      : UChar
   +0x04f SharedDelete     : UChar
   +0x050 Flags            : Uint4B
   +0x058 FileName         : _UNICODE_STRING
   +0x068 CurrentByteOffset : _LARGE_INTEGER
   +0x070 Waiters          : Uint4B
   +0x074 Busy             : Uint4B
   +0x078 LastLock         : Ptr64 Void
   +0x080 Lock             : _KEVENT
```

```
+0x098 Event            : _KEVENT
+0x0b0 CompletionContext : Ptr64 _IO_COMPLETION_CONTEXT
+0x0b8 IrpListLock      : Uint8B
+0x0c0 IrpList          : _LIST_ENTRY
+0x0d0 FileObjectExtension : Ptr64 Void
```

- *Contexte de complétude (CompletionContext)* Pointeur vers une une structure de données de contexte de complétude, dans l'éventualité où un objet port de complétude est associé au fichier.
- *Verrou rapide (Busy)* Attribut utilisé en tant que verrou. Voir IopAcquireFastLock.
- *Position de l'octet courant (CurrentByteOffset)* Identifie l'emplacement courant dans le fichier (valide uniquement pour les E/S synchrones).
- *Accès pour suppression (DeleteAccess)* Voir routine IoSetShareAccess.
- *Suppression en attente (DeletePending)* Indique qu'une opération de suppression concernant le fichier a été demandée. Voir structure FILE_STANDARD_INFORMATION.
- *Périphérique associé (DeviceObject)* Pointeur vers l'objet périphérique représentant le périphérique sur lequel réside le fichier. Voir routine IoGetRelatedDeviceObject.
- *Nom de fichier (FileName)* Chaîne de caractères (au format Unicode) qui identifie le fichier physique auquel se réfère l'objet fichier.
- *Paramètres (Flags)*
- *Opération de verrouillage (LockOperation)* Variable booléenne qui, du moment qu'elle est armée, indique qu'au moins une opération de verrouillage de la mémoire a eu lieu concernant l'objet fichier. Une fois définie, cet attribut conserve toujours la même valeur. Voir service NtLockFile.
- *Plan de cache privé (PrivateCacheMap)* Structure de données que le gestionnaire de cache emploie en vue de d'anticiper quelles prochaines opérations de lecture seraient susceptibles de concerner le fichier.
- *Accès en lecture (ReadAccess)* Voir routine IoSetShareAccess.
- *Pointeur d'objet section (SectionObjectPointer)*
- *Suppression partagée (SharedDelete)* Indique si plusieurs appelants peuvent ouvrir le fichier pour des opérations de suppression. Voir routines IoCheckShareAccess et IoSetShareAccess.
- *Lecture partagée (SharedRead)* Indique si plusieurs appelants peuvent ouvrir le fichier pour des opérations de lecture. Voir routines IoCheckShareAccess et IoSetShareAccess.
- *Écriture partagée (SharedWrite)* Indique si plusieurs appelants peuvent ouvrir le fichier pour des opérations d'écriture. Voir routines IoCheckShareAccess et IoSetShareAccess.
- *Type* Voir constante IO_TYPE_FILE.
- *Paramètres du volume (Vpb)* Pointeur vers un VPB (Volume Parameter Block) qui indique le volume, ou partition, sur lequel réside le fichier.
- *Nombre d'attendeurs (Waiters)* Nombre de threads qui attendent d'acquérir le verrou pour l'objet. Voir routine IopAcquireFileObjectLock.
- *Accès en écriture (WriteAccess)* Voir routine IoSetShareAccess.

Tableau 14.7. FILE_OBJECT Flags

Constante	Valeur	Description

FO_FILE_OPEN	0x00000001	-
FO_SYNCHRONOUS_IO	0x00000002	Voir routine IoIsOperationSynchronous.
FO_ALERTABLE_IO	0x00000004	-
FO_NO_INTERMEDIATE_BUFFERING	0x00000008	Voir service NtCreateFile (CreateOptions = FILE_NO_INTERMEDIATE_BUFFERING).
FO_WRITE_THROUGH	0x00000010	Voir service NtCreateFile (CreateOptions = FILE_WRITE_THROUGH).
FO_SEQUENTIAL_ONLY	0x00000020	Voir service NtCreateFile (CreateOptions = FILE_SEQUENTIAL_ONLY).
FO_CACHE_SUPPORTED	0x00000040	-
FO_NAMED_PIPE	0x00000080	-
FO_STREAM_FILE	0x00000100	Voir routine IoCreateStreamFileObject.
FO_MAILSLOT	0x00000200	-
FO_GENERATE_AUDIT_ON_CLOSE	0x00000400	-
FO_DIRECT_DEVICE_OPEN	0x00000800	-
FO_FILE_MODIFIED	0x00001000	Voir routine FsRtlCopyWrite.
FO_FILE_SIZE_CHANGED	0x00002000	-
FO_CLEANUP_COMPLETE	0x00004000	-
FO_TEMPORARY_FILE	0x00008000	Voir service NtCreateFile (FileAttributes = FILE_ATTRIBUTE_TEMPORARY), routine CcLazyWriteScan.
FO_DELETE_ON_CLOSE	0x00010000	Voir service NtCreateFile (CreateOptions = FILE_DELETE_ON_CLOSE).
FO_OPENED_CASE_SENSITIVE	0x00020000	-
FO_HANDLE_CREATED	0x00040000	Voir routine IopCreateFile.
FO_FILE_FAST_IO_READ	0x00080000	-
FO_RANDOM_ACCESS	0x00100000	Voir service NtCreateFile (CreateOptions = FILE_RANDOM_ACCESS).
FO_FILE_OPEN_CANCELLED	0x00200000	Voir routine IoCancelFileOpen.
FO_VOLUME_OPEN	0x00400000	-
FO_FILE_OBJECT_HAS_EXTENSION	0x00800000	Voir routine IoGetFileObjectFilterContext.
FO_REMOTE_ORIGIN	0x01000000	Voir routine IoIsFileOriginRemote et IoSetFileOrigin.

Énumération des périphériques

Pour gérer les relations entre les périphériques, le gestionnaire PnP crée une arborescence interne appelée arborescence des périphériques. Le gestionnaire PnP crée cette arborescence au démarrage de l'ordinateur, en utilisant les informations provenant principalement des pilotes de bus, et en assure la mise à jour au fur et à mesure des évolutions de la configuration matérielle (autrement dit tandis que des périphériques sont ajoutés ou supprimés).

Chaque noeud défini au sein de l'arborescence des périphériques est appelé nœud de périphérique (device node, ou devnode). Un devnode contient des informations sur les objets périphérique qui représentent le périphérique, plus d'autres informations enregistrées à cet emplacement par le gestionnaire PnP. Il existe par conséquent autant de devnodes que de piles de périphériques.

Compte tenu de l'énumération des périphériques, l'ordre de chargement et d'initialisation des pilotes est celui-ci :

1. Le gestionnaire d'E/S commence par exécuter la routine d'initiation de chaque pilote d'amorçage. Dans l'éventualité où un périphérique a des enfants, le gestionnaire d'E/S les énumère et signale leur présence au gestionnaire PnP. Les périphériques enfants sont configurés et démarrés uniquement si leurs pilotes sont

indispensables pour démarrer le système. Si un périphérique est géré par par un pilote qui n'est pas un pilote d'amorçage, le gestionnaire PnP crée un devnode pour le périphérique mais ne le démarre pas et ne charge pas le pilote correspondant.

2. Une fois les pilotes d'amorçage initialisés, le gestionnaire PnP parcourt l'arborescence des périphériques, charge les pilotes qui ne l'ont pas été à l'étape précédente, et démarre leurs périphériques. Au démarrage de chaque périphérique, le gestionnaire énumère les éventuels périphériques enfants associés et, le cas échéant, procède au chargement des pilotes associés. Le gestionnaire PnP charge les pilotes pour les périphériques détectés quelle que soit les paramètres définis par le pilote en matière de démarrage (valeur Start sous la clé idoine). A l'issue de cette étape, tous les périphériques Plug and Play ont leurs pilotes chargés et démarrés, à l'exception des périphériques non énumérables et des enfants de ceux-ci.

3. Le gestionnaire PnP charge tous les pilotes dont la configuration implique un démarrage en même temps que le système (system-start). Ces pilotes signalent leurs périphériques non énumérables.

4. Le gestionnaire de contrôle des services charge les pilotes marqués à démarrage automatique.

Affichage de l'arborescence des périphériques

L'utilitaire Gestionnaire de périphériques (devmgmt.msc) procure une vue simplifiée des périphériques présents sur le système. Les quelques options accessibles à partir du menu Affichage offrent la possibilité de modifier la manière dont les périphériques sont répertoriées. Voici ces options :

- *Périphériques par type* Affiche les périphériques par type de matériel (claviers, imprimantes, etc.). Le nom de la connexion est affiché sous le type. Il s'agit de la vue par défaut.
- *Périphériques par connexion* Affiche les périphériques par type de connexion, par exemple ceux connectés au bus PCI.
- *Ressources par type* Affiche l'état des ressources allouées en fonction du type de périphérique auquel elles se ramènent. Les types de ressources disponibles incluent DMA (Direct Memory Access), ports d'E/S, requêtes d'interruption (IRQ) et adresses mémoire.
- *Ressources par connexion* Affiche l'état des ressources allouées par type de connexion.
- *Afficher les périphériques cachés* Affiche les périphériques non Plug and Play, les périphériques virtuels, ainsi que les périphériques ayant été physiquement retirés d'ordinateur.

Une autre méthode pour afficher l'arborescence des périphériques consiste à employer la commande !devnode du débogueur noyau.

Attributs de niveau fichier

Le tableau suivant énumère les attributs définis dans l'API Windows relativement aux fichiers.

Attribut	Valeur	Signification
FILE_ATTRIBUTE_-ARCHIVE	0x00000020	Le fichier ou le répertoire est utilisé à des fins d'archivage. Les applications utilisent généralement cet attribut afin de marquer des éléments pour la sauvegarde ou la suppression.
FILE_ATTRIBUTE_-COMPRESSED	0x00000800	Le fichier ou le répertoire est compressé
FILE_ATTRIBUTE_-DIRECTORY	0x00000010	Le fichier est un répertoire
FILE_ATTRIBUTE_-ENCRYPTED	0x00000040	Le fichier ou le répertoire est chiffré. Pour un fichier, cela signifie que toutes les données du fichier sont compressées. Pour un dossier, cela signifie que la compression est la configuration par défaut s'appliquant aux fichiers et sous--répertoires nouvellement créés.

Attribut	Valeur	Signification
FILE_ATTRIBUTE_-HIDDEN	0x00000002	Le fichier ou le répertoire est masqué. Un fichier caché n'a d'autre particularité que celle d'être invisible à l'utilisateur et aux applications qui le représente (gestionnaire de fichiers, notamment).
FILE_ATTRIBUTE_-NORMAL	0x00000080	Le fichier ou le répertoire n'a aucun attribut défini. Cet attribut est valide uniquement s'il est utilisé de façon isolée (tous les autres le remplacent).
FILE_ATTRIBUTE_NOT_-CONTENT_INDEXED	0x00002000	Le fichier ou le répertoire ne doit pas être pris en compte par le service d'indexation de contenu.
FILE_ATTRIBUTE_-OFFLINE	0x00001000	Le fichier n'est pas disponible dans l'immédiat.
FILE_ATTRIBUTE_-READONLY	0x00000001	Le fichier est en lecture seule. Les applications peuvent lire le fichier, mais pas y écrire ou le supprimer. Cet attribut n'est pas honoré par les répertoires.
FILE_ATTRIBUTE_-REPARSE_POINT	0x00000400	-
FILE_ATTRIBUTE_-SPARSE_FILE	0x00000200	Le fichier est un fichier clairsemé (*sparse file*). Un fichier sparse est un fichier où les blocs de zéros suffisamment grands ne sont pas alloués sur le disque.
FILE_ATTRIBUTE_SYSTEM	0x00000004	Le fichier ou le répertoire fait partie de ou est utilisé exclusivement par le système d'exploitation.
FILE_ATTRIBUTE_-TEMPORARY	0x00000100	Le fichier fait office d'espace de stockage temporaire. Les applications suppriment généralement les fichiers temporaires sitôt qu'elles n'en ont plus l'utilité.

Constante	Valeur	Description
FILE_FLAG_BACKUP_SEMANTICS	0x02000000	
FILE_FLAG_DELETE_ON_CLOSE	0x04000000	Détruit le fichier une fois fermé
FILE_FLAG_NO_BUFFERING	0x20000000	Désactive la mise en cache du fichier
FILE_FLAG_OPEN_NO_RECALL	0x00100000	
FILE_FLAG_OPEN_REPARSE_POINT	0x00200000	
FILE_FLAG_OVERLAPPED	0x40000000	
FILE_FLAG_POSIX_SEMANTICS	0x00100000	L'accès au fichier s'effectue conformément aux règles définies en la matière par POSIX, par exemple la sensibilité à la casse.
FILE_FLAG_RANDOM_ACCESS	0x10000000	Les accès au fichier s'effectuent de manière aléatoire. Désactive la fonctionnalité de lecture anticipée intelligente du gestionnaire de cache et empêche ce dernier d'éliminer de son working set les pages correspondant à des vues du fichier.
FILE_FLAG_SESSION_AWARE	0x00800000	
FILE_FLAG_SEQUENTIAL_SCAN	0x08000000	L'accès au fichier s'effectue de manière séquentielle. Force le gestionnaire de cache à : (1) augmenter le nombre de pages lues via la stratégie proactive (voir section Lecture anticipée), (2) marquer comme disponible pour réutilisation les pages du cache qui se trouvent en amont du pointeur de fichier actuel (en supposant qu'aucune autre application n'utilise le fichier)
FILE_FLAG_WRITE_THROUGH	0x80000000	

Tableau 14.8. Eléments auxiliaires sous-jacents aux attributs de niveau fichier

DLL	Élément	Attribut
Kernel32	CreateFile	dwFlagsAndAttributes
Ntdll.dll	NtCreateFile	FileAttributes
	NtQueryInformationFile	FileInformationClass = FileDirectoryInformation
	FILE_DIRECTORY_INFORMATION	FileAttributes

FILE_INFORMATION_CLASS

L'énumération FILE_INFORMATION_CLASS définit différents valeurs à utiliser pour interroger ou modifier des informations se rapportant à un objet fichier.

Tableau 14.9. ProcessInformationClass

Nom	Valeur	Type de données associé
FileDirectoryInformation	01	FILE_DIRECTORY_INFORMATION
FileFullDirectoryInformation	02	FILE_FULL_DIR_INFORMATION
FileBothDirectoryInformation	03	FILE_BOTH_DIR_INFORMATION
FileBasicInformation	04	FILE_BASIC_INFORMATION
FileStandardInformation	05	FILE_STANDARD_INFORMATION
FileInternalInformation	06	FILE_INTERNAL_INFORMATION
FileEaInformation	07	FILE_EA_INFORMATION
FileAccessInformation	08	FILE_ACCESS_INFORMATION
FileNameInformation	09	FILE_NAME_INFORMATION
FileRenameInformation	10	FILE_RENAME_INFORMATION
FileLinkInformation	11	FILE_LINK_INFORMATION
FileNamesInformation	12	FILE_NAMES_INFORMATION
FileDispositionInformation	13	FILE_DISPOSITION_INFORMATION
FilePositionInformation	14	FILE_POSITION_INFORMATION
FileFullEaInformation	15	FILE_FULL_EA_INFORMATION
FileModeInformation	16	FILE_MODE_INFORMATION
FileAlignmentInformation	17	FILE_ALIGNMENT_INFORMATION
FileAllInformation	18	FILE_ALL_INFORMATION
FileAllocationInformation	19	FILE_ALLOCATION_INFORMATION
FileEndOfFileInformation	20	FILE_END_OF_FILE_INFORMATION
FileAlternateNameInformation	21	FILE_NAME_INFORMATION
FileStreamInformation	22	FILE_STREAM_INFORMATION
FilePipeInformation	23	FILE_PIPE_INFORMATION
FilePipeLocalInformation	24	FILE_PIPE_LOCAL_INFORMATION
FilePipeRemoteInformation	25	FILE_PIPE_REMOTE_INFORMATION
FileMailslotQueryInformation	26	FILE_MAILSLOT_QUERY_INFORMATION
FileMailslotSetInformation	27	FILE_MAILSLOT_SET_INFORMATION
FileCompressionInformation	28	FILE_COMPRESSION_INFORMATION
FileObjectIdInformation	29	FILE_OBJECTID_INFORMATION
FileCompletionInformation	30	
FileMoveClusterInformation	31	
FileQuotaInformation	32	FILE_QUOTA_INFORMATION
FileReparsePointInformation	33	FILE_REPARSE_POINT_INFORMATION
FileNetworkOpenInformation	34	FILE_NETWORK_OPEN_INFORMATION
FileAttributeTagInformation	35	FILE_ATTRIBUTE_TAG_INFORMATION
FileTrackingInformation	36	
FileIdBothDirectoryInformation	37	
FileIdFullDirectoryInformation	38	FILE_ID_FULL_DIR_INFORMATION
FileValidDataLengthInformation	39	FILE_VALID_DATA_LENGTH_INFORMATION
FileShortNameInformation	40	FILE_NAME_INFORMATION
FileIoCompletionNotificationInformation	41	FILE_IO_COMPLETION_NOTIFICATION_INFORMATION
FileIoStatusBlockRangeInformation	42	
FileIoPriorityHintInformation	43	FILE_IO_PRIORITY_HINT_INFORMATION

Nom	Valeur	Type de données associé
FileSfioReserveInformation	44	
FileSfioVolumeInformation	45	
FileHardLinkInformation	46	FILE_LINKS_INFORMATION
FileProcessIdsUsingFileInformation	47	FILE_PROCESS_IDS_USING_FILE_INFORMATION
FileNormalizedNameInformation	48	FILE_NAME_INFORMATION
FileNetworkPhysicalNameInformation	49	FILE_NETWORK_PHYSICAL_NAME_INFORMATION
FileIdGlobalTxDirectoryInformation	50	FILE_ID_GLOBAL_TX_DIR_INFORMATION
FileIsRemoteDeviceInformation	51	FILE_IS_REMOTE_DEVICE_INFORMATION
FileUnusedInformation	52	
FileNumaNodeInformation	53	FILE_NUMA_NODE_INFORMATION
FileStandardLinkInformation	54	FILE_STANDARD_LINK_INFORMATION
FileRemoteProtocolInformation	55	FILE_REMOTE_PROTOCOL_INFORMATION
FileRenameInformationBypassAccessCheck	56	FILE_RENAME_INFORMATION
FileLinkInformationBypassAccessCheck	57	FILE_LINK_INFORMATION
FileVolumeNameInformation	58	FILE_VOLUME_NAME_INFORMATION
FileIdInformation	59	FILE_ID_INFORMATION
FileIdExtdDirectoryInformation	60	FILE_ID_EXTD_DIR_INFORMATION
FileReplaceCompletionInformation	61	FILE_COMPLETION_INFORMATION
FileHardLinkFullIdInformation	62	FILE_LINK_ENTRY_FULL_ID_INFORMATION
FileIdExtdBothDirectoryInformation	63	FILE_ID_EXTD_BOTH_DIR_INFORMATION
FileDispositionInformationEx	64	FILE_DISPOSITION_INFORMATION_EX
FileRenameInformationEx	65	FILE_RENAME_INFORMATION
FileRenameInformationExBypassAccessCheck	66	FILE_RENAME_INFORMATION

FILE_COMPRESSION_INFORMATION.

```
typedef struct _FILE_COMPRESSION_INFORMATION {
    LARGE_INTEGER CompressedFileSize;
    USHORT CompressionFormat;
    UCHAR CompressionUnitShift;
    UCHAR ChunkShift;
    UCHAR ClusterShift;
    UCHAR Reserved[3];
} FILE_COMPRESSION_INFORMATION, *PFILE_COMPRESSION_INFORMATION;
```

FILE_DIRECTORY_INFORMATION.

```
typedef struct _FILE_DIRECTORY_INFORMATION {
    ULONG NextEntryOffset;
    ULONG FileIndex;
    LARGE_INTEGER CreationTime;
    LARGE_INTEGER LastAccessTime;
    LARGE_INTEGER LastWriteTime;
    LARGE_INTEGER ChangeTime;
    LARGE_INTEGER EndOfFile;
    LARGE_INTEGER AllocationSize;
    ULONG FileAttributes;
    ULONG FileNameLength;
    WCHAR FileName[1];
} FILE_DIRECTORY_INFORMATION, *PFILE_DIRECTORY_INFORMATION;
```

FILE_BASIC_INFORMATION.

```
typedef struct _FILE_BASIC_INFORMATION {
  LARGE_INTEGER CreationTime;
  LARGE_INTEGER LastAccessTime;
```

```
  LARGE_INTEGER LastWriteTime;
  LARGE_INTEGER ChangeTime;
  ULONG         FileAttributes;
} FILE_BASIC_INFORMATION, *PFILE_BASIC_INFORMATION;
```

Attribut	Description
ChangeTime	Date à laquelle le fichier a été renommé ou que ses attributs ont été modifiés.
CreationTime	Date à laquelle le fichier a été créé.
FileAttributes	Attributs du fichier. Voir constante FILE_ATTRIBUTE_XXX.
LastAccessTime	Date à laquelle le fichier a été accédé pour la dernière fois.
LastWriteTime	Date à laquelle le fichier a été modifié pour la dernière fois.

FILE_ALL_INFORMATION.

```
typedef struct _FILE_ALL_INFORMATION {
    FILE_BASIC_INFORMATION BasicInformation;
    FILE_STANDARD_INFORMATION StandardInformation;
    FILE_INTERNAL_INFORMATION InternalInformation;
    FILE_EA_INFORMATION EaInformation;
    FILE_ACCESS_INFORMATION AccessInformation;
    FILE_POSITION_INFORMATION PositionInformation;
    FILE_MODE_INFORMATION ModeInformation;
    FILE_ALIGNMENT_INFORMATION AlignmentInformation;
    FILE_NAME_INFORMATION NameInformation;
} FILE_ALL_INFORMATION, *PFILE_ALL_INFORMATION;
```

FILE_DISPOSITION_INFORMATION_EX.

```
#define FILE_DISPOSITION_DO_NOT_DELETE                  0x00000000
#define FILE_DISPOSITION_DELETE                         0x00000001
#define FILE_DISPOSITION_POSIX_SEMANTICS                0x00000002
#define FILE_DISPOSITION_FORCE_IMAGE_SECTION_CHECK      0x00000004
#define FILE_DISPOSITION_ON_CLOSE                       0x00000008

typedef struct _FILE_DISPOSITION_INFORMATION_EX {
  ULONG Flags;
} FILE_DISPOSITION_INFORMATION_EX, *PFILE_DISPOSITION_INFORMATION_EX;
```

FILE_NETWORK_OPEN_INFORMATION.

```
typedef struct _FILE_NETWORK_OPEN_INFORMATION {
  LARGE_INTEGER CreationTime;
  LARGE_INTEGER LastAccessTime;
  LARGE_INTEGER LastWriteTime;
  LARGE_INTEGER ChangeTime;
  LARGE_INTEGER AllocationSize;
  LARGE_INTEGER EndOfFile;
  ULONG FileAttributes;
} *PFILE_NETWORK_OPEN_INFORMATION, FILE_NETWORK_OPEN_INFORMATION;
```

FILE_IS_REMOTE_DEVICE_INFORMATION.

```
typedef struct _FILE_IS_REMOTE_DEVICE_INFORMATION {
  BOOLEAN IsRemote;
} *PFILE_IS_REMOTE_DEVICE_INFORMATION, FILE_IS_REMOTE_DEVICE_INFORMATION;
```

FILE_STANDARD_INFORMATION.

```
typedef struct _FILE_STANDARD_INFORMATION {
  LARGE_INTEGER AllocationSize;
  LARGE_INTEGER EndOfFile;
  ULONG NumberOfLinks;
  BOOLEAN DeletePending;
  BOOLEAN Directory;
```

```
} FILE_STANDARD_INFORMATION, *PFILE_STANDARD_INFORMATION;
```

FILE_MODE_INFORMATION.

```
typedef struct _FILE_MODE_INFORMATION {
    ULONG Mode;
} FILE_MODE_INFORMATION, *PFILE_MODE_INFORMATION;
```

FILE_POSITION_INFORMATION.

```
typedef struct _FILE_POSITION_INFORMATION {
  LARGE_INTEGER CurrentByteOffset;
} FILE_POSITION_INFORMATION, *PFILE_POSITION_INFORMATION;
```

Gestion des fichiers

La tableau qui suit montre l'éventail des rôles possibles concernant des fichiers.

Tableau 14.10. Opérations concernant les fichiers

Rôle	Fonction	Service	Routine
Création ou éventuellement ouverture d'un fichier	CreateFile	NtCreateFile	IoCreateFile
Ouverture d'un fichier	OpenFile	NtOpenFile	IoCreateFile
Suppression d'un fichier	DeleteFile	NtDeleteFile	
Fermeture d'un handle apparenté à un fichier	CloseHandle	NtClose	ObCloseHandle
Ecriture des pages d'un fichier mis en cache	FlushFileBuffers	NtFlushBuffersFile	IRP_MJ_FLUSH_-BUFFERS
Annulation d'une opération d'E/S	CancelIo	NtCancelIoFile	IoCancelIrp
Copie un fichier d'un emplacement vers un autre	CopyFile		
Déplacement d'un fichier d'un répertoire à un autre	MoveFile(Ex)		
Réunir des informations de base concernant un fichier	GetFileAttributes	NtQueryAttributesFile	
Réunir des informations étendues concernant un fichier	GetFileAttributesEx	NtQueryFullAttributesFile	
Obtenir la taille d'un fichier	GetFileSize	NtQueryInformationFile	
Verrouiller une plage d'octets d'un fichier	LockFile(Ex)	NtLockFile	
Libérer une plage d'octets verrouillée d'un fichier	UnlockFile	NtlockFile	
Rechercher des fichiers dans un répertoire	FindFirstFile(Ex)	NtQueryDirectoryFile	
Rechercher des fichiers dans un répertoire	FindNextFile	NtQueryDirectoryFile	
Libérer les ressources allouées lors d'une recherche de fichiers	FindClose	NtClose	

Droits d'accès aux fichiers

Tableau 14.11. Mappage générique des droits d'accès aux fichiers

Droit générique	Signification
GENERIC_EXECUTE	FILE_READ_ATTRIBUTES
	FILE_EXECUTE
	STANDARD_RIGHTS_EXECUTE
	SYNCHRONIZE
GENERIC_READ	FILE_READ_ATTRIBUTES
	FILE_READ_DATA
	FILE_READ_EA
	STANDARD_RIGHTS_READ
	SYNCHRONIZE
GENERIC_WRITE	FILE_APPEND_DATA
	FILE_WRITE_ATTRIBUTES

FILE_WRITE_DATA

FILE_WRITE_EA

STANDARD_RIGHTS_WRITE

SYNCHRONIZE

Tableau 14.12. Droits d'accès concernant les fichiers

Constante	Valeur	Description
FILE_ADD_FILE	0x2	Requis pour créer un fichier dans un répertoire.
FILE_ADD_SUBDIRECTORY	0x4	Requis pour créer un sous-répertoire.
FILE_APPEND_DATA	0x4	Requis pour ajouter des données à un fichier.
FILE_DELETE_CHILD	0x40	Requis pour supprimer un répertoire.- Autorise l'effacement du répertoire même si les objets enfants ont une ACL interdisant cette opération.
FILE_EXECUTE	0x20	Requis pour exécuter un fichier.
FILE_LIST_DIRECTORY	0x1	Requis pour lister le contenu d'un répertoire.
FILE_READ_ATTRIBUTES	0x80	Requis pour lire les attributs d'un fichier.
FILE_READ_DATA	0x1	Requis pour lire le contenu d'un fichier.
FILE_LIST_DIRECTORY	0x1	Requis pour lister un répertoire.
FILE_READ_EA	0x0008	Requis pour lire les attributs étendus d'un fichier.
FILE_WRITE_DATA	0x2	Requis pour écrire des données dans un fichier.

Ports de complétion

Un intérêt majeur parmi tous ceux offerts par les ports de complétion est que la concurrence, c'est à dire le nombre de threads d'une application qui servent activement les requêtes client, est contrôlée à l'aide du système, qui a par nature une vision plus réaliste des charges au compte de la plateforme - particulièrement au sens de l'ordonnancement.

Quand une application crée un port de complétion, elle est tenue de lui assigner une valeur de concurrence, laquelle correspond sensiblement à un niveau de concurrence souhaité (nombre d'opérations s'effectuant en même temps). Dans la pratique, cette valeur indique le nombre maximal de threads associés au port susceptibles d'être exécutés simultanément. À milles lieues d'être arbitraire, le choix de cette valeur est particulièrement délicat. L'idéal est qu'à chaque instant il y ait autant de threads actifs que de processeurs dont la machine dispose, afin d'une part ne pas bloquer d'éventuelles requêtes clients (signe d'une prise en compte de la concurrence qui a été sous-dimensionné), et d'autre part minimiser les basculements de contexte (cas inverse, où l'ordonnanceur a pour l'occasion affaire à un nombre excessif de threads).

Gestion des tampons IRP

Les tampons IRP véhiculent essentiellement trois sortes d'informations distinctes :

- Informations de contrôle, telles que le type auquel la demande appartient.
- Informations d'état, qui permettent un suivi de l'avancée d'une demande.
- Données personnalisées que les composants du système d'E/S et les applications s'échangent entre eux, cela par l'intermédiaire de tampons mémoire dédiés.

Les pilotes accèdent aux tampons de données associés à un IRP à l'aide de l'une ou l'autre des méthodes suivantes :

- **E/S tamponnées (*Buffered I/O*)** Le système alloue un tampon en mémoire non paginée et y copie les données en provenance ou à destination de l'appelant. Ce mode offre sécurité et simplicité, au prix de traitements additionnels susceptibles de nuire aux performances.

- **E/S directes (*Direct I/O*)** Le système effectue une description physique des pages qu'occupe un tampon, pages que les pilotes peuvent par la suite mapper dans l'espace d'adressage système. Ce mode limite les copies et optimise les performances pour les transferts volumineux.

- **E/S ni directes ni tamponnées (*Neither I/O*)** Le système se contente de passer au pilote un pointeur vers le tampon de l'appelant. Ce mode, où aucune protection ou vérification n'est à l'origine assurée, offre une flexibilité maximale mais impose une rigueur toute particulière.

Les méthodes susmentionnées, au-delà des avantages et inconvénients que chacune présente, sont exposées à des risques de degrés divers.

- Les E/S bufferisées sont intrinsèquement sûres. Le système atteste en toutes circonstances la validité des tampons.

- Les E/S directes comportent certains risques. Le système assure la validité des adresses mode utilisateur en verrouillant les pages en mémoire. Cependant, rien ne garantit que l'application ne modifiera pas les données sous-jacentes à ces adresses.

- Les E/S ni l'un ni l'autre impliquent une attention toute particulière. Un pilote dans cette configuration n'a en l'état aucune assurance que le pointeur soit même valide au moment où il tente d'accéder au tampon de l'appelant. L'accès se déroulera en toute sécurité si, et seulement si, des mesures adéquates sont entreprises.

Tableau 14.13. Caractéristiques comparées des trois modèles d'E/S

Caractéristique	E/S tamponnées	E/S directes	E/S ni l'un ni l'autre
Copies de données	Oui (tampon intermédiaire)	Non	Non
Adéquation aux transferts de grande taille	Moindre	Excellente	Variable
Taille typique des tampons	Petite (≤ 4 Ko)	Moyenne à grande (> 4 Ko)	Variable
Accès par le pilote	Tampon système	Mappage de la mémoire utilisateur	Accès direct à l'adresse fournie (sans validation)
Sécurité d'accès	Très élevée	Moyenne (requiert précautions)	Faible (aucune protection par le système)
Nécessite un MDL	Non	Oui	Non
Compatible avec DMA	Non	Oui	Non (sauf gestion personnalisée complexe)
IRQL d'accès au tampon	Tous	Tous	IRQL bas uniquement (paginable)
Usage typique	Opérations simples ou petits transferts	Transferts volumineux, périphériques DMA	Traitement internes non critiques

Sélection d'une méthode de gestion de tampon IRP Le type de gestion de tampon qu'effectue le gestionnaire d'E/S dépend des exigences que revendique en la matière un pilote. Pour les opérations de lecture et d'écriture, le pilote entérine ce choix en le spécifiant via l'objet périphérique associé (attribut `Flags` de la structure `DEVICE_OBJECT`). Les opérations de contrôle d'E/S (issues de l'API `DeviceIoControl`, ou de son homologue natif `NtDeviceIoControl`) reposent quant à elles sur des codes de commande internes, lesquels précisent, entre autres, la stratégie de gestion de tampon à adopter lors de la création des IRP correspondants (attribut `Method` de la macro `CTL_CODE`).

```
// Buffered I/O
DeviceObject->Flags |= DO_BUFFERED_IO;

// Direct I/O
DeviceObject->Flags |= DO_DIRECT_IO;
```

Pour chaque type de gestion de tampon le gestionnaire d'E/S place dans l'IRP des références applicables vers les emplacements des tampons d'entrée et de sortie.

Les pilotes emploient généralement des E/S bufferisées quand les appelants effectuent des requêtes dont les volumes de transfert résultants font moins d'une page (4 Ko sur les processeurs x86), et emploient des E/S directes pour les requêtes plus grosses. Seule une petite minorité de pilotes s'appuient sur des E/S ni l'un ni l'autre. ## E/S tamponnées

Des différents scénarios que Windows prend en charge en matière de gestion des E/S, la forme tamponnée est sans conteste la plus accessible. Dans une telle configuration, le gestionnaire d'E/S alloue un tampon dans le pool non paginé, tampon dont la taille est égale à celle du tampon de l'appelant. Suite à cela, en fonction de la nature de l'opération pour laquelle il est mis à contribution (E/S lecture ou E/S écriture), le gestionnaire d'E/S effectue une copie des données dans l'un ou l'autre sens. Pour les opérations d'écriture, le gestionnaire d'E/S copie les données du tampon de l'appelant dans le tampon alloué quand il crée l'IRP. Pour les opérations de lecture, le gestionnaire d'E/S copie les données du tampon alloué dans le tampon de l'utilisateur quand l'IRP se termine, puis libère le tampon alloué.

Avec des E/S tamponnées, un pilote peut accéder au tampon système depuis n'importe quel thread (l'adresse où il se situe en mémoire est valide indépendamment du contexte d'exécution) et à n'importe quel IRQL (son allocation s'est faite depuis une zone qui ne peut pas être paginée vers le disque).

E/S directes

Dans le cas d'E/S directes, le gestionnaire d'E/S verrouille en mémoire le tampon de l'utilisateur, le mettant de la sorte hors de portée des schémas de pagination, puis mémorise une description des pages physiques sous-jacentes sous la forme d'une MDL (*Memory Descriptor List*). Les pilotes qui dans le cadre de leur fonctionnement normal s'appuient sur de l'accès direct à la mémoire n'ont pas besoin de plus. Quand un pilote non DMA doit accéder au contenu du tampon, il peut le faire en mappant le tampon dans l'espace d'adressage du système.

E/S ni directes ni tamponnées

Le gestionnaire d'E/S ne gère pas de tampons, mais transmet au pilote l'adresse virtuelle du tampon de l'appelant. Pour accéder à ce tampon en toute sécurité, le pilote doit être en cours d'exécution dans le contexte du thread appelant. En règle générale, par conséquent, seuls les pilotes de haut niveau, tels les pilotes de système de fichiers, peuvent utiliser cette méthode pour accéder aux tampons.

Les pilotes utilisant les E/S nini pour accéder à des tampons qui risquent d'être situés dans l'espace d'adressage utilisateur doivent porter une attention toute particulière au fait que les adresses manipulées en ces occasions sont, d'une part, valides, d'autre part qu'elles ne référencent pas de la mémoire mode noyau. Tout manquement dans ce domaine peut engendrer des effondrements ou introduire des failles de sécurité (en permettant par exemple à une application d'accéder à de la mémoire noyau voire d'injecter du code dans le noyau). Pour contrer ces possibles dérives, le noyau fournit aux composants de l'exécutif, pilotes et autres codes système, deux fonctions centrées sur ces aspects, ProbeForRead et ProbeForWrite, qui vérifient qu'un tampon réside entièrement dans la partie mode utilisateur de l'espace d'adressage.

WDM et WDF

Les pilotes de périphériques conçus pour Windows s'appuient sur l'un ou l'autre des modèles suivants :

- Windows Driver Model (WDM)
- Windows Driver Foundation (WDF)

Le modèle de pilote Windows (WDM, Windows Driver Model) définit un ensemble de spécifications qui régissent les caractéristiques et la manière de procéder pour concevoir des pilotes de périphériques.

WDM répond à la plus grande partie des exigences en ce qui concerne les pilotes de périphériques (et, partant, les périphériques eux-mêmes), dont comment regrouper plusieurs pilotes au sein d'une hiérarchie commune,

partager du code commun, gérer les E/S émanant des composants du système, moduler le niveau d'alimentation des équipements conformes au standard Plug and Play, etc.

Les pilotes WDM interagissent avec le système d'exploitation et communiquent entre eux par l'intermédiaire d'un protocole basé sur des paquets. Ces derniers permettent de véhiculer diverses informations liées au changement des périphériques, à la gestion de l'énergie, à l'allocation ou au démantèlement de diverses ressources, et d'autres.

WDM, aussi bien sur le plan fonctionnel que structurel, repose sur une séparation des fonctionnalités générales d'une catégorie de périphériques des fonctions spécifiques à un périphérique particulier. Par exemple, les dispositifs d'interface, tels que claviers et souris, partagent des caractéristiques communes, mais diffèrent les uns des autres en termes de contrôle et de configuration

L'infrastructure de pilote Windows (WDF, Windows Driver Foundation) unifie et simplifie la gestion des périphériques. Ses principaux objectifs incluent l'évolutivité et la réutilisabilité, ce qui permet aux concepteurs de logiciels d'appliquer les mêmes concepts à différents types de pilotes et de réduire le nombre d'opérations requises pour orchestrer le fonctionnement des périphériques. (WDM, par contraste, exige une connaissance relativement étendue de nombreuses notions ne serait-ce que pour écrire un pilote de base).

Dans la perspective WDF, il existe deux types de pilotes, à quoi correspond dans les faits deux cadriciels de développement modernes et majeurs : Kernel-Mode Driver Framework (KMDF) et User-Mode Driver Framework (UMDF).

Voici, dans les grandes lignes, ce qui distingue sur le plan des améliorations, WDF de WDM :

- Possibilité de pilotes en mode utilisateur
- Possibilité de versions multiples de pilotes
- Simplification de la conception et de la réalisation de pilotes pour périphériques multifonctions
- Homogénéisation de la gestion de l'énergie
- Amélioration de la prise en compte des annulations d'E/S

KMDF procure une interface simplifiée vers les services WDM, effaçant ainsi une large partie de la complexité dudit modèle. Les pilotes KMDF s'inscrivent dans un paradigme fondée sur les événements. Ils s'appuient sur les fonctions de la bibliothèque KMDF pour effecter des tâches quand ces dernières ne sont pas spécifiques aux matériels qu'ils gèrent, par exemple la gestion générale de l'alimentation ou la synchronisation.

UMDF permet à certaines classes de pilotes d'être implémentés en tant que pilotes en mode utilisateur. De ce fait, quand un pilote UMDF connait un dysfonctionnement, le processus apparenté meurt et redémarre généralement aussitôt, de sorte que le système ne soit pas affecté : le périphérique passe simplement à l'état Indisponible tandis que le service hébergeant le pilote redémarre.

UMDF exécute chaque pilote en mode utilisateur dans ce qui est essentiellement un service en mode utilisateur et emploie ALPC pour communiquer avec un pilote intermédiaire en mode noyau qui fournit un accès réel au matériel. Les pilotes UMDF sont écrits en C++ en utilisant des classes et des sémantiques similaires à celles de COM.

Kernel-Mode Driver Framework (KMDF)

Structure d'un pilote KMDF

Bien que KMDF soit un modèle de pilote différent de WDM, la structure de pilote de base indiquée plus haut s'applique dans une large mesure autant pour l'un que pour l'autre des deux schémas. Chaque pilote est tenue de comporter les fonctions suivantes :

- *Une routine d'initialisation* Comme tout autre pilote Windows, un pilote KMDF possède une routine DriverEntry chargée de mener à bien toutes les étapes de configuration faisant partie du pilote, et de préparer les phases ultérieures.
- *Une routine d'ajout de périphérique*
- *Une ou plusieurs routines de traitement* Le fonctionnement de tout pilote KMDF est régi par des événements dont émanent des appels de méthode. Un pilote KMDF expose en vue de répondre aux événements qui animent le système un ensemble plus ou moins important de routines.

Les pilotes conformes au modèle KMDF agissent sur la base d'événements entrainant l'exécution de fonctions de rappel (callbacks). Les événements essentiels au fonctionnement du pilote ou qui nécessitent un traitement spécialisé exigent de facto une méthode de rappel. Autrement, un pilote peut autoriser l'infrastructure à accomplir une action générique par défaut. Par exemple, lors du retrait d'un périphérique amovible, un pilote a le choix soit de prendre en charge l'événement associé (EvtDeviceEject), soit d'en laisser la responsabilité à KMDF, dont le comportement par défaut est auquel cas de notifier l'utilisateur que le périphérique n'est pas éjectable. Tous les événements, cependant, n'ont pas un scénario qui leur revient. L'ajout d'un périphérique, par exemple, constitue un événement (EvtDriverDeviceAdd) pour lequel chaque pilote Plug-and-Play doit inclure un rappel spécifique. Pour finir, notez que les événements définis dans la perspective KMDF n'ont aucun rapport avec les objets Windows du même nom, exposés en l'occurence en tant que primitive de synchronisation.

Modèle de données KMDF

Comme Windows dans son ensemble, KMDF est fondé sur des objets. Chaque objet définit encapscule des méthodes (fonctions) et des propriétés (données) auxquelles les pilotes peuvent accéder, et est associé à différents événements qu'il possible de prendre en charge par l'intermédiaire des routines appropriées (système de callbacks).

Contrairement aux objets gérés par le noyau ou l'exécutif Windows, qui font référence à des types d'objet distincts et sont isolés les uns vis-à-vis des autres, les objets KMDF font partie d'une hiérarchie. Le niveau le plus élevé au sein de cette hiérarchie correspond à l'objet WDFDRIVER, correspondant à la représentation d'un pilote.

Le tableau suivant répertorie les types d'objet établis dans la perspective KMDF.

Objet	Type	Description
Liste enfant (Child list)	WDFCHILDLIST	Liste des périphériques enfants.
Collection	WDFCOLLECTION	Liste d'objets de type similaire, tels que les périphériques pour lesquels un pilote de filtre tamise les demandes.
Périphérique (Device)	WDFDEVICE	Instance d'un périphérique. Un pilote a généralement un objet WDFDEVICE pour chaque périphérique qu'il contrôle.
Tampon commun DMA (DMA common buffer)	WDFCOMMON BUFFER	Espace de mémoire auquel un pilote et un périphérique peuvent accéder de manière directe (DMA).
Contrôle DMA (DMA Enabler)	WDFDMAENABLER	Active DMA sur un canal donné pour un pilote.
Transaction DMA (DMA Transaction)	WDFDMATRANSACTION	Instance d'une transaction DMA.
Appel de procédure différée (Deferred Procedure Call)	WDFDPC	Instance d'un appel de procédure différée
Pilote (Driver)	WDFDRIVER	Représente un pilote et les paramètres le concernant, tels ses points d'entrée.
Fichier (File)	WDFFILEOBJECT	Instance d'un objet fichier au travers duquel pilotes externes et applications peuvent accéder à un périphérique, ou à un fichier enregistré dessus.
Objet générique	WDFOBJECT	Regroupe les données personnalisées définies par un pilote.

Objet	Type	Description
File d'attente d'E/S	WDFQUEUE	Instance d'une file d'attente d'E/S.
Requête d'E/S	WDFREQUEST	Instance d'une requête parmi une file d'attente d'E/S.
Cible d'E/S	WDFIOTARGET	Représente la pile de périphériques à laquelle un pilote transfère une requête d'E/S.
Interruption	WDFINTERRUPT	Instance d'une interruption que le pilote doit gérer.
Liste look-aside	WDFLOOKASIDE	Liste dynamique de tampons mémoire constituée de blocs de taille fixe alloués depuis le pool paginé ou non paginé.
Mémoire	WDFMEMORY	Mémoire utilisée par le pilote, généralement un tampon d'entrée ou de sortie associé à une requête d'E/S.
Clé de registre	WDFKEY	Instance d'une clé définie dans le Registre Windows.
Liste des ressources	WDFCMRESLIST	Liste des ressources affectées au périphérique.
Liste des plages de ressources	WDFIORESLIST	Représente une configuration possible pour un périphérique.
Liste des exigences de ressources	WDFIORESREQLIST	Représente toutes les configurations possibles pour un périphérique.
Chaîne	WDFSTRING	Instance d'une chaîne de caractères au format Unicode.
Spin lock	WDFSPINLOCK	Représente un verrou tournant, qui synchronise les accès au niveau DISPATCH_LEVEL.
Wait lock	WDFWAITLOCK	Représente un verrou d'attente, qui synchronise les accès au niveau PASSIVE_-LEVEL.
Minuterie	WDFTIMER	Instance d'une minuterie.
Périphérique USB	WDFUSBDEVICE	Instance d'un périphérique USB
Interface USB	WDFUSBINTERFACE	Identifie une interface sur périphérique USB.
Instance WMI	WDFWMIINSTANCE	Représente un bloc de données WMI individuel associé à un fournisseur.
Fournisseur WMI	WDFWMIPROVIDER	Décrit le schéma WMI pris en charge par le pilote.
Élément de travail	WDFWORKITEM	Instance d'un élément du travail.

KMDF orchestre ses propres objets en interne, leur assurant une visibilité auprès des pilotes par l'intermédiaire de handles. Gérées hors le cadre du gestionnaire d'objets Windows, les structures logicielles liées à KMDF ne peuvent par conséquent être manipulées via fonctions ObXxx du système.

Attributs d'objet

Chaque objet KMDF a un un ensemble d'attributs qui le définissent auprès de l'infrastructure, et déterminent comment il doit être pris en charge. Ces attributs sont généralement configurés à leurs valeurs par défaut, mais peuvent être remplacées par un pilote lors de la création de l'objet en spécifiant une structure WDF_OBJECT_ATTRIBUTES (similaire à la structure OBJECT_ATTRIBUTES du gestionnaire d'objets lors de la création d'un objet noyau).

Attribut	Description
ContextSizeOverride	Taille de la zone de contexte d'objet.
ContextTypeInfo	Type de la zone de contexte d'objet.

Attribut	Description
EvtCleanupCallback	Méthode appelée pour informer un pilote que le système à prépare à supprimer l'objet (des références peuvent toujours exister).
EvtDestroyCallback	Méthode appelée pour informer un pilote de la suppression imminente de l'objet (le compteur de référence de l'objet est tombé à 0)
ExecutionLevel	IRQL maximal auquel certaines méthodes de l'objet sont exécutées.
ParentObject	Identifie le parent de cet objet.
Size	Taille de l'objet.
SynchronizationScope	Primitives de synchronisation.

Affichage des pilotes KMDF

Pour afficher la liste des pilotes KMDF installés, utilisez la commande !wdfkd.wdfldr.

```
lkd> !wdfkd.wdfldr
``````````````````````````````````````````````````---
 KMDF Drivers
``````````````````````````````````````````````````---
 LoadedModuleList      0xfffff80e903ca238
``````````````````````````--
LIBRARY_MODULE  0xffffb90e3a4888a0
  Version       v1.27
  Service       \Registry\Machine\System\CurrentControlSet\Services\Wdf01000
  ImageName     Wdf01000.sys
  ImageAddress  0xfffff80e902e0000
  ImageSize     0xd1000
  Associated Clients: 22

  ImageName                     Ver   WdfGlobals         FxGlobals          ImageAddress       ImageSize
  monitor.sys                   v1.15 0xffffb90e3fd26b50 0xffffb90e3fd269a0 0xfffff80646030000 0x00016000
  peauth.sys                    v1.15 0xffffb90e3af7e6a0 0xffffb90e3af7e4f0 0xfffff806465c0000 0x000d6000
  winquic.sys                   v1.15 0xffffb90e3f50ff60 0xffffb90e3f50fdb0 0xfffff8064a310000 0x0002b000
  mslldp.sys                    v1.15 0xffffb90e3f4adba0 0xffffb90e3f4ad9f0 0xfffff8064a2d0000 0x0001a000
  UsbHub3.sys                   v1.15 0xffffb90e3af4fb20 0xffffb90e3af4f970 0xfffff806498a0000 0x00093000
  NdisVirtualBus.sys            v1.15 0xffffb90e3af4eeb0 0xffffb90e3af4ed00 0xfffff806497d0000 0x0000d000
  intelppm.sys                  v1.15 0xffffb90e3af44200 0xffffb90e3af44050 0xfffff80649790000 0x0003e000
  vmgencounter.sys              v1.15 0xffffb90e3d3fa1c0 0xffffb90e3d3fa010 0xfffff80649750000 0x0000b000
  ucx01000.sys                  v1.15 0xffffb90e3af801c0 0xffffb90e3af80010 0xfffff80649710000 0x0003f000
  USBXHCI.SYS                   v1.23 0xffffb90e3af121c0 0xffffb90e3af12010 0xfffff80649690000 0x00078000
  HDAudBus.sys                  v1.25 0xffffb90e3d3fb3d0 0xffffb90e3d3fb220 0xfffff80649400000 0x0001f000
  umbus.sys                     v1.15 0xffffb90e3ab0e8e0 0xffffb90e3ab0e730 0xfffff806491c0000 0x00015000
  CompositeBus.sys              v1.15 0xffffb90e3d20bd10 0xffffb90e3d20bb60 0xfffff806490b0000 0x00011000
  Vid.sys                       v1.15 0xffffb90e3ad48ed0 0xffffb90e3ad48d20 0xfffff80649000000 0x00084000
  cdrom.sys                     v1.15 0xffffb90e3d208de0 0xffffb90e3d208c30 0xfffff80e915f0000 0x0002e000
  EhStorClass.sys               v1.15 0xffffb90e3a463f30 0xffffb90e3a463d80 0xfffff80e90d00000 0x0001c000
  vdrvroot.sys                  v1.15 0xffffb90e3af7b870 0xffffb90e3af7b6c0 0xfffff80e906d0000 0x00012000
  msisadrv.sys                  v1.15 0xffffb90e3d347e90 0xffffb90e3d347ce0 0xfffff80e90650000 0x0000b000
  WindowsTrustedRTProxy.sys     v1.15 0xffffb90e3a52aa40 0xffffb90e3a52a890 0xfffff80e90620000 0x0000b000
  WindowsTrustedRT.sys          v1.15 0xffffb90e3a526bf0 0xffffb90e3a526a40 0xfffff80e90600000 0x00016000
  intelpep.sys                  v1.15 0xffffb90e3a521ec0 0xffffb90e3a521d10 0xfffff80e905b0000 0x00043000
  acpiex.sys                    v1.15 0xffffb90e3a4bcdb0 0xffffb90e3a4bcc00 0xfffff80e90410000 0x00024000
``````````````````````````--
Total: 1 library loaded
Wudfrd.sys is not loaded.
Not listing UMDF drivers.
```

## Pilotes signés et non signés

Windows intègre de façon à garantir l'authenticité des pilotes un mécanisme permettant de leur attacher une signature numérique.

Toutes les signatures se rapportant aux pilotes ne sont pas égales.

- Le niveau de confiance le plus élevé est attribué aux pilotes qui sont signés du laboratoire WHQL (Windows Hardware Quality Lab) à l'aide du programme Logo Windows (Windows Logo Program). Ces pilotes peuvent être installés par tout utilisateur sur n'importe quelle plateforme (32 ou 64 bits), ce sans avertissement ou requêtes de consentement. La présence d'une signature issue de Microsoft confirme que le pilote a passé avec succès des tests de compatibilité et de fiabilité extensifs.
- Les pilotes accompagnés d'un certificat fourni par une autorité certifié et émis à des personnes ou des entités explicitement approuvés (Trusted Publishers) peuvent être installés sans aucune invite pour l'utilisateur.
- Si un pilote est signé d'un éditeur dont le certificat ne se trouve pas dans le magasin des éditeurs de confiance (Trusted Publishers), seul un administrateur peut procéder à son installation.
- Les pilotes non signés ou disposant d'une signature invalide ou qui ne peut être vérifiée peuvent être installés par un administrateur sur les versions 32 bits de Windows, mais pas sur des versions 64 bits.

## Magasin de pilotes

Pour faciliter la prise en charge du matériel - autant en ce qui concerne les composants internes (carte mère, contrôleurs de stockage, etc.) que les dispositifs externes (écrans, claviers, souris et autres dispositifs de pointage) - Windows intègre nativement une bibliothèque essentielle de pilotes de périphériques.

Sur des ordinateurs exécutant une version 32 bits de Windows, le lieu de stockage correspondant au magasin de pilotes est \System32\DriverStore. Sur des ordinateurs 64 bits, le magasin de pilotes 64 bits se trouve dans le répertoire \System32\DriverStore et le magasin des pilotes 32 bits dans \System32\SysWOW64\DriverStore. Dans la configuration par défaut, quiconque se connecte à la station de travail peut lire et exécuter des fichiers depuis ces emplacements. Par contre, seul un programme d'installation ayant l'autorisation d'un membre du groupe Administrateurs peut y créer ou modifier des fichiers et dossiers.

Avant Windows Vista, l'installation d'un pilote de périphérique s'effectuait en principe en une phase unique. La chose se déroule désormais en deux étapes : le *staging* et l'installation proprement dite. Le processus de staging consiste à placer les fichiers et composants associés à un pilote dans le magasin de pilotes. Le matériel n'a pas besoin d'être installé à ce stade. Une telle façon de procéder donne en premier lieu à un administrateur la possibilité de déployer un ensemble de pilotes dans le magasin pour prendre en compte les équipements matériels constituant le parc informatique. Cela permet également de garantir la stabilité de ce parc en ne déployant que pilotes testés et vérifiés.

Une fois les fichiers nécessaires à l'installation d'un périphérique incorporés au magasin, il suffit à un utilisateur (dotés ou non de privilèges administratifs) d'insérer le matériel dans une interface idoine ou de le relier à l'ordinateur pour que le matériel soit reconnu et le pilote automatiquement installé.

De nombreuses méthodes viennent au cours de leurs opérations actualiser le stock de pilotes, parmi lesquelles l'assistant d'ajout de matériel, Windows Update et les services WSUS. Chaque pilote acheminé depuis les canaux de distribution officiels est numériquement signé par Microsoft, cela pour prouver son authenticité et, du point de vue du système d'exploitation, attester de sa légitimité.

## User Mode Driver Framework (UMDF)

Compte tenu la sévérité des contraintes inhérentes à la sphère noyau, un nombre croissant de pilotes Windows fonctionnent à un niveau d'exécution moindre (mode utilisateur), cela à l'aide d'une base de développement dédiée : User-Mode Driver Framework (UMDF).

Au niveau des objectifs, UMDF vise une harmonie fonctionnelle totale avec KMDF, y compris en ce qui concerne le modèles objet, de programmation et d'E/S. En pratique, toutefois, les deux cadriciels ne sont pas pas strictement identiques, ni même corrélés, quelques objets reconnus dans l'un ne l'étant pas dans l'autre.

Par comparaison avec KMDF, UMDF offre les avantages que voici :

- Les pilotes UMDF s'exécutent en mode utilisateur, ce qui signifie qu'une erreur sérieuse émanant de l'un d'entre eux (comme une erreur critique) conduit à la fermeture du processus dans lequel le pilote s'exécute, plutôt comme en mode noyau à un effondrement global du système.

- Le processus hôte UMDF se trouve sous l'égide du compte Service local, limité dans ses interactions avec la station de travail (machine locale) et les ordinateurs qui potentiellement entourent celle-ci.

- L'exécution en mode utilisateur signifie que le niveau de requête d'interruptions du processeur est maintenu au niveau passif (`PASSIVE_LEVEL`). De ce fait, le pilote concerné peut honorer les défauts de page et s'appuyer sur des objets noyau pour la synchronisation (événements, mutex, etc.).

Les principaux inconvénients qui entravent la généralisation de UMDF sont : (1) une latence accrue, s'expliquant du fait transitions entre modes processeur ; et (2) l'incompatibilité avec certains types de périphériques hautes performance, dont la prise en charge ne peut être assurée convenablement dans le mode utilisateur.

Contrairement aux pilotes KMDF, qui s'intègrent à l'espace d'adressage du noyau par l'intermédiaire de fichiers SYS, les pilotes UMDF se voient réglementés par un processus hôte (exécutant l'image %SystemRoot %\System32\WUDFHost.exe), similaire à un processus d'hébergement de service. Ledit processus inclut le pilote lui-même, l'infrastructure logicielle sous-jacente (WUDFx02000.dll) et un environnement d'exécution (WUDFPlatform.dll) responsable de plusieurs tâches liées à la gestion du pilote.

Les composants suivants participent de l'architecture UMDF.

- *Réflecteur* Le réflecteur gère la communication entre les composants en mode noyau, tels que les piles de périphériques, et les processus hôtes UMDF, exécutés en mode utilisateur.

- *Processus hôte* Le processus hôte héberge les composants mode utilisateur d'une pile de périphérique. Chaque pile UMDF a son propre processus hôte. Bien qu'il soit exécuté sous l'identité du compte service local, un processus hôte n'est pas en réalité un service Windows, et n'a donc pas de lien avec SCM, mais bel et bien avec le gestionnaire de pilotes.

- *Gestionnaire de pilotes* Le gestionnaire de pilotes est responsable de tous les processus hôtes UMDF exécutés sur le système. Il assure le démarrage et l'arrêt de tels processus, et maintient durant leur cycle de vie un ensemble d'informations d'état. Le gestionnaire de pilotes s'exécute en tant que service Windows standard et est configuré pour un démarrage automatique dès l'installation du premier pilote UMDF sur la station de travail. Une seule instance du gestionnaire de pilotes prend en charge tous les processus hôtes.

- *Pilotes en mode noyau* Les cas pour lesquels une prise en charge spécifique est indiquée peuvent être résolu à l'aide d'un pilote en mode noyau. Il est de la sorte tout à fait de rencontrer un périphérique géré à la fois par un pilote UMDF et par un pilote KMDF (ou même WDM).

## Architecture WDF

Les principaux composants de WDF sont :

- *Applications* Applications mode utilisateur qui, du fait des entrées-sorties auxquelles elles donnent lieu, viennent de la sorte se constituer en tant que clientèle de WDF. En surface, les applications interagissent de la même manière (comprendre utilisent les mêmes fonctions) avec KMDF qu'avec UMDF.

- *Sous-systèmes noyau* Composants de l'exécutif Windows (gestionnaire d'E / S, gestionnaire PnP, etc.) qui prennent part à l'accomplissement d'une requête d'E/S. Ces sous-système s'appuient pour leur communication sur des représentations IRP.

- *Infrastructure logicielle* Intègre deux modèles, l'un concernant les données, l'autre les E/S. Le modèle de données (ou modèle d'objet) régit la manière dont les objets issues de WDF sont créés et gérés. Le modèle d'E/

S détermine comment les requêtes sont contrôlées. A la réception d'un IRP, WDF reconditionne les données au format défini par le modèle et les transmet au pilote WDF pour traitement. Lorsque le pilote a terminé de traiter la requête, celui-ci retourne la demande à WDF, qui complète alors l'IRP et la retourne au gestionnaire d'E/S.

- *Objets WDF* Représentent des constructions de pilote courantes, par exemple un périphérique, un fichier, un tampon DMA, une file File d'attente d'E/S, etc.

- *Evénements* Les objets WDF incorporent des événements dont les pilotes peuvent supplanter le comportement par défaut en fournissent leur propre traitement.

## Comparaison entre pilotes WDM et pilotes WDF

Comparativement aux pilotes WDM, les pilotes WDF se distinguent essentiellement sur deux points :

- Les pilotes WDF nécessitent moins de lignes de code, à la fois globalement et spécifiquement pour ce est qui est de la reconnaissance des périphériques (Plug and Play) et de la gestion de l'alimentation. Moins de code signifie moins de complexité, une meilleure maintenabilité et une fiabilité accrue.

- Les scénarios qui exigent de la synchronisation peuvent avec WDM être difficiles à implémenter de manière optimale. Par contraste, les pilotes WDF ont accès à des primitives de synchronisation intégrées, et bénéficient de la sorte de garanties mieux établies.

- Dans la perspective WDF, c'est l'infrastructure elle-même qui gère la majorité des interactions avec le système.

## Types d'E/S

Dans la perspective du modèle d'E/S Windows, les demandes auxquelles il donne suite sont communément de trois sortes :

- Les *requêtes d'écriture*, qui transmettent des données à pilote afin qu'elles soient acheminées (écrites) sur un périphérique.

- Les *requêtes de lecture*, qui concernent des données qu'un pilote renseigne avant de les retourner à qui les a sollicités.

- Les *requêtes de contrôle*, qui permettent aux logiciels de communiquer entre eux à des fins autres que la lecture ou l'écriture de données. Pour certains périphériques, les demandes de contrôle d'E/S constituent le type de demande le plus courant.

Au delà de ces aspects, les applications Windows émettrices d'E/S se voient conférer de nombreuses possibilités en la matière. Par exemple, elle peuvent spécifier des E/S synchrones ou asynchrones, des E/S gérés en tant que partie de la mémoire virtuelle (dites E/S de fichiers mappés), et enfin des E/S hautes performances conçues pour mener à bien plusieurs opérations en une seule requête (E/S scatter/gather). En plus de cela, le gestionnaire d'E/S offre aux pilotes un mécanisme spécial leur permettant de court-circuiter le schéma traditionnel pour le traitement des E/S (E/S rapides). Dans cette section, nous allons expliquer toutes ces variantes.

### E/S synchrones et E/S asynchrones

La plupart des opérations d'E/S émises par les applications sont synchrones, et exigent dès lors du processus appelant d'attendre que le périphérique transfère les données et renvoie un code d'état quand l'opération est finie. Le programme est alors en mesure de continuer et, selon qu'elles les données aient été correctement prises en charge, y accéder sans plus de délai. Sous leur forme la plus simple, les fonctions Windows ReadFile et WriteFile sont exécutées de façon synchrone, et terminent en l'occurence l'opération d'E/S avant de rendre le contrôle à l'appelant.

Les E/S asynchrones ont pour avantage de désynchroniser les opérations d'entrées/sorties logiques des opérations d'entrées/sorties physiques, permettant ainsi à une application d'émettre une requête d'E/S, puis de poursuivre son exécution tandis que le périphérique transfère les données. Afin de tirer partie des E/S asynchrones, l'appelant doit spécifier l'utilisation dudit mécanisme par le biais du flag FILE_FLAG_OVERLAPPED lors de l'appel à la fonction CreateFile. Après avoir émis une opération d'E/S asynchrone, le thread doit veiller à ne pas accéder aux données concernées par l'E/S tant que le périphérique n'a pas terminé de les transférer. Qui plus est, compte tenu de la temporalité imprévisible des échanges, le thread doit synchroniser son exécution avec l'achèvement de la requête d'E/S, en surveillant un handle d'un objet de synchronisation (qu'il s'agisse d'un objet événement, d'un port de complétude d'E/S ou de l'objet fichier lui-même) que le système basculera à l'état signalé une fois l'E/S accomplie.

Indépendamment du type de requête d'E/S, en interne les opérations d'E/S envoyées à un pilote pour le compte de application s'effectuent de manière asynchrone. Cela signifie que, une fois qu'une requête d'E/S a été initiée, le pilote rend la main au système d'E/S. Le fait que le système d'E/S redonne ou non immédiatement le contrôle à l'appelant dépend de ce que le fichier a été ouvert pour des E/S synchrones ou asynchrones. Il est possible de vérifier le statut d'une E/S asynchrone en attente en instance par le biais de la fonction HasOverlappedIoCompleted. Dans l'éventualité où c'est un port de complétude qui fait office de méthode de synchronisation, employez alors la fonction GetQueuedCompletionStatus.

### E/S scatter/gather

En plus des quelques solutions d'E/S que que nous avons vu jusqu'à présent, Windows fournit enfin un type spécial d'E/S hautes performances appelé disperser/rassembler (scatter/gather), accessible depuis les fonctions Windows ReadFileScatter et WriteFileScatter. Ces fonctions permettent à une application de mobiliser via une opération unique différentes régions mémoire, et ce faisant émettre une même lecture ou écriture depuis plusieurs tampons en mémoire virtuelle vers une portion connexe d'un fichier disque.

## Vérification des E/S

Comme d'autres composants fondamentaux de Windows, le gestionnaire d'E/S honore diverses options permettant de vérifier la justesse des opérations sur lesquelles il a la main mise.

*Vérification d'E/S* Piste l'utilisation des interfaces fondamentales via lesquelles les pilotes s'intègrent au système d'E/S. Exemples d'infractions sévèrement sanctionnées : achèvement d'un IRP (IoCompleteRequest) contenant un statut invalide, objet de périphérique invalide lors de sa soumission au gestionnaire d'E/S (IoCallDriver), initialisation d'une minuterie déjà initialisée (IoInitializeTimer), libération d'un IRP (IoFreeIrp) resté associé à un thread, exécution d'une routine d'E/S tandis que l'IRQL du processus est élevé. Tout échec de vérification d'E/S de niveau 1 entraine un effondrement du système avec le code d'arrêt DRIVER_VERIFIER_IOMANAGER_VIOLATION.

*Vérification d'E/S de niveau 2* Assure un suivi rigoureux concernant les opérations de finalisation d'IRP et l'utilisation de la pile IRP.

*Contrôle DMA* Instaure différents contrôles portant sur l'utilisation correcte des des fonctions et sur les tampons que le gestionnaire d'E/S fournit en ce qui concerne les opérations DMA.

*Vérification d'intégrité de disque* Surveille les opérations de lecture et écriture disque et calcule des sommes de contrôle pour les données concernées. Dans l'éventualité où la valeur la plus récente ne concorde pas avec celle enregistrée préalablement, suggérant une altération matérielle du disque, Vérificateur de pilote fait s'effondrer le système avec le code d'arrêt DRIVER_VERIFIER_DETECTED_VIOLATION (0x000000C4).

*Vérification SCSI* Contrôle l'utilisation que fait un pilote de miniport SCSI des routines fournies par le pilote de port. Si le pilote utilise injustement une fonction, ne répond pas convenablement à une demande ou met trop de temps pour ce faire, une erreur d'arrêt a lieu (SCSI_VERIFIER_DETECTED_VIOLATION). (Notez que cette option ne fait pas partie de celle affichée par la boite de dialogue du vérificateur. Elle est cependant activée de facto quand un pilote de miniport SCSI est sélectionné pour la vérification.)

## Ports d'achèvement d'E/S

Quand une application crée un port d'achèvement, elle le fait en spécifiant une valeur indiquant le nombre maximal de threads autorisés simultanément. Dans l'éventualité où le nombre de threads actifs associées à un port atteint la valeur fixée, Windows force alors les threads surnuméraires à entrer dans un état d'attente.

Le tableau qui suit répertorie les opérations envisageables concernant les ports d'achèvement.

***Tableau 14.14. Opérations concernant les ports d'achèvement.***

| Opération | Fonction | Service | Routine |
|---|---|---|---|
| Créer un port d'achèvement | CreateIoCompletionPort | NtCreateIoCompletion | ObCreateObject(Type = IoCompletionType) |
| | GetQueuedCompletionStatus | NtRemoveIoCompletion | KeRemoveQueue |
| Met un file, pour un port d'achèvement, une notification de fin d'E/S | PostQueuedCompletionStatus | NtSetIoCompletion | KeInsertQueue |

## Examen du répertoire \Driver

Dès lors qu'un pilote a mené a bien son chargement, le gestionnaire d'E/S insère l'objet idoine dans l'espace de noms global sous le répertoire \Driver. (Corolaire de ce qui précède, un pilote dont le nom est absent du répertoire susmentionné n'a pas été chargé.) Si vous exécutez la commande !objet du débogueur noyau et spécifiez le répertoire \Driver, vous devriez voir un résultat comme celui-ci :

```
lkd> !object \Driver
Object: ffffbf06e3416260  Type: (ffffd10a9fa9d640) Directory
    ObjectHeader: ffffbf06e3416230 (new version)
    HandleCount: 0  PointerCount: 128
    Directory Object: ffffbf06e3009100  Name: Driver

    Hash Address          Type                    Name
    ---- -------          ----                    ----
     00  ffffd10aabc4be30 Driver                  kldbgdrv
         ffffd10aa290ed10 Driver                  fvevol
         ffffd10a9fb4c6d0 Driver                  vdrvroot
     01  ffffd10aad2b3e20 Driver                  PptpMiniport
         ffffd10aa40748b0 Driver                  usbuhci
         ffffd10aa4010370 Driver                  GpuEnergyDrv
         ffffd10aa2178ca0 Driver                  NetBT
         ffffd10a9fadcc50 Driver                  acpiex
         ffffd10a9fadea00 Driver                  Wdf01000
     02  ffffd10aad231820 Driver                  usbprint
         ffffd10aad63a2b0 Driver                  WdNisDrv
         ffffd10aad555300 Driver                  mpsdrv
         .
         .
         .
```

## Examen du répertoire \Device

Pour voir les périphériques que le gestionnaire d'E/S a enraciné dans l'espace de noms du gestionnaire d'objets, exécutez la commande !objet du débogueur noyau en spécifiant le répertoire \Device. Vous devriez alors voir quelque chose qui ressemble à ceci :

```
lkd> !object \Device
Object: ffffbf06e300d250  Type: (ffffd10a9fa9d640) Directory
    ObjectHeader: ffffbf06e300d220 (new version)
    HandleCount: 2  PointerCount: 65791
    Directory Object: ffffbf06e3009100  Name: Device
```

```
Hash Address          Type                   Name
---- -------          ----                   ----
 00  ffffd10aa96c8e00 Device                 00000058
     ffffd10aad2d67c0 Device                 kldbgdrv
     ffffd10aad2af4e0 Device                 00000044
     ffffd10aaa781050 Device                 NDMP2
     ffffd10aa2c09060 Device                 00000030
     ffffd10aa21dd060 Device                 NTPNP_PCI0002
 01  ffffd10aa93d2060 Device                 00000054
     ffffbf06eb0ad5f0 SymbolicLink           {60b1ba98-c528-4e00-b24b-c778b71842d2}
     ffffd10aa950fa60 Device                 00000040
     ffffd10aa96b8050 Device                 NDMP3
     ffffd10aa2be5710 Device                 gpuenergydrv
     ffffd10aa21de060 Device                 NTPNP_PCI0003
 02  ffffd10aad340900 Device                 wdnisdrv
     ffffd10aa49220a0 Device                 USBPDO-5
     ffffd10aa41ae060 Device                 00000050
     ffffd10aa96cb050 Device                 NDMP4
     .
     .
     .
```

## Déchargement de pilote

À divers stades de l'exécution de Windows, il peut s'avérer indispensable de retirer un pilote de la mémoire, que ce soit dans le cadre d'une mise à jour, d'une désinstallation, ou pour toute autre raison émanant de la gestion dynamique des modules du noyau. Dans une telle éventualité, les pilotes peuvent - et idéalement doivent - définir auprès du gestionnaire d'E/S une routine spéciale, dite de déchargement, prévue pour rétablir la configuration du système telle qu'elle était avant leur introduction.

Jouant un rôle fondamental dans la préservation de la fiabilité et de la robustesse du système, la routine de déchargement de tel ou tel pilote est pour lui l'ultime occasion de démanteler les ressources mobilisées au cours de son fonctionnement : désenregistrer les gestionnaires d'événements, libérer la mémoire allouée dynamiquement, supprimer les dispositifs symboliques créés, etc. À défaut, des fuites de ressources - qu'il s'agisse de mémoire, de handles ou d'objets noyau, persisteraient au-delà du cycle de vie du pilote, nuisant de la sorte aux performances, voire à la sécurité, de la plateforme.

Notez, à titre d'avertissement, que le système ne transfère le contrôle à la routine de déchargement que si la mise en route du pilote s'est conclue avec succès. Par conséquent, toute ressource allouée durant cette phase doit, en cas d'échec, être libérée immédiatement dans la fonction d'initialisation (`DriverEntry`), et non reportée à la routine de déchargement.

***Tableau 14.15. Déchargement de pilote***

| Élément | Lieu |
|---|---|
| DriverUnload | DRIVER_OBJECT |

## Annulation d'E/S

Certaines opérations d'E/S, du fait des circonstances qui les entourent (à proprement parler dans ce contexte, qui les entravent), peuvent n'être plus en mesure d'aller de l'avant : retrait d'un périphérique tandis que des IRP le concernant sont encore en cours, interruption par l'utilisateur d'une tâche considérée anormalement longue (par exemple un transfert réseau), fin prématurée d'un processus ou d'un thread, et bien d'autres exemples. Pour éviter que le système ne reste alors bloqué indéfiniment, le gestionnaire d'E/S permet aux pilotes d'enregistrer une routine spéciale pour les opérations susceptibles d'être annulés (typiquement celles encore en file d'attente et dont le traitement n'a pas encore débuté). Lorsque l'utilisateur ou le système tente d'annuler une opération, le gestionnaire d'E/S donne alors la main à cette routine, définie à l'aide de la fonction `IoSetCancelRoutine`, offrant ainsi au pilote concerné la possibilité de nettoyer la requête. Gérer correctement l'annulation d'E/S en instance

est une capacité essentielle autant pour la fiabilité que la réactivité du système. À défaut, cela peut mener à des situations anormales, par exemple un processus impossible à terminer : l'application disparait de l'écran mais reste listée dans le Gestionnaire des tâches (ou tout autre utilitaire du même style), une opération d'E/S en attente empêchant concrètement d'y mettre fin.

En matière d'annulation d'E/S, l'API Windows expose trois interfaces. La première, `CancelSynchronousIo`, annule toutes les opérations d'E/S synchrones en attente sur un thread, indépendamment de la nature de l'objet attendu. La seconde, `CancelIo`, offre la possibilité d'interrompre les E/S asynchrones en attente sur un handle (tel qu'un fichier), pourvu que celles-ci aient été émises par le thread appelant (les opérations qui émanent d'autres threads ne sont autrement dit pas impactées). Enfin, `CancelIoEx`, plus flexible, permet d'annuler toutes les E/S asynchrones sur ce même handle tant qu'elles proviennent du même processus.

# Chapitre 15. Gestion de l'alimentation

## Gestionnaire d'alimentation

Composant majeur de l'exécutif Windows, le gestionnaire d'alimentation a pour rôle la coordination des événements électriques, fonction qu'il assure en émettant des notifications d'E/S de gestion électrique à destination des périphériques compatibles.

Le gestionnaire d'alimentation décide quand opérer une transition d'état électrique en fonction de plusieurs facteurs :

- Niveau d'activité du système
- Niveau de la batterie
- Sollicitations d'arrêt ou de mise en veille émanant des applications
- Actions de l'utilisateur, par exemple l'appui sur un bouton d'alimentation
- Options d'alimentation du panneau de configuration

**Visualisation des capacités d'alimentation du système.** L'extension !pocaps du débogueur noyau permet de visualiser les capacités d'alimentation de l'ordinateur. Voici un exemple de sortie de la commande :

```
lkd> !pocaps
PopCapabilities @ 0xfffff8033d600360
  Misc Supported Features:  PwrButton S1 S5 FullWake
  Processor Features:
  Disk Features:
  Battery Features:         BatteriesPresent
    Battery 0 - Capacity:        0  Granularity:        0
    Battery 1 - Capacity:        0  Granularity:        0
    Battery 2 - Capacity:        0  Granularity:        0
  Wake Caps
    Ac OnLine Wake:         Sx
    Soft Lid Wake:          Sx
    RTC Wake:               S1
    Min Device Wake:        Sx
    Default Wake:           Sx
```

*Tableau 15.1. Structures de données concernant l'alimentation*

| Structure | Visibilité |
|---|---|
| EXECUTION_STATE | NtSetThreadExecutionState |
| ADMINISTRATOR_POWER_POLICY | NtPowerInformation |
| POWER_PLATFORM_INFORMATION | NtPowerInformation |
| PROCESSOR_POWER_INFORMATION | NtPowerInformation |
| POWER_STATE_HANDLER | NtPowerInformation |
| PROCESSOR_STATE_HANDLER | NtPowerInformation |
| SYSTEM_BATTERY_STATE | NtPowerInformation |
| SYSTEM_POWER_CAPABILITIES | NtPowerInformation |
| SYSTEM_POWER_INFORMATION | NtPowerInformation |
| SYSTEM_POWER_POLICY | NtPowerInformation |
| SYSTEM_POWER_STATUS | GetSystemPowerStatus |

*Tableau 15.2. Énumérations concernant l'alimentation*

| Énumération | Visibilité |
|---|---|
| DEVICE_POWER_STATE | |
| LATENCY_TIME | RequestWakeupLatency |

| Énumération | Visibilité |
|---|---|
| POWER_ACTION | NtInitiatePowerAction |
| POWER_INFORMATION_LEVEL | CallNtPowerInformation, NtPowerInformation |
| POWER_REQUEST_TYPE | PowerClearRequest |
| POWER_STATE_HANDLER_TYPE | POWER_STATE_HANDLER |
| POWER_STATE_TYPE | PoSetPowerState |
| SYSTEM_POWER_STATE | NtInitiatePowerAction |

*Tableau 15.3. Interfaces concernant l'alimentation*

| Fonction | Service |
|---|---|
| CancelDeviceWakeupRequest | NtCancelDeviceWakeupRequest |
| GetPwrCapabilities | NtPowerInformation (SystemPowerCapabilities) |
| GetSystemPowerStatus | NtPowerInformation |
| GetDevicePowerState | NtGetDevicePowerState |
| | NtInitiatePowerAction |
| IsPwrShutdownAllowed | NtPowerInformation (SystemPowerCapabilities) |
| IsPwrSuspendAllowed | NtPowerInformation (SystemPowerCapabilities) |
| IsSystemResumeAutomatic | NtIsSystemResumeAutomatic |
| RequestDeviceWakeup | NtRequestDeviceWakeup |
| | NtPowerInformation |
| RequestWakeupLatency | NtRequestWakeupLatency |
| SetThreadExecutionState | NtSetThreadExecutionState |
| SetSystemPowerState | NtSetSystemPowerState |

*Tableau 15.4. POWER_INFORMATION_LEVEL*

| ID | Classe | Type de données associé |
|---|---|---|
| 0x0000 | SystemPowerPolicyAc | SYSTEM_POWER_POLICY |
| 0x0001 | SystemPowerPolicyDc | SYSTEM_POWER_POLICY |
| 0x0002 | VerifySystemPolicyAc | |
| 0x0003 | VerifySystemPolicyDc | |
| 0x0004 | SystemPowerCapabilities | SYSTEM_POWER_CAPABILITIES |
| 0x0005 | SystemBatteryState | SYSTEM_BATTERY_STATE |
| 0x0006 | SystemPowerStateHandler | |
| 0x0007 | ProcessorStateHandler | |
| 0x0008 | SystemPowerPolicyCurrent | SYSTEM_POWER_POLICY |
| 0x0009 | AdministratorPowerPolicy | |
| 0x000A | SystemReserveHiberFile | |
| 0x000B | ProcessorInformation | PROCESSOR_POWER_INFORMATION |
| 0x000C | SystemPowerInformation | SYSTEM_POWER_INFORMATION |
| 0x000D | ProcessorStateHandler2 | |
| 0x000E | LastWakeTime | |
| 0x000F | LastSleepTime | |
| 0x0010 | SystemExecutionState | |
| 0x0011 | SystemPowerStateNotifyHandler | |
| 0x0012 | ProcessorPowerPolicyAc | |
| 0x0013 | ProcessorPowerPolicyDc | |
| 0x0014 | VerifyProcessorPowerPolicyAc | |
| 0x0015 | VerifyProcessorPowerPolicyDc | |
| 0x0016 | ProcessorPowerPolicyCurrent | |

| ID | Classe | Type de données associé |
|---|---|---|
| 0x0017 | SystemPowerStateLogging | |
| 0x0018 | SystemPowerLoggingEntry | |
| 0x0019 | SetPowerSettingValue | |
| 0x001A | NotifyUserPowerSetting | |
| 0x001B | PowerInformationLevelUnused0 | |
| 0x001C | SystemMonitorHiberBootPowerOff | |
| 0x001D | SystemVideoState | |
| 0x001E | TraceApplicationPowerMessage | |
| 0x001F | TraceApplicationPowerMessageEnd | |
| 0x0020 | ProcessorPerfStates | |
| 0x0021 | ProcessorIdleStates | |
| 0x0022 | ProcessorCap | |
| 0x0023 | SystemWakeSource | |
| 0x0024 | SystemHiberFileInformation | |
| 0x0025 | TraceServicePowerMessage | |
| 0x0026 | ProcessorLoad | |
| 0x0027 | PowerShutdownNotification | |
| 0x0028 | MonitorCapabilities | |
| 0x0029 | SessionPowerInit | |
| 0x002A | SessionDisplayState | |
| 0x002B | PowerRequestCreate | |
| 0x002C | PowerRequestAction | |
| 0x002D | GetPowerRequestList | |
| 0x002F | ProcessorInformationEx | |
| 0x0030 | NotifyUserModeLegacyPowerEvent | |
| 0x0031 | GroupPark | |
| 0x0032 | ProcessorIdleDomains | |
| 0x0033 | WakeTimerList | |
| 0x0034 | SystemHiberFileSize | |
| 0x0035 | ProcessorIdleStatesHv | |
| 0x0036 | ProcessorPerfStatesHv | |
| 0x0037 | ProcessorPerfCapHv | |
| 0x0038 | ProcessorSetIdle | |
| 0x0039 | LogicalProcessorIdling | |
| 0x003A | UserPresence | |
| 0x003B | PowerSettingNotificationName | |
| 0x003C | GetPowerSettingValue | |
| 0x003D | IdleResiliency | |
| 0x003E | SessionRITState | |
| 0x003F | SessionConnectNotification | |
| 0x0040 | SessionPowerCleanup | |
| 0x0041 | SessionLockState | |
| 0x0042 | SystemHiberbootState | |
| 0x0043 | PlatformInformation | |
| 0x0044 | PdcInvocation | |
| 0x0045 | MonitorInvocation | |
| 0x0046 | FirmwareTableInformationRegistered | |

| ID | Classe | Type de données associé |
|---|---|---|
| 0x0047 | SetShutdownSelectedTime | |
| 0x0048 | SuspendResumeInvocation | |
| 0x0049 | PlmPowerRequestCreate | |
| 0x004A | ScreenOff | |
| 0x004B | CsDeviceNotification | |
| 0x004C | PlatformRole | |
| 0x004D | LastResumePerformance | |
| 0x004E | DisplayBurst | |
| 0x004F | ExitLatencySamplingPercentage | |
| 0x0050 | RegisterSpmPowerSettings | |
| 0x0051 | PlatformIdleStates | |
| 0x0052 | ProcessorIdleVeto | |
| 0x0053 | PlatformIdleVeto | |
| 0x0054 | SystemBatteryStatePrecise | |
| 0x0055 | ThermalEvent | |
| 0x0056 | PowerRequestActionInternal | |
| 0x0057 | BatteryDeviceState | |
| 0x0058 | PowerInformationInternal | |
| 0x0059 | ThermalStandby | |
| 0x005A | SystemHiberFileType | |
| 0x005B | PhysicalPowerButtonPress | |
| 0x005C | QueryPotentialDripsConstraint | |
| 0x005D | EnergyTrackerCreate | |
| 0x005E | EnergyTrackerQuery | |
| 0x005F | UpdateBlackBoxRecorder | |

## Schémas d'alimentation

Windows centralise les grands arbitrages énergétiques au sein d'ensembles opérationnels dits schémas (ou plans, ou encore modes de gestion) d'alimentation. Un mode de gestion de l'alimentation est une combinaison de paramètres matériels et système (tels que la luminosité de l'écran, le mode veille, etc.) qui définit comment l'ordinateur utilise l'énergie.

Windows intègre nativement trois modes de gestion de l'alimentation :

- *Utilisation normale* Mode par défaut, qui équilibre la consommation énergétique par rapport aux performances du système.
- *Économie d'énergie* Économise l'énergie en réduisant les performances système.
- *Performances élevées* Privilégie les performances au au détriment de l'économie d'énergie.

Pour voir concrètement les paramètres qu'englobe un plan d'alimentation, dans la fenêtre Options d'alimentation, cliquez sur Modifier les paramètres du plan puis sélectionnez Modifier les paramètres d'alimentation avancés. La liste déroulante située en haut de la fenêtre, qui offre la possibilité de basculer entre les différents modes, permet ainsi de déterminer rapidement dans quelle mesure l'un diffère d'un autre,

**Création de modes de gestion de l'alimentation.** Pour créer un schéma d'alimentation, procédez comme suit :

1. Ouvrez la fenêtre Options d'alimentation, par exemple via la catégorie Système et sécurité du Panneau de configuration.

2. Dans le volet gauche de la fenêtre Options d'alimentation, cliquez sur Créer un mode de gestion de l'alimentation.

3. Dans la page créer un plan d'alimentation, sélectionnez le mode dont les caractéristiques correspondent le mieux à vos attentes (les paramètres dudit mode serviront de base à celui auquel vous donnerez suite).

4. Dans la zone Nom du mode, entrez un nom pour le mode, puis cliquez sur suivant. Apparait alors la page Modifier les paramètres du mode.

5. Utilisez les listes Éteindre l'écran et Mettre l'ordinateur en veille pour spécifier, respectivement, quand éteindre l'écran et quand mettre l'ordinateur en veille en cas d'inactivité. Dans les deux cas, optez pour l'option Jamais en vue de désactiver la fonctionnalité.

6. Cliquez sur Créer pour finaliser la création du mode, ce qui a pour effet la réapparition de la page Choisir ou personnaliser un mode de l'alimentation, qui affiche désormais le mode que vous venez de créer dans la liste des favoris. Le mode originel est quant à lui relégué dans la zone Modes supplémentaires.

**Gestion des schémas d'alimentation depuis la ligne de commandes.** Windows intègre pour administrer les modes de gestion de l'alimentation depuis la ligne de commandes l'utilitaire Powercfg.exe, dont voici les options les plus communément rencontrées.

- *-a* Indique les états de veille disponibles sur le système
- *-d [guid]()* Supprime le mode de gestion de l'alimentation qui porte le GUID spécifié
- *-h* Active ou désactive la fonction de veille prolongée
- *-l* Répertorie les modes de gestion définis sur la machine*
- *-q [guid]()* Affiche le contenu du mode de gestion de l'alimentation qui porte le GUID spécifié
- *-s [guid]()* Fait du mode ayant le GUID spécifié le mode actif
- *-setabsentia [guid]()* Fait du mode ayant le GUID spécifié le mode employé quand il n'y a pas de session ouverte Si quelques commandes reconnues par Powercfg fonctionnent à partir d'une session d'utilisateur standard, certaines opération, tels que la création ou modification de modes, exigent des droits d'administrateur.

Pour obtenir la liste des schémas d'alimentation définis sur la machine, spécifiez le commutateur -l.

```
C:\>powercfg -l

Modes de gestion de l'alimentation existants (* Actif)
````````````````````````---
GUID du mode de gestion de l'alimentation : 381b4222-f694-41f0-9685-ff5bb260df2e  (Utilisation normale) *
GUID du mode de gestion de l'alimentation : 8c5e7fda-e8bf-4a96-9a85-a6e23a8c635c  (Performances élevées)
GUID du mode de gestion de l'alimentation : a1841308-3541-4fab-bc81-f71556f20b4a  (Économie d'énergie)

Les schémas d'alimentation existants sont répertoriés par nom et par GUID. Le mode actif est mis en évidence via une
 astérisque. Le listing qui précède montre que la machine a trois modes de gestion et que le mode actif est TODO. Pour
 déterminer rapidement le mode d'alimentation actif, utilisez Powercfg en spécifiant le commutateur -getactivescheme.
```

Pour obtenir des informations détaillées concernant un mode d'alimentation en particulier, spécifiez le commutateur -q suivi du GUID du mode. Les représentations GUID des paramètres d'alimentation n'étant certainement pas pas chose facile à mémoriser, Powercfg reconnait des alias pour les GUID les plus utilisés. Par exemple, sachant que l'alias se rapportant au mode Utilisation normale est SCHEME_BALANCED, il est possible pour définir le mode actif d'employer la commande powercfg -setactive SCHEME_BALANCED. Pour voir la liste des alias définis, tapez powercfg aliases.

```
C:\>*powercfg aliases*
a1841308-3541-4fab-bc81-f71556f20b4a  SCHEME_MAX
```
```

```
8c5e7fda-e8bf-4a96-9a85-a6e23a8c635c  SCHEME_MIN
381b4222-f694-41f0-9685-ff5bb260df2e  SCHEME_BALANCED
e73a048d-bf27-4f12-9731-8b2076e8891f  SUB_BATTERY
637ea02f-bbcb-4015-8e2c-a1c7b9c0b546    BATACTIONCRIT
d8742dcb-3e6a-4b3c-b3fe-374623cdcf06    BATACTIONLOW
                                        .
                                        .
                                        .
```

# Chapitre 16. Gestion du stockage

## Terminologie du stockage

- Les *disques* correspondent à des périphériques de stockage physiques tels que disque dur, clé USB ou CD-ROM.
- Les *volumes simples* s'apparentent aux secteurs d'une même partition. Par nature, chaque volume de la sorte est contenu entièrement dans un seul périphérique physique.
- Les *volumes multi partition* sont des objets qui représentent des secteurs provenant de plusieurs partitions situées sur un même disque.
- Les *volumes fractionnés* combinent de l'espace provenant de disques physiques distincts.
- Windows met en oeuvre pour définir des volumes sur les disques deux schémas de partitionnement : MBR et GPT. Par comparaison avec les disques MBR, limités à 4 partitions de 2,2 To chacune, les disques GPT prennent en charge de plus grands volumes (jusqu'à 18 exaoctets) et plus de partitions (jusqu'à 128 sur un disque de base).

## Types de stockage disque

Windows offre deux types de stockage sur disque : les disques de base, qui renferment des partitions, et les disques dynamiques, qui contiennent non des partitions mais des volumes. Le type de stockage d'un disque dur détermine l'utilisation de son espace. Lorsque vous configurez le système de stockage de l'ordinateur, vous devez choisir un type de stockage pour chaque disque.

- *Stockage de base* Un disque configuré pour le stockage de base, ou disque de base, peut contenir des partitions principales et des partitions étendues avec des lecteurs logiques. Les nouveaux disques que vous ajoutez à un ordinateur exécutant Windows appartiennent par défaut à cette catégorie. Le stockage de base garantit une compatibilité avec les partitions de disque et les jeux de volume manipulées sous toutes les versions de Microsoft Windows.
- *Stockage dynamique* Un disque converti du format de base au format dynamique peut contenir des volumes simples, des volumes fractionnés, des volumes en miroir (RAID 1), des volumes agrégés par bandes (RAID 0) et des volumes agrégés par bandes avec parité (RAID-5).

La différence majeure entre ces deux technologies est que l'une divise le disque en un nombre limité de partitions et de disques logiques (stockage de base), tandis que l'autre divise les disques en un nombre illimité de volumes (stockage dynamique).

Les types de stockage sont mutuellement exclusifs. Il n'est autrement dit pas possible de combiner les deux types sur un même disque. Le raisonnement corollaire inverse implique qu'il est néanmoins tout à fait possible de combiner les types de stockage sur disque dans un système comportant plusieurs disques.

## Disques de base

Par défaut, Windows gère tous les disques comme des disques de base, le gestionnaire de volumes (pilote FtDisk, sur lequel nous reviendrons plus loin) les configurant en tant que tel dès leur prise en compte parmi le système de stockage de l'ordinateur.

Les disques de base imposent une limite dans le nombre de partitions qu'ils sont capables d'accueillir. Chaque disque utilisant le stockage de base peut ainsi être constitué d'au maximum quatre partitions principales, ou de trois partitions principales et d'une partition étendue, qui elle peut inclure jusqu'à 24 disques logiques.

## Etats des disques et des volumes

Gestion des disques permet de connaitre l'état et les propriétés des disques et des volumes qu'ils contiennent. Le statut du disque apparait sous le numéro de disque dans la vue Représentation graphique, et dans le champ Statut dans la vue Liste des disques. Le statut du volume apparait parmi les informations de volume dans la vue Représentation graphique, et dans la colonne Statut de la vue Liste des volumes. Dans des circonstances normales, toutes les informations définies relativement à l'état des disques et des volumes sur le système devraient indiquer que chaque disque est En ligne et que chaque volume est sain.

La liste ci-après donne une brève description des messages de statut possibles pour les disques, ainsi que quelques pistes à suivre en cas de problème.

- *En ligne* Le disque est configuré correctement et ne présente pas de problèmes connus. Ceci est l'état normal du périphérique, qu'il s'agisse d'un disque de base ou d'un disque dynamique.

- *En ligne (Erreurs)* Le système d'exploitation a rencontré des erreurs en lisant ou en écrivant des données d'une région d'un disque dynamique. Cliquez avec le bouton droit sur le disque et choisissez Réactiver le disque pour retourner à son statut En ligne. Si des erreurs persistent, c'est que le disque est peut-être endommagé ou qu'il requiert un examen approfondi.

- *Etranger* Le disque provient d'un autre système que celui pour lequel il a été configuré. Pour résoudre ce problème, effectuez un clic droit sur le disque puis sélectionnez Importer un disque étranger.

- *Hors ligne* Indique que le disque n'est pas accessible. Ce code, qui n'a de sens qu'avec des disques dynamiques, peut survenir si le système de fichiers est corrompu, ou pointer du doigt une panne matérielle.

- *Aucun média* Aucun support n'est inséré dans le lecteur. Seuls les lecteurs de disques amovibles, tels que les lecteurs de CD et de DVD, affichent ce statut.

Les états suivants concernent les volumes.

- *Echec* Le système d'exploitation se trouve dans l'impossibilité de statuer concernant l'état du volume. L'un des disques sous-jacents est peut-être manquant, endommagé, ou avoir des problèmes de connectivité.

- *Formatage* Le volume est en cours de formatage. Un pourcentage de réalisation est affiché, hormis si l'option choisie à cet égard consistait en un formatage rapide.

- *Inconnu* Le secteur d'amorce du disque physique ne peut être lu, prévenant toute possibilité de consulter les volumes qu'il héberge. Cette situation peut survenir si le secteur d'amorce est endommagé ou si aucune signature de disque n'a été créée.

- *Sain* Le volume a été correctement formaté et n'a pas de problème connu.

- *Sain (Risque)* Windows a rencontré des erreurs lors de la lecture ou de l'écriture du disque physique contenant ce volume dynamique.

- *Sain (Partition inconnue)* Windows ne reconnaît pas la partition. Le scénario le plus probable en ces circonstances est que la partition a été créé par un autre système exploitation selon des modalités non définies.

## Administration des disques

Windows offre plusieurs outils en vue de manipuler les disques dont est équipé la station de travail : la console Ordinateur, l'utilitaire Gestion des disques, ainsi que les commandes (d'ailleurs relativement méconnues) Fsutil et Diskpart.

## Utilitaire Gestion des disques

L'outil sans conteste le plus usité pour gérer les disques sur des ordinateurs locaux ou distants sous Windows est à l'ordinaire Gestion des disques, qui offre en la matière tout le nécessaire pour administrer périphériques de stockage, partitions, volumes, lecteurs logiques et systèmes de fichiers associés.

Se présentant sous la forme d'un composant logiciel enfichable - à ce titre apparenté à MMC, Gestion des disques est accessible soit depuis une console pré configurée, par exemple Gestion de l'ordinateur, soit par une console personnalisée à laquelle vous l'avez incorporé.

Plusieurs méthodes permettent l'accès à la console de gestion des disques, choisissez l'une ou l'autre d'entre elles : (1) sélectionnez Gestion des disques dans le volet gauche de la fenêtre Gestion de l'ordinateur ; (2) depuis une invite de commandes ou la boîte de dialogue Exécuter, saisissez diskmgmt.msc.

L'affichage au sein de la console Gestion des disques est réparti en deux volets, disposés de part et d'autre d'une ligne modulable horizontale. Dans la configuration par défaut, le volet supérieur liste les différents volumes intégrés au système, et fournit pour chacun des informations sur le type (de base ou dynamique), le statut, l'espace libre disponible, la tolérance aux éventuelles pannes, etc. Pour exécuter des commandes sur un volume, faites un clic droit sur la première colonne de ce volet (colonne nommée Volume) et choisissez l'option appropriée dans le menu contextuel.

Le volet du bas procure une vue graphique de la structure organisationnelle qui régit chaque périphérique de stockage physique. Dans les titres du coté gauche, sur chaque entrée (chaque ligne), figure le numéro que Windows a automatiquement associé au périphérique (Disque 0, Disque 1, et ainsi de suite), de même que le type, la taille et l'état du périphérique. À droite de ces éléments se trouvent des rectangles symbolisant les volumes définis sur le disque. Un clic droit sur l'intitulé d'un disque (rectangle blanc) affiche une liste d'actions pertinentes pour tout le périphérique. Un clic droit sur le rectangle correspondant à un volume fait apparaitre un panel d'options en rapport avec ledit volume.

La manière dont la Gestion des disque affiche les informations peut être modifiée à l'aide du menu Affichage. Par exemple, pour personnaliser les informations montrées dans le volet supérieur ou inférieur de la fenêtre, cliquez sur Haut ou sur Bas dans le menu Affichage. Trois choix sont alors offerts : Liste des volumes, Liste des disques et Représentation graphique .

### Mise à jour des informations de l'outil Gestion des disques

S'il est en principe toujours au fait des dernières modifications apportées, l'outil Gestion des disques peut ne pas refléter l'état réel du système de stockage de l'ordinateur. Utilisez dans cette éventualité les commandes Actualiser et Analyser les disques de nouveau du menu Action pour mettre à jour les informations appropriées en la matière.

- La commande Actualiser met à jour les informations relatives à la lettre du lecteur, au système de fichiers, au volume et aux supports amovibles.
- La commande Analyser les disques de nouveau met à jour les informations relatives au matériel. Lorsque l'outil Gestion des disques analyse de nouveau les disques, il considère tous les disques concernés par les modifications de la configuration. Il met également à jour les informations relatives aux supports amovibles, aux lecteurs optiques, aux partitions, aux volumes, aux systèmes de fichiers et aux lettres de lecteur.

## Console Ordinateur

La fenêtre ou console Ordinateur offre une rampe d'accès vers les divers périphériques de stockage reconnues par Windows, incluant les lecteurs de disques, les disques amovibles et les éventuels emplacements réseau.

- *Périphériques et lecteurs* Répertorie les unités de stockage physiquement raccordés à la machine (disques locaux). Pour afficher le contenu d'une unité, activez son nom dans le volet de navigation ou effectuez un

double clic sur son nom dans la zone d'affichage. Cliquez avec le bouton droit de la souris pour afficher les commandes d'administration.

- *Emplacement réseau* Énumère les lecteurs réseau. Un lecteur réseau donne accès à un partage réseau (par exemple une imprimante ou une périphérique média) ou à un disque d'une autre machine. Pour établir une connexion à un lecteur réseau, cliquez avec le bouton droit de la souris sur Réseau puis sélectionnez Connecter un lecteur réseau afin de donner lieu à l'assistant du même nom. Vous pouvez lancer la console Ordinateur en saisissant la commande shell:mycomputerforlder depuis le contrôle Exécuter.

### DiskPart

Homologue, sur le plan non graphique, de la console Gestion des disques, DiskPart permet d'administrer disques, partitions et volumes depuis l'invite de commandes, ou plus avantageusement encore par l'intermédiaire de scripts.

Pour faire appel à Diskpart, saisissez diskpart dans une invite de commandes élevée. Un prompt s'affiche alors, signe que vous êtes à ce moment entré au niveau de l'interprète de commandes servant d'interface pour le mode interactif. Tapez help ou un point d'interrogation pour voir une liste énumérant toutes les commandes et leur utilisation.

### Fsutil

Le tableau suivant répertorie les sous-commandes que fsutil prend en charge.

***Tableau 16.1. Commandes fsutil***

| Commande | Description |
|---|---|
| 8dot3name | Gestion des noms au format 8.3 |
| behavior | Comportement du système de fichiers |
| dax | Gestion des volumes DAX (Direct Access) |
| dirty | Gestion du bit d'intégrité du volume |
| file | Commandes spécifiques aux fichiers |
| fsinfo | Informations sur le système de fichiers |
| hardlink | Gestion des liaisons permanentes |
| objectid | Gestion des ID d'objet |
| quota | Gestion des quotas de disque |
| repair | Gestion des réparations spontanées |
| reparsepoint | Gestion des points d'analyse |
| resource | Gestion du Gestionnaire de ressources de transaction |
| sparse | Gestion des fichiers partiellement alloués |
| tiering | Gestion des propriétés de hiérarchisation du stockage |
| transaction | Gestion des transactions |
| usn | Gestion du journal USN (Update Sequence Number) |
| volume | Gestion des volumes |
| wim | Gestion de l'hébergement WIM |

## ReadyBoost

Sur le plan structurel, les fonctions implantées dans ReadyBoost voient le jour par l'intermédiaire d'un service, concrétisé par un processus hôte de service exécutant emdmgmt.dll, et d'un pilote de filtre, Ecache.sys. (EMD est l'abréviation de External Memory Device, soit le nom de ReadyBoost pendant la phase de développement).

Lorsque vous insérez une unité de stockage flash, telle qu'une clé USB, dans un des connecteurs de support de l'ordinateur, le service Windows ReadyBoost effectue automatiquement une étude dudit périphérique, cela

afin de déterminer si ses caractéristiques de performances conviennent pour une utilisation en tant que source complémentaire de cache. Les résultats de cette analyse sont stockés dans le registre sous HKLM\Software \Microsoft\Windows NT\Currentversion\Emdmgmt.

Pour des raisons de sécurité éminemment compréhensibles, l'ensemble de l'information mise en cache sur un périphérique associé à ReadyBoost bénéficie d'une protection cryptographique. Le pilote chiffre chacun des blocs écrits à l'aide d'un chiffrement AES en utilisant une clé de session générée aléatoirement au démarrage de l'ordinateur. Les données de cache d'une station de travail ne peuvent de cette façon pas être lues depuis une autre.

## ReadyDrive

Contrairement à ReadyBoost, qui fonctionne conjointement à une mémoire flash externe, Windows ReadyDrive fait appel à des disques durs spécialement conçus qui possède une mémoire flash intégrée. Combinant technologie à plateaux et support électronique, les disques hybrides offrent l'opportunité d'améliorer considérablement les temps de démarrage et de reprise après une mise en veille (simple ou prolongée), ainsi que de diminuer de façon sensible la rotation du disque physique (comprendre que ce dernier se trouve à l'arrêt plus souvent qu'à l'accoutumée), ce qui tend à en réduire l'usure, et dans un contexte de mobilité (ordinateurs portables) à économiser la batterie.

Comme d'autres stratégies d'optimisation de niveau système, la fonctionnalité ReadyDrive ne requiert pas l'attention d'un opérateur, étant activée automatiquement dès lors que l'ordinateur est doté de lecteurs hybrides.

## Rechercher des erreurs sur les disques

Les erreurs affectant disques et systèmes de fichiers peuvent très sérieusement écorner la fiabilité de l'ordinateur, allant de l'incapacité d'ouvrir ou d'enregistrer des fichiers à l'apparition de pannes de premier niveau (effondrements). Afin de parer à de tels imprévus, Windows intègre nativement diverses méthodes permettant de solutionner de manière pérenne de nombreuses erreurs de disque.

Pour connaître l'intégrité d'un support de stockage, et réparer les éventuelles erreurs qui s'y trouvent, employez l'utilitaire Vérification de disques, chkdsk.exe. Il en existe deux versions - une version graphique, qui comprend des fonctions basiques et présente l'avantage d'une prise en main très simple, et une version en ligne de commande, qui offre un éventail d'options beaucoup plus large.

Pour vérifier si un disque comporte des erreurs, suivez ces étapes :

1. Ouvrez Ordinateur, cliquez avec le bouton droit sur l'icône du lecteur à vérifier puis choisissez Propriétés dans le menu contextuel. Ceci fait, Windows fait apparaitre la fenêtre des propriétés du lecteur.

2. Dans l'onglet Outils, cliquez sur Vérifier.

3. Dans la fenêtre nouvellement affichée, choisissez Analyser le lecteur pour commencer le processus de vérification du disque.

Notez que vérifier un disque peut prendre un certain temps : selon les paramètres spécifiés (analyse seule, réparation, etc.), le nombre de fichiers concernés, le taux de fragmentation et la taille totale du volume à inspecter, une tel processus peut durer de quelques minutes à plusieurs heures. Notez encore que Vérification de disques identifie les problèmes essentiellement sur la base d'incohérences dans le système de fichiers et les métadonnées que ce dernier englobe (index des fichiers, journal USN, etc.), et n'est donc pas en mesure de réparer des données corrompues au sein de fichiers qui semblent être structurellement intacts.

Comme nous l'avons déjà mentionné, Vérification de disques honore deux modes d'utilisation : graphique et texte. Les commutateurs de la commande chkdsk en ligne de commande s'emploient comme suit :

- *nomdefichier* Spécifie les fichiers à contrôler du point de vue de la fragmentation (FAT/FAT32 seulement)

- *volume* Spécifie la lettre de lecteur (suivie du signe deux-points), le point de montage ou le nom de volume correspondant au stockage à manipuler.
- */?* Affiche l'aide à l'invite de commandes.
- */C* Ignore la vérification des cycles au sein de l'arborescence de dossiers.
- */F* Corrige les erreurs sur le disque. Le disque doit être verrouillé. Dans l'éventualité où chkdsk ne peut pas verrouiller le lecteur, un message s'affiche vous demandant si vous souhaitez vérifier le lecteur la prochaine fois que vous redémarrez l'ordinateur.
- */I* Effectue une vérification minimale des entrées d'index (NTFS seulement).
- */L:taille* Modifie la taille du fichier journal (NTFS seulement). La taille du fichier journal par défaut pour chkdsk est de 65 536 Ko. Vous pouvez vérifier la taille actuelle du fichier journal en spécifiant le commutateur /L sans lui adjoindre de paramètre de taille.
- */R* Localise les secteurs défectueux et tente de récupérer les informations qui peuvent l'être (implique l'emploi du commutateur /F).
- */V* Pour les systèmes de fichiers FAT/FAT32, affiche le chemin d'accès complet et le nom de chaque fichier lors de la vérification du disque. Sur les systèmes de fichiers NTFS, affiche en plus les éventuels messages de nettoyage.
- */X* Force le démontage préalable du disque si nécessaire (implique l'emploi du commutateur /F).
- */forceofflinefix* Ignore toute réparation en ligne (NTFS seulement, implique l'emploi du commutateur /scan). Les anomalies éventuellement rencontrées sont mises en file d'attente pour pour une réparation hors connexion (correspondant au commutateur /spotfix).
- */freeorphanedchains* Libère les chaînes de cluster orphelines au lieu de récupérer leur contenu (FAT/FAT32/exFAT seulement).
- */markclean* Marque le volume comme étant nettoyé si aucune corruption n'a été détectée, même si l'option /F n'a pas été spécifiée (FAT/FAT32/exFAT seulement)
- */offlinescanandfix* Effectue une analyse et une réparation hors connexion sur le volume.
- */perf* Maximise la rapidité du processus d'analyse, à l'éventuel détriment des autres tâches effectuées sur le système.
- */scan* Exécute une analyse en ligne sur le volume (NTFS uniquement). En ligne, dans ce contexte, signifie que le volume n'a pas besoin d'être démonté, mais peut rester actif tout au long du processus d'analyse.
- */spotfix* Passe en revue les anomalies rencontrées lors d'une analyse en ligne et tente de les résoudre hors ligne.

## Nettoyage de disque

La saturation de l'espace disque constitue parfois le facteur le plus explicatif d'une dégradation des performances. Il est dont utile de vérifier préventivement le taux de remplissage des différents volumes et, le cas échéant, prendre les mesures qui s'imposent pour améliorer ou corriger la situation. La fonction Nettoyage de disque de Windows offre de telles possibilités.

Dans un premier temps, Nettoyage de disque (cleanmgr.exe) parcourt le disque en vue d'identifier quels fichiers ne sont plus utiles (temporaires, doublons, précédentes installations du système, contenu de la corbeille, etc.), et après quoi propose à l'utilisateur de supprimer ces éléments. Les types de fichiers analysés et proposés à la suppression sont relativement nombreux, à savoir :

- Fichiers programmes téléchargés
- Fichiers Internet temporaires
- Fichiers archivés de rapport d'erreurs
- Cache de nuanceur DirectX
- Fichiers d'optimisation de livraison
- Téléchargements
- Corbeille
- Fichiers temporaires
- Miniatures

Suivez ces étapes pour utiliser Nettoyage de disque.

1. Depuis le groupe de programmes Outils d'administration, effectuez un double-clic sur l'icône Nettoyage de disque.
2. Après avoir sélectionné le lecteur à analyser dans la liste déroulante, cliquez sur le bouton OK.
3. Cochez les catégories à nettoyer et repérez l'espace disque qui sera récupéré dans la zone Espace total gagné.
4. En cochant une catégorie de fichiers, vous pouvez voir quels fichiers sont appelés à être supprimée en cliquant sur le bouton Afficher les fichiers. Une boite de dialogue apparait alors, avec la totalité des fichiers de la rubrique concernée.

## Cacher des lecteurs de disque

Il est très simple d'invisibiliser des volumes auprès de l'Explorateur Windows. Pour ce faire, sollicitez l'Éditeur du Registre (ou tout autre utilitaire du même genre) et rendez-vous à HKCU\Software\Microsoft\Windows \CurrentVersion\Policies\Explorer, clé sous laquelle la valeur NoDrives énumère les mappages de lecteur masqués. Les correspondances lecteurs / valeurs sont de la sorte : 1 pour le lecteur A, 2 pour le lecteur B, 3 pour le lecteur C, et ainsi de suite.

Notez que lecteurs masqués à l'aide du paramètre de registre NoDrives le sont uniquement aux interfaces Windows standards (Poste de travail, Explorateur, boîte de dialogue Fichier/Ouvrir, etc.), mais restent toujours accessibles, notamment en mode commande.

## Changer la lettre d'un lecteur

Tout lecteur de stockage reconnu par Windows, Indépendamment des caractéristiques qui le définit, se voit assigné une lettre et un label (ou nom). Si ce ce dernier n'a pas de valeur autre qu'informative, la lettre d'affectation est quant à elle beaucoup plus significative, et partant, beaucoup plus importante.

Procédez comme suit pour modifier la lettre d'affectation d'un lecteur :

1. Démarrez le composant enfichable MMC Gestion des disques (diskmgmt.msc).
2. À l'aide du bouton droit de la souris, cliquez sur le lecteur dont vous souhaitez changer la lettre. Dans le Menu contextuel alors affiché, sélectionnez la commande Modifier la lettre de lecteur et les chemins d'accès.
3. Dans la boite de dialogue Modifier la lettre de lecteur et les chemins d'accès..., cliquez sur le bouton Modifier.

4. Dans la liste déroulante de l'option Attribuer la lettre de lecteur suivante, sélectionnez la nouvelle lettre du lecteur.

5. Validez en cliquant sur OK.

## Scinder un disque en plusieurs lecteurs

Les choix opérés en matière de stockage, s'il convient de les considérer avec méthode et attention, n'ont pour la plupart pas de caractère définitif. Voici par exemple la marche à suivre en vue de récupérer une partie vide (ou non utilisée) d'un disque pour en faire un nouveau lecteur logique.

1. À partir de la console Gestion des disques (diskmgmt.msc), identifiez le lecteur à traiter et, du bouton droit de la souris, cliquez dessus.

2. Dans le menu contextuel, sélectionnez Réduire le volume.

3. Dans le champ de la fenêtre Réduire, saisissez la quantité d'espace à réduire. Ladite boite de dialogue fournit les informations suivantes :

   - *Taille totale en Mo avant réduction* Capacité totale actuelle du volume en Mo. Il s'agit de la taille formatée du volume.

   - *Espace de réduction disponible (en Mo)* Quantité maximale d'espace disque qu'il est possible de réduire sur le volume. Il ne s'agit pas de la quantité totale d'espace libre sur le volume, mais de la quantité d'espace susceptible d'être réclamée, sans inclure les éventuelles données réservées au système d'exploitation (fichiers d'échange, MFT, clichés instantanés, etc.)

   - *Quantité d'espace à réduire (en Mo)* Quantité totale d'espace dont sera amputé le volume. La valeur initiale correspond par défaut à la quantité maxime d'espace qu'il est possible de retirer au volume.

   - *Taille totale en Mo après réduction* Capacité totale du volume, en Mo, une fois que le processus de réduction aura eu lieu.

4. Validez en cliquant sur le bouton Réduire.

5. Une fois le processus terminé (et, partant, la réduction effectuée), le volume apparait alors avec un espace non alloué d'une taille correspondant à la quantité d'espace qui a été soustraite au lecteur d'origine. Effectuez un clic secondaire sur cette zone puis, dans le menu contextuel, sélectionnez Nouveau volume simple.

6. La fenêtre Assistant de création d'un volume simple apparait alors. Suivez les diverses étapes qui vous sont alors proposées pour finaliser la configuration d'un nouveau volume.

7. Sa création terminée, le nouveau volume est rendu visible par le système d'exploitation.

## Compresser des disques

Pour compresser tout le contenu d'un disque, procédez en suivant les indications ci-dessous :

1. Dans l'Explorateur Windows ou la console Gestion des disques, faites un clic droit sur le disque à compresser puis choisissez Propriétés.

2. Sélectionnez Compresser ce lecteur pour augmenter l'espace disponible et cliquez sur le bouton OK.

## Compresser des répertoires ou des fichiers

Windows offre pour diminuer l'espace qu'occupe sur le disque fichiers et repértoires la possibilité de les compresser. Pour cela :

1. Via l'Explorateur Windows, déplacez-vous dans l'arborescence jusqu'au dossier ou fichier que vous souhaitez compresser et sélectionnez-le avec un clic droit.

2. Dans le menu contextuel qui s'affiche, choisissez Propriétés.

3. Dans la boite de dialogue Propriétés de..., cliquez sur le bouton Avancé, situé au niveau de la zone Attributs.

4. La boite de dialogue Attributs avancés apparait ; cochez la case Compresser le contenu pour libérer de l'espace disque dans la zone Attributs de compression ou de chiffrement.

5. Cliquez sur le bouton OK pour valider la modification.

6. Cliquez sur le bouton OK de la boite de dialogue Propriétés de..., restée ouverte.

7. Si la modification des attributs concerne un fichier individuel, Windows le compresse. S'il s'agit d'un repertoire, Windows en compresse tous les fichiers. Si le repertoire contient des sous-dossiers, une boite de dialogue (Confirmation des modifications d'attributs) s'affiche, vous invitant à spécifier l'étendue des modifications (uniquement ce dossier, ou tous les sous-dossiers et fichiers). Dès qu'un répertoire a été compressé, tout fichier ajouté ou copié dans celui-ci est automatiquement compressé.

## Schéma de partition

La subdivision de tout support de stockage en plusieurs unités ne peut s'effectuer qu'à l'aune d'un agencement standard dit table de partitionnement, dont la question du choix le mieux adapté donne lieu à deux options : MBR (Master boot record) et GPT (GUID Partition Table).

### Partitionnement MBR

Les BIOS standard des machines x86 imposent pour ce qui concerne le partitionnement un certain nombre d'exigences, la plus fondamentale étant est que le premier secteur du disque doit contenir l'enregistrement d'amorçage principal (MBR, Master Boot Record). Au démarrage d'un processeur x86, le BIOS lit le MBR (comprendre qu'il le charge en mémoire) et traite une portion de ce dernier en tant que code exécutable, laquelle séquence sert dans ce contexte à initier le processus d'amorçage du système d'exploitation, ou éventuellement de passer le relais à un chargeur d'amorçage (*boot loader*).

En plus des unités d'information précitées, le MBR contient aussi la table des partitions. Une table de partitions comprend quatre entrées qui définissent les emplacements d'une à quatre partitions principales (ou primaires) du disque. Les partitions primaires sont des sous-ensemble du disque auxquels il est possible d'accéder pour y stocker des fichiers. Un type spécial de partition, dit *partition étendue*, contient un autre MBR avec sa propre table de partitions, et nécessite un effort de configuration supplémentaire, cela par le biais du fractionnement d'une partition étendue en disques logiques (nous y reviendrons plus loin). Un tel mécanisme permet de la sorte aux systèmes d'exploitation (et partant, à Windows) de s'affranchir de la limite de quatre partitions par disque.

Chaque partition définie dans le style MBR a, conformément au type auquel elle répond, un système de fichiers, lequel conditionne la manière dont sont en interne stockés les fichiers et répertoires sur le périphérique. De nombreux types de partition prédéfinis existent, tenant ainsi compte d'une large majorité des systèmes de fichiers, dont FAT32 et NTFS.

### Partitionnement GPT

Conçu à l'origine pour des ordinateurs hautes performances fondés sur Itanium, GPT (GUID Partition Table) est recommandé pour des disques de taille égale ou supérieure à 2 To. GPT tire son nom du fait que chaque partition définie dans ce schéma a un GUID qui lui est associé. Bien qu'émanant de la norme EFI (Extensible Firmware Interface), GPT est également utilisé sur certains BIOS, en grande partie à cause des limitations inhérentes au partitionnement MBR, qui limite la taille des partitions à 2 To. Autres avantages de GPT : il utilise de façon à garantir

l'intégrité des informations concernées (table des partitions, notamment) des sommes de contrôle CRC, et gère une copie de sauvegarde desdites informations. De ce fait, dans l'éventualité où des données sont corrompues, les systèmes GPT peuvent attirer l'attention sur le problème en cause, voire le résoudre en récupérant les données endommagées à un autre endroit du disque.

Contrairement à MBR, qui amalgame table de partitionnement et code maitre de démarrage, GPT lit le gestionnaire d'amorçage du système d'exploitation sous la forme d'un fichier enregistré sur disque. Dans Windows, cette opération est effectuée à l'aide du fichier bootmgfw.efi situé dans une partition indépendante dans le répertoire \EFI\Microsoft\Boot.

## Comparaison entre MBR et GPT

S'il existe d'importantes différences structurelles entre les styles de partition MBR et GPT (voir tableau plus bas), la plupart des tâches d'administration relatives aux disques s'effectuent dans les deux cas de la même façon. Cela signifie qu'une fois les disques en place et (correctement) configurés, peu importe qu'ils emploient l'un ou l'autre des schémas susmentionnés. Quelques remarques toutefois :

- Bien que GPT relève du standard EFI (au même titre que MBR relève du BIOS), les tables de partitionnement GUID peuvent également être utilisées pour le partitionnement sur les machines BIOS. Diverses limitations se présentent alors, dépendantes du système d'exploitation utilisé. Ainsi, les versions de Windows fonctionnant sur BIOS ne peuvent pas démarrer à partir d'un lecteur partitionné avec GPT.
- Il est possible de transformer un disque MBR en GPT, et vice-versa.
- Si toutes les versions de Windows (32 et 64 bits) savent lire et écrire des disques GPT, seules les versions 64 bits peuvent démarrer à partir d'un tel support.

***Tableau 16.2. Comparaison entre MBR et GPT***

| | **MBR** | **GPT** |
|---|---|---|
| Nombre de partitions primaires autorisées | 4 | Illimité (dépend du système d'exploitation) |
| Taille maximale de partition | 2 téraoctets (To) | 18 exaoctets (Eo) |
| Addresses de secteur employées | 32 bits | 64 bits |
| Sécurité | Aucune | Intégrité structurale (sommes de controle) et redondance des informations |
| Support pour l'amorçage multiple (multiboot) | Faible | Fort (chargeur d'amorçage enregistré sur une partition séparée) |

## Visualisation du schéma de partition utilisé

Pour savoir quel schéma de partition, de MBR ou GPT, est en vigueur sur le disque, suivez les étapes que voici.

*Visualisation du schéma de partition utilisé via DiskPart*

Ouvrez une fenêtre d'invite de commande, saisissez diskpart et validez. Dans l'éventualité où une invite UAC est affichée, cliquez sur OK. Depuis la session DiskPart interactive, entrez la commande list disk. Vous devriez à ce moment voir quelque chose de similaire à ce qui suit.

```
DISKPART> list disk

  N° disque  Statut         Taille   Libre    Dyn  GPT
  ``````-  `````````-  ```---  ```---  ---  ---
  Disque 0    En ligne        931 G octets    77 M octets        *
  Disque 1    En ligne       3725 G octets   128 M octets        *
```

Une étoile est affichée sous la colonne GPT pour chaque lecteur conforme à cet égard.

*Visualisation du schéma de partition utilisé via Gestion des disques*
```

Depuis une console MMC Gestion des disques (diskmgmt.msc), faites un clic droit sur la case du disque concerné et cliquez sur Propriétés. Dirigez-vers vers l'onglet Volumes, dans lequel l'entrée Type de partition indique Secteur de démarrage principal si le disque est MBR ou Taille de partition GUID (GPT) s'il est GPT.

*Visualisation du schéma de partition utilisé via Powershell*

Après avoir ouvert une session dans l'invite de commandes PowerShell, saisissez la commande Get-Disk qui, entre autres caractéristiques, affiche à quel style se rattache un disque.

```
PS C:\> get-disk
Number Friendly Name Serial Number                  HealthStatus         OperationalStatus      Total Size Partition
                                                                                                            Style
------ ------------- -------------                  ------------         -----------------      ---------- ----------
0      HGST HTS72... JR100YBNJZVHXM                 Healthy              Online                  931.51 GB GPT
1      WD Element... WX41EA9CE6P8                   Healthy              Online                    3.64 TB GPT
```

## Organisation sur disque de GPT

En tant que système de partitionnement moderne, GPT opère sur la base de blocs logiques , et utilise de fait pour adresser les secteurs d'un disque la méthode LBA. Tout disque conforme à GPT honore l'agencement que voici.

- Pour des raisons de compatibilité, le premier secteur (LBA 0) d'un disque GPT consiste en un enregistrement d'amorçage principal (MBR), appelée en la circonstance MBR protecteur, et servant ici à protéger les disques GPT des écritures provenant d'utilitaires disques qui ne reconnaissent pas les informations de GPT.
- Le premier en-tête GPT, qui commence sur le deuxième bloc logique (LBA 1) du périphérique, renferme des caractéristiques tels le GUID du disque, les secteurs utilisables pour les partitions et l'emplacement de table des partitions.
- La table des partitions principale (LBA 2) inclut, par défaut, 128 entrées de partition, chacune concernant une partition.
- La table des partitions secondaire, en principe identique à la table principale. Elle est généralement employée comme table de sauvegarde dans l'éventualité où la première table viendrait à être endommagée.
- Le second en-tête GPT, qui réside sur le dernier secteur logique du disque, et peut être utilisé pour récupérer les informations d'une table GPT en cas d'endommagement du premier en-tête.

### MBR hérité (ou protecteur)

Les premières informations à se voir concrétiser dans un disque GPT le sont sous forme d'un enregistrement d'amorçage principal (MBR). Au-delà d'offrir une certaine forme de compatibilité avec les logiciels gérant uniquement MBR (compatibilité rétroactive, donc), cette structure, dite MBR protecteur (*protective MBR*), sert en réalité surtout à empêcher lesdits logiciels de manipuler sans discernement des disques GPT, et auquel cas d'écraser les données qui s'y trouvent.

```
0000000000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
00000000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000001A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000001B0  00 00 00 00 00 00 00 00 24 2C 75 C3 00 00 00 00  ........$,uÃ....
00000001C0  02 00 EE FE 7F 80 01 00 00 00 FF FF FF FF 00 00  ..îþ.€....ÿÿÿÿ..
00000001D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000001E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000001F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA  ..............Uª
```

Ici (et c'est traditionnellement le cas), Windows a défini une seule partition (offset 1BE-1CD) de type GPT (0xEE) qui recouvre 2 To du disque (FF FF FF FF, 4294967295 secteurs, soit la taille maximale représentable avec une adresse 32 bits et des blocs de 512 octets). Les systèmes ou logiciels non compatibles avec GPT qui voient le disque refusent en la circonstance de le modifier, à moins d'effacer la partition. Une telle façon de procéder permet ainsi de minimiser les risques d'effacement accidentel.

## Entête GPT

```
0000000200  45 46 49 20 50 41 52 54 00 00 01 00 5C 00 00 00  EFI PART....\...
0000000210  84 6E 1B C4 00 00 00 00 01 00 00 00 00 00 00 00  „n.Ä............
0000000220  AF 6D 70 74 00 00 00 00 22 00 00 00 00 00 00 00  ¯mpt....".......
0000000230  8E 6D 70 74 00 00 00 00 20 10 56 57 8B 94 87 45  Žmpt.... .VW‹”‡E
0000000240  A8 DF 77 F7 8C 19 A7 98 02 00 00 00 00 00 00 00  ¨ßw÷Œ.§˜........
0000000250  80 00 00 00 80 00 00 00 5D F0 C3 29 00 00 00 00  €...€...]ðÃ)....
0000000260  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000270  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000280  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000290  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000002A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000002B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000002C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000002D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000002E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000002F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000300  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000310  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000320  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000330  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000340  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000350  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000360  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000370  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000380  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000390  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000003A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000003B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000003C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000003D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000003E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000003F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

- **Signature** Qualifie les les unités d'information après celle-ci en tant que sous-ensemble d'un système de partitionnement GPT. Exemple (offset 0-7) : 45 46 49 20 50 41 52 54, à quoi correspond la chaîne de caractères EFI PART.

- **Version** Numéro de version du standard EFI auquel l'entête GPT est conforme. Exemple (offset 8-11) : 00 00 01 00 (0x 0x00000100) pour la version 1.0.
- **Taille de l'entête** Comptabilise le nombre d'octets qu'occupe l'entête GPT. Exemple (offset 12-15) : 5C 00 00 00, soit 92 octets.
- **Protection de l'entête** Somme de contrôle CRC générée à partir des 92 premiers octets de l'entête GPT. Exemple (offset 16-19) : 84 6E 1B C4.
- **Réservé** Séquence d'octets réservée pour une utilisation ultérieure. Exemple (offset 20-23) : 00 00 00 00.
- **Position de l'entête primaire** Numéro du secteur où loge l'entête GPT primaire. Exemple (offset 24-31) : 01 00 00 00 00 00 00 00, ou secteur 1, correspondant au second secteur du disque.
- **Position de l'entête secondaire** Numéro du secteur où loge l'entête GPT secondaire, offrant une copie de sauvegarde l'entête principale. Exemple (offset 32-39) : AF 6D 70 74 00 00 00 00, ici le secteur 1 953 525 167, dernier du disque.
- **Premier secteur adressable** Numéro du premier secteur utilisable pour les partitions. Exemple (offset 40-47) : 22 00 00 00 00 00 00 00, ou secteur 34.
- **Dernier secteur adressable** Numéro du dernier secteur utilisable pour les partitions. Exemple (offset 48-55) : 22 00 00 00 00 00 00 00, ou secteur 1 953 525 134.
- **GUID du disque**Exemple (offset 56-71) : 20 10 56 57 8B 94 87 45 A8 DF 77 F7 8C 19 A7 98
- **Position de la table des partitions** Numéro du secteur où loge la table des partitions. Exemple (offset 72-79) : 02 00 00 00 00 00 00 00, ou secteur 2.
- **Nombre de partitions autorisées** Nombre de partitions que le système d'exploitation autorise sur un même disque. Exemple (offset 80-83) : 80 00 00 00, soit 128 partitions.
- **Taille d'un descripteur de partition** Comptabilise le nombre d'octets qu'occupe un descripteur de partition. Exemple (offset 84-87) : 80 00 00 00, soit 128 octets.
- **Protection de la table des protections** Somme de contrôle CRC générée à partir des données que renferme la table des partitions. Exemple (offset 88-91) : 5D F0 C3 29.

## Table des partitions

```
Offset(h)   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000000400  28 73 2A C1 1F F8 D2 11 BA 4B 00 A0 C9 3E C9 3B  (s*Á.øÒ.ºK. É>É;
0000000410  B0 9C 8D 03 04 C8 79 4D 85 82 64 3F 91 09 FF 0B  °œ...ÈyM...,d?'.ÿ.
0000000420  00 08 00 00 00 00 00 00 FF 27 03 00 00 00 00 00  ........ÿ'......
0000000430  00 00 00 00 00 00 00 80 45 00 46 00 49 00 20 00  .......€E.F.I. .
0000000440  73 00 79 00 73 00 74 00 65 00 6D 00 20 00 70 00  s.y.s.t.e.m. .p.
0000000450  61 00 72 00 74 00 69 00 74 00 69 00 6F 00 6E 00  a.r.t.i.t.i.o.n.
0000000460  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000470  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000480  16 E3 C9 E3 5C 0B B8 4D 81 7D F9 2D F0 02 15 AE  .ãÉã\.¸M.}ù-ð..®
0000000490  D2 89 B6 67 48 4E 4C 46 8F 12 01 53 59 A1 0F C1  Ò‰¶gHNLF...SY¡.Á
00000004A0  00 28 03 00 00 00 00 00 FF 27 07 00 00 00 00 00  .(......ÿ'......
00000004B0  00 00 00 00 00 00 00 80 4D 00 69 00 63 00 72 00  .......€M.i.c.r.
00000004C0  6F 00 73 00 6F 00 66 00 74 00 20 00 72 00 65 00  o.s.o.f.t. .r.e.
00000004D0  73 00 65 00 72 00 76 00 65 00 64 00 20 00 70 00  s.e.r.v.e.d. .p.
00000004E0  61 00 72 00 74 00 69 00 74 00 69 00 6F 00 6E 00  a.r.t.i.t.i.o.n.
00000004F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000500  A2 A0 D0 EB E5 B9 33 44 87 C0 68 B6 B7 26 99 C7  ¢ Ðëå¹3D‡Àh¶·&™Ç
0000000510  DE 2E FE 7D 86 21 17 4A 96 1D 61 6F A7 CE 6D 91  Þ.þ}†!.J–.ao§Îm'
```

```
0000000520  00 28 07 00 00 00 00 00 FF B7 F5 22 00 00 00 00  .(......ÿ·õ"....
0000000530  00 00 00 00 00 00 00 00 42 00 61 00 73 00 69 00  ........B.a.s.i.
0000000540  63 00 20 00 64 00 61 00 74 00 61 00 20 00 70 00  c. .d.a.t.a. .p.
0000000550  61 00 72 00 74 00 69 00 74 00 69 00 6F 00 6E 00  a.r.t.i.t.i.o.n.
0000000560  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000570  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000580  A4 BB 94 DE D1 06 40 4D A1 6A BF D5 01 79 D6 AC  ¤»”ÞÑ.@M¡j¿Õ.yÖ¬
0000000590  CB DE 3E FF 75 A3 DF 4C B4 C5 C0 CE 5C A1 D0 B3  ËÞ>ÿu£ßL´ÅÀÎ\¡Ð³
00000005A0  00 B8 F5 22 00 00 00 00 FF B7 0E 23 00 00 00 00  .¸õ"....ÿ·.#....
00000005B0  01 00 00 00 00 00 00 80 42 00 61 00 73 00 69 00  .......€B.a.s.i.
00000005C0  63 00 20 00 64 00 61 00 74 00 61 00 20 00 70 00  c. .d.a.t.a. .p.
00000005D0  61 00 72 00 74 00 69 00 74 00 69 00 6F 00 6E 00  a.r.t.i.t.i.o.n.
00000005E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000005F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

- **Type de partition** Modèle auquel se conforme la partition. Exemple (offset 0-15) : 28 73 2A C1 1F F8 D2 11 BA 4B 00 A0 C9 3E C9 3B, à qui correspond le GUID C12A7328-F81F-11D2-BA4B-00A0C93EC93B, associé à une partition système EFI.
- **GUID de partition** Valeur unique assignée à la partition en vue de facilement l'identifier. Exemple (offset 16-31) : B0 9C 8D 03 04 C8 79 4D 85 82 64 3F 91 09 FF 0B.
- **Premier secteur** Numéro du premier secteur de la partition. Exemple (offset 32-39) : 00 08 00 00 00 00 00 00, ou secteur 2048.
- **Dernier secteur** Numéro du dernier secteur de la partition. Exemple (offset 40-47) : FF 27 03 00 00 00 00 00, ou secteur 206847.
- **Attributs** Caractéristiques de la partition. Exemple (offset 48-55) : 00 00 00 00 00 00 00 80.
- **Nom** Nom attribué à la partition. Les noms établis à cet égard dépendent du type auquel se rapporte la partition.

*Tableau 16.3. Types de partition*

| Type | GUID |
|---|---|
| Partition système EFI | C12A7328-F81F-11D2-BA4B-00A0C93EC93B |
| Réservé Microsoft | E3C9E316-0B5C-4DB8-817D-F92DF00215AE |
| Partition de données de base | EBD0A0A2-B9E5-4433-87C0-68B6B72699C7 |
| Windows Recovery Environment | DE94BBA4-06D1-4D40-A16A-BFD50179D6AC |

## Table des partitions secondaire

Afin d'anticiper une éventuelle corruption de la table des partitions, chaque disque GPT en conserve une copie de sauvegarde, généralement enregistré à partir du trente-troisième secteur à partir de la fin du disque.

```
Offset(h)   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000000400  28 73 2A C1 1F F8 D2 11 BA 4B 00 A0 C9 3E C9 3B  (s*Á.øÒ.ºK. É>É;
0000000410  B0 9C 8D 03 04 C8 79 4D 85 82 64 3F 91 09 FF 0B  °œ...ÈyM...,d?'.ÿ.
0000000420  00 08 00 00 00 00 00 00 FF 27 03 00 00 00 00 00  ........ÿ'......
0000000430  00 00 00 00 00 00 00 80 45 00 46 00 49 00 20 00  .......€E.F.I. .
0000000440  73 00 79 00 73 00 74 00 65 00 6D 00 20 00 70 00  s.y.s.t.e.m. .p.
0000000450  61 00 72 00 74 00 69 00 74 00 69 00 6F 00 6E 00  a.r.t.i.t.i.o.n.
0000000460  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000470  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
.
.
.
Offset(h)   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
E8E0DB1E00  28 73 2A C1 1F F8 D2 11 BA 4B 00 A0 C9 3E C9 3B  (s*Á.øÒ.ºK. É>É;
E8E0DB1E10  B0 9C 8D 03 04 C8 79 4D 85 82 64 3F 91 09 FF 0B  °œ...ÈyM...,d?'.ÿ.
```

```
E8E0DB1E20  00 08 00 00 00 00 00 FF 27 03 00 00 00 00 00  ........ÿ'......
E8E0DB1E30  00 00 00 00 00 00 00 80 45 00 46 00 49 00 20 00  .......€E.F.I. .
E8E0DB1E40  73 00 79 00 73 00 74 00 65 00 6D 00 20 00 70 00  s.y.s.t.e.m. .p.
E8E0DB1E50  61 00 72 00 74 00 69 00 74 00 69 00 6F 00 6E 00  a.r.t.i.t.i.o.n.
E8E0DB1E60  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
E8E0DB1E70  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
.
.
.
```

## Entête secondaire

```
Offset(h)   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
E8E0DB5E00  45 46 49 20 50 41 52 54 00 00 01 00 5C 00 00 00  EFI PART....\...
E8E0DB5E10  66 D6 86 67 00 00 00 00 AF 6D 70 74 00 00 00 00  fÖ†g....¯mpt....
E8E0DB5E20  01 00 00 00 00 00 00 00 22 00 00 00 00 00 00 00  ........".......
E8E0DB5E30  8E 6D 70 74 00 00 00 00 20 10 56 57 8B 94 87 45  Žmpt.... .VW‹”‡E
E8E0DB5E40  A8 DF 77 F7 8C 19 A7 98 8F 6D 70 74 00 00 00 00  ¨ßw÷Œ.§˜.mpt....
E8E0DB5E50  80 00 00 00 80 00 00 00 5D F0 C3 29 00 00 00 00  €...€...]ðÃ)....
.
.
.
```

Pour rappel, l'entête primaire, tel qu'il se présente dans notre configuration, se compose comme suit :

```
Offset(h)   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000000200  45 46 49 20 50 41 52 54 00 00 01 00 5C 00 00 00  EFI PART....\...
0000000210  84 6E 1B C4 00 00 00 00 01 00 00 00 00 00 00 00  „n.Ä............
0000000220  AF 6D 70 74 00 00 00 00 22 00 00 00 00 00 00 00  ¯mpt....".......
0000000230  8E 6D 70 74 00 00 00 00 20 10 56 57 8B 94 87 45  Žmpt.... .VW‹”‡E
0000000240  A8 DF 77 F7 8C 19 A7 98 02 00 00 00 00 00 00 00  ¨ßw÷Œ.§˜........
0000000250  80 00 00 00 80 00 00 00 5D F0 C3 29 00 00 00 00  €...€...]ðÃ)....
.
.
.
```

Enfin, voici à titre informatif, quels attributs diffèrent entre les deux entêtes :

- **Somme de contrôle** 84 6E 1B C4 dans un cas, 66 D6 86 67 dans l'autre.
- **Position de l'entête** 01 00 00 00 00 00 00 00, soit le premier secteur dans un cas, AF 6D 70 74 00 00 00 00, soit le 1 953 525 167ème secteur dans l'autre. Chaque copie pointe de la sorte vers l'autre.
- **Position de l'autre entête** AF 6D 70 74 00 00 00 00, soit le 1 953 525 167ème secteur dans un cas, le premier (01 00 00 00 00 00 00 00) dans l'autre. Chaque copie pointe de la sorte vers l'autre.

## Volumes actifs, système et de démarrage

Quelques-uns des volumes définis sur les disques enregistrent des informations essentielles au bon fonctionnement du système, et méritent à cet égard une attention spéciale.

- **Actif** Le volume Actif est celui à partir duquel la machine démarre après sa mise sous tension. De ce fait, le volume actif contient le code machine exécutable donnant lieu au choix à système à exécuter (si plus d'un est installé), les fichiers de démarrage du système d'exploitation, et doit être une partition principale d'un disque de base. Le processus de démarrage continue à partir du volume dit Système.
- **Système** Le volume Système est celui à partir duquel le système d'exploitation démarre. Il contient les fichiers spécifiques au matériel qui chargent le système d'exploitation. Il existe toujours un volume système, tandis que chacun des systèmes Windows installé sur la machine conserve habituellement ses fichiers sur son propre volume, appelé volume de démarrage.

- **Démarrer** Le volume Démarrer correspond à celui où les fichiers sous-jacents à un système Windows particulier sont stockés. Sur la plupart des configurations, les volumes Système et Démarrer ne font qu'un.
- **Fichier d'échange** Un volume fichier d'échange contient un fichier sur lequel s'appuie le système pour paginer une partie de la mémoire vers le disque. Windows étant capable de gérer plusieurs fichiers de pagination sur différents supports, il est tout à fait possible que l'étiquette Fichier d'échange se présente sur plusieurs volumes.
- **Vidage sur incident** Le volume Vidage sur incident est celui où Windows mémorise les données qui résultent d'un effondrement du système. Par défaut, les informations enregistrées à cette occasion le sont dans le %SystemRoot%, mais peuvent l'être sur n'importe quel volume.

## Organisation disque de MBR

Chaque périphérique MBR renferme comme premier secteur un enregistrement de démarrage principal, lequel comporte la table des partitions du disque et séquence d'instructions exécutables.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000000000  33 C0 8E D0 BC 00 7C 8E C0 8E D8 BE 00 7C BF 00  3ÀŽÐ¼.|ŽÀŽØ¾.|¿.
000000010  06 B9 00 02 FC F3 A4 50 68 1C 06 CB FB B9 04 00  .¹..üó¤Ph..Ëû¹..
000000020  BD BE 07 80 7E 00 00 7C 0B 0F 85 0E 01 83 C5 10  ½¾.€~..|.....ƒÅ.
000000030  E2 F1 CD 18 88 56 00 55 C6 46 11 05 C6 46 10 00  âñÍ.ˆV.UÆF..ÆF..
000000040  B4 41 BB AA 55 CD 13 5D 72 0F 81 FB 55 AA 75 09  ´A»ªUÍ.]r..ûUªu.
000000050  F7 C1 01 00 74 03 FE 46 10 66 60 80 7E 10 00 74  ÷Á..t.þF.f`€~..t
000000060  26 66 68 00 00 00 00 66 FF 76 08 68 00 00 68 00  &fh....fÿv.h..h.
000000070  7C 68 01 00 68 10 00 B4 42 8A 56 00 8B F4 CD 13  |h..h..´BŠV.‹ôÍ.
000000080  9F 83 C4 10 9E EB 14 B8 01 02 BB 00 7C 8A 56 00  Ÿƒ Ä.žë.¸...».|ŠV.
000000090  8A 76 01 8A 4E 02 8A 6E 03 CD 13 66 61 73 1C FE  Šv.ŠN.Šn.Í.fas.þ
0000000A0  4E 11 75 0C 80 7E 00 80 0F 84 8A 00 B2 80 EB 84  N.u.€~.€.„Š.²€ë„
0000000B0  55 32 E4 8A 56 00 CD 13 5D EB 9E 81 3E FE 7D 55  U2äŠV.Í.]ëž.>þ}U
0000000C0  AA 75 6E FF 76 00 E8 8D 00 75 17 FA B0 D1 E6 64  ªunÿv.è..u.ú°Ñæd
0000000D0  E8 83 00 B0 DF E6 60 E8 7C 00 B0 FF E6 64 E8 75  èƒ.°ßæ`è|.°ÿædèu
0000000E0  00 FB B8 00 BB CD 1A 66 23 C0 75 3B 66 81 FB 54  .û¸.»Í.f#Àu;f.ûT
0000000F0  43 50 41 75 32 81 F9 02 01 72 2C 66 68 07 BB 00  CPAu2.ù..r,fh.».
000000100  00 66 68 00 02 00 00 66 68 08 00 00 00 66 53 66  .fh....fh....fSf
000000110  53 66 55 66 68 00 00 00 00 66 68 00 7C 00 00 66  SfUfh....fh.|..f
000000120  61 68 00 00 07 CD 1A 5A 32 F6 EA 00 7C 00 00 CD  ah...Í.Z2öê.|..Í
000000130  18 A0 B7 07 EB 08 A0 B6 07 EB 03 A0 B5 07 32 E4  . ·.ë. ¶.ë. µ.2ä
000000140  05 00 07 8B F0 AC 3C 00 74 09 BB 07 00 B4 0E CD  ...‹ð¬<.t.»..´.Í
000000150  10 EB F2 F4 EB FD 2B C9 E4 64 EB 00 24 02 E0 F8  .ëòôëý+Éädë.$.àø
000000160  24 02 C3 49 6E 76 61 6C 69 64 20 70 61 72 74 69  $.ÃInvalid parti
000000170  74 69 6F 6E 20 74 61 62 6C 65 00 45 72 72 6F 72  tion table.Error
000000180  20 6C 6F 61 64 69 6E 67 20 6F 70 65 72 61 74 69   loading operati
000000190  6E 67 20 73 79 73 74 65 6D 00 4D 69 73 73 69 6E  ng system.Missin
0000001A0  67 20 6F 70 65 72 61 74 69 6E 67 20 73 79 73 74  g operating syst
0000001B0  65 6D 00 00 00 63 7B 9A 19 82 76 15 00 00 00 20  em...c{š.,v....
0000001C0  21 00 0C FE FF FF 00 08 00 00 00 F8 B6 03 00 00  !..þÿÿ.....ø¶...
0000001D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000001E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000001F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA  ..............Uª
```

| Position(h) | Taille(o) | Attribut |
|---|---|---|
| 000 | 446 | Routine d'amorçage |
| 1b8 | 4 | Identifiant de disque |
| 1be | 64 | Table des partitions |
| 1fe | 2 | Signature |

### Table des partitions

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
```

```
0000001B0  .. .. .. .. .. .. .. .. .. .. .. .. .. .. 00 20  em...c{š.,v....
0000001C0  21 00 0C FE FF FF 00 08 00 00 00 F8 B6 03 00 00  !..þÿÿ.....ø¶...
0000001D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000001E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000001F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 .. ..  ..............Uª
```

| Position(h) | Taille(o) | Attribut |
|---|---|---|
| 00 | 1 | État |
| 01 | 1 | Tête de départ |
| 02 | 2 | Secteur/cylindre de départ |
| 04 | 1 | Type |
| 05 | 1 | Tête finale |
| 06 | 2 | Secteur/cylindre final |
| 08 | 4 | Secteur relatif |
| 0C | 4 | Nombre de secteurs |

- **État** Indique si la partition est amorçable (0x80) ou non (0x00).
- **Secteur/cylindre de départ** La valeur Secteur est mémorisée par les 6 bits de poids faible du premier octet ; la valeur Cylindre par les 2 bits de poids fort du premier octet et l'octet suivant.
- **Type** Indique le système de fichiers que la partition emploie et éventuellement les caractéristiques d'accès spéciales pour y accéder, par exemple si la partition est cachée.
- **Secteur/cylindre final** La valeur Secteur est mémorisée par les 6 bits de poids faible du premier octet ; la valeur Cylindre par les 2 bits de poids fort du premier octet et l'octet suivant.
- **Secteur relatif** Décalage entre le début du disque et le début de la partition, exprimé en nombre de secteurs.
- **Nombre de secteurs** Nombre total de secteurs que couvre la partition.

***Tableau 16.4. Types de partition***

| Type(h) | Description |
|---|---|
| 01 | partition ou lecteur logique FAT 12 |
| 04 | Partition ou lecteur logique FAT 16 |
| 05 | partition étendue |
| 07 | partition NTFS. |
| 0B | Partition principale FAT32 |
| 0C | Partition étendue FAT32 |

## Identifiant de disque

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000001B0  65 6D 00 00 00 63 7B 9A 19 82 76 15 00 00 00 20  em...c{š.,v....

DISKPART> detail disk

MMC Disk USB Device
ID du disque : 15768219
.
.
.
```

# Chapitre 17. Systèmes de fichiers

## Terminologie relative aux systèmes de fichiers

Un fichier est une unité informationnelle physiquement stockée sur un support de mémoire de masse permanent (par exemple, un disque dur). Un répertoire (appelé également dossier) est un objet informatique pouvant contenir des fichiers, ou d'autres répertoires. On appelle *chemin* (en anglais *path*) la succession des répertoires en partant de la racine pour atteindre un fichier. Sous les systèmes Windows, un chemin est de la forme : c:\répertoire1\répertoire2\fichier.txt.

Les clusters (ou unités d'allocation) sont les blocs d'un volume adressables par le système d'exploitation. L'unité d'allocation est, dans ce contexte, la plus petite quantité d'espace disque susceptible d'être allouée pour l'enregistrement d'un fichier. Moult systèmes de fichiers - à peu près tous en réalité, en tout cas tout ceux reconnus par Windows - utilisent des unités d'allocation plutôt que des secteurs car cela est plus efficace pour optimiser l'espace disque.

Les métadonnées sont les données d'un volume qui servent à la gestion interne du format de système de fichiers. Les principales dans ce contexte incluent le nom du fichier, sa taille, son emplacement sur le disque, des estampilles horaires le concernant (par exemple date de création et de dernière modification), et ainsi de suite.

La fragmentation interne est l'espace qui n'est pas utilisé par un fichier (ou une portion de fichier) quand il ne remplit pas complètement un bloc. Par exemple, en considérant une taille de bloc de 4096 octets et une taille de fichier de 2024 octets, les 2024 octets restants sont pour ainsi dire gaspillés.

## Montage et démontage

De même qu'un fichier demande d'être *ouvert* pour être employé, un système de fichiers doit être *monté* avant d'être disponible pour les processus du système. Un système de fichiers monté est *joint* à l'arborescence des répertoires du système au point de montage spécifié. Le système de fichiers racine est toujours et inconditionnellement monté. Tous les autres systèmes de fichiers peuvent être connectés (montés) ou déconnectés (démontés) soit à la demande, soit de manière permanente au démarrage de l'ordinateur.

La réalité la plus visible de la procédure de montage consiste en l'intégration au sein d'un espace de nommage spécifique du système de fichiers dont un volume est munie, permettant en pratique de rendre visible le contenu (structure des répertoires et fichiers) de ladite zone de stockage.

Windows associe une lettre à chacune des partitions gérées. Le chemin pour un fichier spécifique est exprimé sous la forme lettre-disque:\chemin\vers\fichier.

Les systèmes d'exploitation de la famille Windows découvrent automatiquement tous les disques et montent tous les systèmes de fichiers localisés au moment du démarrage de l'ordinateur. (D'autres systèmes, comme Unix, privilégient des commandes de montage explicites.)

Le démontage est l'opération inverse du montage, nécessaire par exemple lors du retrait d'un périphérique USB. Le montage ne peut fonctionner que si deux conditions cumulatives sont réunies : il faut d'une part, (1) qu'aucune opération d'E/S ne soit en cours concernant le volume, et (2) qu'aucun processus aie son répertoire de travail hébergé par le volume.

## Formats de système de fichiers Windows

Windows assure une prise en charge intégrée de plusieurs systèmes de fichiers principaux pour les périphériques directement connectés à l'ordinateur.

- **NTFS (New Technology File System)** Le système de fichiers principal et le plus avancé, conçu pour répondre aux besoins actuels à la fois en termes de sécurité, de fiabilité et de performances : gestion des noms longs, possibilité de configurer chaque fichier ou dossier avec des autorisations spécifiques, résistance aux pannes, chiffrement, et bien plus encore.

- **FAT (File Allocation Table)** Un système plus ancien, compatible avec de nombreux systèmes d'exploitation (Windows, macOS, Linux, etc.) mais encombré de faiblesses plus ou moins significatives : tailles et noms de fichiers limités (4 gigaoctets dans un cas, 255 caractères dans l'autre), pas de contrôle d'accès (ce qui signifie concrètement que tous les utilisateurs peuvent accéder librement à l'ensemble des fichiers et dossiers de la partition), absence de tout mécanisme de journalisation, etc.
- **exFAT (Extended File Allocation Table)** Optimisé pour les périphériques comme les clés USB, avec un meilleur support pour les fichiers de grande taille.
- **CDFS (CD-ROM File System)** Conçu pour accéder aux données sur CD-ROM.
- **Universal Disk Format (UDF)** Conçu pour les supports optiques modernes tels que DVD et Blu-ray.

## ISO-9660

ISO 9660, aussi connue sous l'appellation CDFS (Compact Disc File System), est une norme de système de fichiers destinée aux supports optique qui offre une pleine comptabilité entre ordinateurs et systèmes d'exploitation.

Une version gratuite de la norme ISO 9660 peut être téléchargée sur le site de l'association Ecma, sous la référence Ecma-119. La norme ISO 9660 a été élargie par les normes ISO/IEC 13346 et ISO/IEC 13490, qui peuvent être téléchargées respectivement sous les références Ecma-167 et Ecma-168.

Le standard ISO-9660 comprend trois niveaux d'échanges. L'ISO 9660 Niveau 1 peut être lu par presque toutes les plates-formes informatiques, mais impose en contrepartie des restrictions plutôt sévères. En voici quelques-unes :

- Seuls les majuscules (allant de A à Z), les chiffres (0 à 9) et le caractère de soulignement (_) sont autorisés dans les noms de fichiers.
- Seuls huit caractères pour le nom du fichier et trois pour son extension (fondé sur les limites du DOS).
- Les noms de répertoire ne peuvent excéder plus de 8 caractères. Ils sont soumis aux mêmes contraintes que ceux des fichiers et ne peuvent comporter d'extension.
- Seuls huit niveaux de profondeur sont autorisés, c'est à dire que l'arborescence des répertoires ne doit pas présenter plus de 8 niveaux d'imbrication.
- Les fichiers doivent être contigus.

Les règles d'échange de Niveau 2 sont soumises aux mêmes contraires que celles du Niveau 1, mais permettent de spécifier des noms de fichier qui s'étendent jusqu'à 31 caractères. Enfin, les règles de Niveau 3 sont identiques à celles du niveau 2, à ceci près que les fichiers n'ont pas besoin d'être contigus.

Aujourd'hui, dans l'optique contemporaine, où l'industrie a retenu le format UDF en tant que stratégie la plus pertinente au niveau des supports optiques, ISO-9660 est considéré comme un format hérité.

## Joliet

Pensé à l'origine comme un moyen de s'affranchir des restrictions inhérentes à ISO-9660 (notamment en ce qui concerne l'internationalisation), le format Joliet est une extension dudit standard qui permet l'utilisation du jeu de caractères Unicode parmi tous les champs texte du système de fichiers, y compris les noms de répertoire et de fichier, et le nom du volume. Apparu pour la première fois sous Windows 95, Joliet est désormais reconnu et pris en charge par la majorité des systèmes d'exploitation.

La liste qui suit énumère quelques-unes des caractéristiques les plus marquantes en ce qui concerne Joliet.

- Les noms de fichiers ou de répertoire peuvent comprendre jusqu'à 64 caractères Unicode 128 octets).
- Les noms de répertoires peuvent posséder une extension.
- Les répertoires peuvent posséder plus de huit niveaux.

### UDF

UDF (Universal Disk Format) est une spécification de système de fichiers destinée au stockage sur des médias enregistrables. Défini et soutenu par l'OSTA (Optical Storage Technology Association, littéralement : association pour la technologie du stockage optique), le format UDF a comme avantage d'être ouvert, indépendant du système d'exploitation, utilisable sans royalties, et normalisé (ISO/IEC 13346 et ECMA-167).

### FAT12, FAT16 et FAT32

Devenu à force d'adaptation un standard de l'industrie, FAT (*File Allocation Table*) offre de bonnes performances dans des environnements simples, mais n'atteint pas la fiabilité et l'évolutivité de certains systèmes de fichiers modernes. Il est cependant pris en charge par la plupart des systèmes d'exploitation actuels et dans de très nombreux équipements mobiles ou systèmes embarqués. Par considération pour les besoins informatiques actuels (au sens historique du terme) en matière de stockage, FAT a été modernisé trois fois (quatre si l'on considère exFAT). Il n'existe donc pas un unique format de système de fichiers FAT, mais plusieurs dont la différence tient au nombre de bits utilisés pour l'adressage des clusters d'un disque, respectivement 12, 16 et 32 bits. Du point de vue de l'utilisateur, cette diversité se ressent essentiellement au niveau de la taille maximale théorique d'un volume : 16 Mo pour un volume FAT12, 4 Go pour FAT16 et 2 To pour FAT32. Le terme FAT employé sans numéro désigne communément FAT32, soit la dernière itération du format, qui est également la plus répandue.

Initialement conçu pour être utilisé sur des disquettes à une époque où leur capacité n'atteignait pas 1 Mo, FAT12 repose sur un système d'adressage ne pouvant aller au delà de 4096 (2^12) combinaisons. Les tailles de clusters variant ici (comprendre dans ce contexte précis) entre 512 octets et 8 Ko, un volume FAT12 ne peut excéder 32 Mo. Par conséquent, Windows utilise FAT12 pour toutes les disquettes 5 pouces et pour les disquettes 3,5 pouces de 1,44 Mo.

FAT16, qui porte la numérotation des clusters à 16 bits, peut adresser 65 536 (2^16) clusters dont les tailles sont comprises entre 512 octets et 64 Ko. La taille maximale d'un volume FAT16 est de la sorte limitée à 4 Go. FAT32 peut quant à lui théoriquement adresser des volumes de 8 To. (Nous reviendrons plus en détail sur ce point plus loin dans ce chapitre.)

### exFAT

Version améliorée du format FAT traditionnel, Extended FAT, ou exFAT, est un système de fichiers conçu par Microsoft principalement pour les mémoires flash et les supports de stockage externes (disques durs et assimilés), avec comme ambition première d'offrir des fonctionnalités avancées similaires à celles proposées par NTFS, sans toutefois disposer des mêmes travers que son prédécesseur en ce qui concerne la gestion des métadonnées.

Le premier système d'exploitation compatible avec exFAT fut Windows CE 6.0. Le format est désormais reconnu nativement par de nombreux systèmes d'exploitation, tels que Windows depuis la version 7 SP1 et OS X antérieurs à 10.6.5 (Snow Leopard).

## Unité d'allocation

L'unité d'allocation est la plus petite quantité d'espace que le système d'exploitation peut traiter relativement à un système de fichiers - autrement dit le plus petit élément pouvant être lu ou écrit depuis le logiciel vers le matériel.

Chaque unité d'allocation correspond à un ou plusieurs secteurs physiques du disque dur. Dans la pratique, cela revient à dire qu'un fichier peut occuper plusieurs unités d'allocation, mais qu'une unité d'allocation peut contenir une seule portion (éventuellement la totalité) d'un fichier. Il n'existe pas de taille d'unité d'allocation idéale, dans la mesure où cela nécessiterait que toute taille de fichier soit un multiple parfait de la taille d'unité d'allocation.

Pour connaitre la taille d'unité d'allocation en vigueur en vigueur sur un volume, consultez la propriété Octets par secteur rendue visible depuis fsutils, ou la propriété Taille d'unité d'allocation depuis diskpart.

```
C:\WINDOWS\system32>fsutil fsinfo ntfsinfo c:
Numéro de série du volume NTFS :       0xb6c6f47ec6f43fe3
Version NTFS    :                  3.1
Version LFS    :                  2.0
Nombre de secteurs :                  0x000000000e9f97ff
Nombre total de clusters :                 0x0000000001d3f2ff
Clusters libres  :                  0x0000000001514b6a
Total réservé :                  0x000000000000133f
Octets par secteur  :               512
Octets par secteur physique :       512
Octets par cluster :               4096
Octets par segment d'enreg. de fich.    : 1024
Clusters par segment d'enreg. de fich. : 0
Longueur de données valide MFT :           0x0000000023a00000
LCN de démarrage MFT  :                  0x00000000000c0000
LCN de démarrage MFT2 :                  0x0000000000000002
Début de la zone MFT :                  0x000000000058da20
Fin de la zone MFT   :                  0x000000000059a240
Nombre d'étendues de réglage d'appareil max :     256
Nombre d'octets de réglage d'appareil max :       0xffffffff
Nombre d'étendues de réglage de volume max :     62
Nombre d'octets de réglage de volume max :       0x40000000
Identificateur du Gestionnaire de ressources :        25DF3E47-8FF3-11E5-80E2-ECB1D78A5854
```

## Organisation disque de FAT32

Compte tenu des évolutions technologiques auxquelles les systèmes de fichiers ont du s'adapter, FAT a évolué en différentes versions : FAT12, FAT16 et FAT32.

Chaque disque formatée à l'aide de FAT peut être vu comme une suite d'éléments logiques distincts, on y trouve ainsi :

- un enregistrement d'amorçage principal, accompagné de la table des partitions ;
- une zone réservée au secteur de chargement ;
- un exemplaire de la table d'allocation des fichiers ;
- un second exemplaire de la table d'allocation des fichiers, sauvegarde de la première ;
- le répertoire principal (root directory) ou dossier racine ;
- la zone des données et sous répertoires.

Sur un périphérique FAT32, les informations ayant trait aux volumes disque s'étendent sur 32 secteurs, qui dans la configuration par défaut se répartissent ainsi :

- 3 secteurs réservés à l'enregistrement de démarrage du volume (secteurs logiques 0 à 2)
- 3 secteurs réservés, non utilisés et initialisés à zéro (secteurs logiques 3 à 5).
- 3 secteurs occupés par une copie de l'enregistrement de démarrage du volume (secteurs logiques 6 à 8)

- 23 secteurs réservés, non utilisés et initialisés à zéro

## Enregistrement de démarrage

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000000000  EB 58 90 4D 53 44 4F 53 35 2E 30 00 02 20 30 09  ëX.MSDOS5.0.. 0.
000000010  02 00 00 00 00 F8 00 00 3F 00 FF 00 00 08 00 00  .....ø..?.ÿ.....
000000020  00 F8 B6 03 68 3B 00 00 00 00 00 00 02 00 00 00  .ø¶.h;..........
000000030  01 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000040  80 00 29 CF DF EC BC 4E 4F 20 4E 41 4D 45 20 20  €.)ÏßìNO NAME
000000050  20 20 46 41 54 33 32 20 20 20 33 C9 8E D1 BC F4    FAT32   3ÉŽÑ¼ô
000000060  7B 8E C1 8E D9 BD 00 7C 88 56 40 88 4E 02 8A 56  {ŽÁŽÙ½.|ˆV@ˆN.ŠV
000000070  40 B4 41 BB AA 55 CD 13 72 10 81 FB 55 AA 75 0A  @´A»ªUÍ.r..ûUªu.
000000080  F6 C1 01 74 05 FE 46 02 EB 2D 8A 56 40 B4 08 CD  öÁ.t.þF.ë-ŠV@´.Í
000000090  13 73 05 B9 FF FF 8A F1 66 0F B6 C6 40 66 0F B6  .s.¹ÿÿŠñf.¶Æ@f.¶
0000000A0  D1 80 E2 3F F7 E2 86 CD C0 ED 06 41 66 0F B7 C9  Ñ€â?÷â†ÍÀí.Af.·É
0000000B0  66 F7 E1 66 89 46 F8 83 7E 16 00 75 39 83 7E 2A  f÷áf‰Føƒ~..u9ƒ~*
0000000C0  00 77 33 66 8B 46 1C 66 83 C0 0C BB 00 80 B9 01  .w3f‹F.fƒÀ.».€¹.
0000000D0  00 E8 2C 00 E9 A8 03 A1 F8 7D 80 C4 7C 8B F0 AC  .è,.é¨.¡ø}€Ä|‹ð¬
0000000E0  84 C0 74 17 3C FF 74 09 B4 0E BB 07 00 CD 10 EB  „Àt.<ÿt.´.»..Í.ë
0000000F0  EE A1 FA 7D EB E4 A1 7D 80 EB DF 98 CD 16 CD 19  î¡ú}ëä¡}€ëß˜Í.Í.
000000100  66 60 80 7E 02 00 0F 84 20 00 66 6A 00 66 50 06  f`€~...„ .fj.fP.
000000110  53 66 68 10 00 01 00 B4 42 8A 56 40 8B F4 CD 13  Sfh....´BŠV@‹ôÍ.
000000120  66 58 66 58 66 58 66 58 EB 33 66 3B 46 F8 72 03  fXfXfXfXë3f;Før.
000000130  F9 EB 2A 66 33 D2 66 0F B7 4E 18 66 F7 F1 FE C2  ùë*f3Òf.·N.f÷ñþÂ
000000140  8A CA 66 8B D0 66 C1 EA 10 F7 76 1A 86 D6 8A 56  ŠÊf‹ÐfÁê.÷v.†ÖŠV
000000150  40 8A E8 C0 E4 06 0A CC B8 01 02 CD 13 66 61 0F  @ŠèÀä..Ì¸..Í.fa.
000000160  82 74 FF 81 C3 00 02 66 40 49 75 94 C3 42 4F 4F  ‚tÿ.Ã..f@Iu”ÃBOO
000000170  54 4D 47 52 20 20 20 20 00 00 00 00 00 00 00 00  TMGR    ........
000000180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000001A0  00 00 00 00 00 00 00 00 00 00 00 00 0D 0A 45 72  ..............Er
0000001B0  72 2E 20 64 69 73 71 75 65 FF 0D 0A 50 72 65 73  r. disqueÿ..Pres
0000001C0  73 65 7A 20 75 6E 65 20 74 6F 75 63 68 65 20 70  sez une touche p
0000001D0  6F 75 72 20 72 65 64 82 6D 61 72 72 65 72 0D 0A  our red‚marrer..
0000001E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000001F0  00 00 00 00 00 00 00 00 AC 01 BA 01 00 00 55 AA  ........¬.º...Uª
```

| Position(h) | Taille(o) | Attribut |
|---|---|---|
| 000 | 3 | Instruction de saut |
| 003 | 8 | Fabricant d’origine |
| 00B | 2 | Octets par secteur |
| 00D | 1 | Secteurs par cluster |
| 00E | 2 | Nombre de secteurs réservés |
| 010 | 1 | Nombre de tables d’allocation (FAT) |
| 011 | 2 | Compteur d’entrées de répertoire (FAT12 et FAT16 seulement) |
| 013 | 2 | Compteur de secteurs sur le volume (FAT12 et FAT16 seulement) |
| 015 | 1 | Type de media |
| 016 | 2 | Nombre de secteurs par FAT |
| 018 | 2 | Secteurs par piste |
| 01A | 2 | Compteur de têtes |
| 01C | 2 | Compteur de secteurs cachés |
| 020 | 4 | Nombre de secteurs dans le volume |
| 024 | 4 | Secteurs par table |
| 028 | 2 | Attributs du disque. |
| 02A | 2 | Version |

| | | |
|---|---|---|
| 02C | 4 | Premier cluster (racine) |
| 030 | 2 | Premier cluster (FSInfo) |
| 032 | 2 | Premier cluster (copie de sauvegarde) |
| 034 | 12 | Réservé pour des ajouts ultérieurs |
| 040 | 1 | Numéro de disque physique |
| 041 | 1 | Réservé pour un usage ultérieur |
| 042 | 1 | Signature étendue |
| 043 | 4 | Numéro de série du volume |
| 047 | 11 | Nom du disque |
| 052 | 8 | Type de système |
| 1FE | 2 | Marqueur de fin de secteur |

- *Instruction de saut* Branchement inconditionnel menant à l'adresse de départ du code de démarrage.
- *Nom du fabricant* Nom du système ou composant à partir duquel le disque a été formaté. Les environnements Microsoft Windows ne tiennent pas compte de ce champ. Quelques pilotes FAT inclus à d'autres systèmes d'exploitation le font. Exemple bloc (3-A) : 4D 53 44 4F 53 35 2E 30, ce qui tend à montrer que le formatage a eu lieu sur un système "MSDOS5.0" - à titre informatif, la procédure s'est effectuée sur Windows 10.
- *Octets par secteur* Nombre d'octets par secteur. Les valeurs admises à cet égard sont 512, 1 024, 2 048 et 4 096.
- *Secteurs par cluster* Nombre de secteurs par unité d'allocation (cluster). Les valeurs admises à cet égard sont 1, 2, 4, 8, 16, 32, 64 et 128.
- *Nombre de secteurs réservés* Nombre de secteurs réservés, correspondant au nombre de secteurs qui précédent la première table d'allocation des fichiers.
- *Nombre de tables d'allocation* Nombre de tables d'allocation que le volume pérennise. Une large majorité des systèmes compatibles FAT32 gèrent deux tables : l'une utilisée dans le cadre du fonctionnement normal, l'autre comme copie de sauvegarde de la première.
- *Type de media* Catégorie à laquelle s'apparente le périphérique. 0xF8 pour les supports non amovibles, 0xF0, 0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE ou 0xFF pour les autres. Une telle valeur, qui remonte à l'époque des premiers systèmes MS-DOS, n'est présentement plus utilisée pour quoi que ce soit.
- *Secteurs par piste* Nombre de secteurs par piste. Une telle valeur, pertinente uniquement dans ce qu'elle révèle de la géométrie du périphérique, est employée en tant que paramètres CHS auprès de l'interruption 0x13, en charge des accès disque.
- *Compteur de têtes* Nombre de têtes. Une telle valeur, pertinente uniquement dans ce qu'elle révèle de la géométrie du périphérique, est employée en tant que paramètres CHS auprès de l'interruption 0x13, en charge des accès disque.
- *Compteur de secteurs cachés* Nombre de secteurs présents sur le volume avant le secteur de démarrage proprement dit.
- *Nombre de secteurs du volume* Taille du volume, comptabilisée en nombre de secteurs.
- *Secteurs par table* Nombre de secteurs qu'occupe sur disque une table d'allocation des fichiers. Windows emploie cette valeur, couplée au nombre de tables définies sur le volume et au nombre de secteurs réservés, en vue de déterminer où loge sur disque le répertoire racine.
- *Version* Numéro majeur et mineur de la version de FAT32. Une telle numérotation, conçue dans un but d'extensibilité, prend en charge la possibilité d'étendre le niveau de support qu'un composant ou système FAT32 est susceptible de fournir.

- *Premier cluster (racine)* Numéro du premier cluster où réside le répertoire racine.
- *Premier cluster (FSInfo)* Numéro du premier cluster où réside la structure d'information de système de fichiers
- *Premier cluster (copie de sauvegarde)* Numéro du premier cluster où réside la copie de sauvegarde du secteur de démarrage.
- *Numéro de disque physique* Numéro du disque tel qu'exigé par l'interruption BIOS 0x13. Une telle valeur n'est pertinente que si le disque est un périphérique de démarrage
- *Signature étendue* 0x29 par défaut
- *Numéro de série du volume* Numéro de série aléatoire assigné au volume au moment de son formatage.
- *Nom du disque* Nom du disque sur 11 caractères ('NO NAME' si pas de nom).

## FSInfo

Les informations au travers desquelles FAT32 gère fichiers et répertoires grandissent au même rythme que ces derniers occupent le disque. De ce fait, pour optimiser le fonctionnement du système, l'enregistrement de démarrage de chaque partition FAT32 renferme une structure spéciale, dite FSInfo, visant à court-circuiter autant que possible les recherches sur les clusters, soulageant ainsi l'ordinateur de coûteuses opérations de calcul.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000000200  52 52 61 41 00 00 00 00 00 00 00 00 00 00 00 00  RRaA............
000000210  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000220  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000230  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000240  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000250  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000260  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000270  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000280  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000290  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000002A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000002B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000002C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000002D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000002E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000002F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000300  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000310  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000320  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000330  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000340  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000350  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000360  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000370  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000380  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000390  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000003A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000003B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000003C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000003D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000003E0  00 00 00 00 72 72 41 61 BB B3 1D 00 07 00 00 00  ....rrAa»³......
0000003F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA  ..............Uª
```

Le numéro du secteur où réside la structure FSInfo est enregistrée à la position 0x30 du secteur d'amorçage du disque concerné, attribut dont la valeur est usuellement 1, correspondant au second secteur du volume.

| Position(h) | Taille(o) | Attribut |
|---|---|---|
| 000 | 4 | Signature |

| Position(h) | Taille(o) | Attribut |
|---|---|---|
| 004 | 480 | Réservé1 |
| 1e4 | 4 | Signature2 |
| 1e8 | 4 | Nombre de clusters libres |
| 1ec | 4 | Premier cluster libre |
| 1f0 | 12 | Réservé2 |
| 1fc | 4 | Signature3 |

- *Signature* Séquence d'octets dont la valeur (0x41615252) permet d'identifier sans ambiguïté une structure FsInfo.
- *Réservé1, Réservé2* Octets réservés pour une extension future.
- *Signature2* Séquence d'octets dont la valeur (0x61417272) permet d'identifier sans ambiguïté une structure FsInfo.
- *Nombre de clusters libres* Comptabilise le nombre de clusters libres sur le volume. Une valeur 0xFFFFFFFF pour cet attribut signifie que ledit nombre est inconnu, et doit par conséquent faire l'objet d'un calcul.
- *Premier cluster libre* Numéro du premier cluster disponible en vue d'y apporter de nouvelles données, ce qui permet à Windows (ou tout autre système d'exploitation reconnaissant FAT32) de ne pas perdre de temps à explorer tous les clusters à la recherche d'un immédiatement utilisable. Une telle valeur correspondant en règle générale au numéro du dernier cluster ayant été alloué pour stocker un fichier. Une valeur 0xFFFFFFFF signifie que l'opération doit s'effectuer depuis le tout début de la table FAT, c'est à dire à partir du second cluster.

## Répertoire racine

***Tableau 17.1. Descriptif d'une entrée du répertoire racine***

| Position(h) | Taille(o) | Attribut |
|---|---|---|
| 00 | 8 | Nom de fichier |
| 08 | 3 | Extension |
| 0B | 1 | Attributs du fichier |
| 0C | 1 | Réservé |
| 0E | 2 | Heure de création |
| 10 | 2 | Date de création |
| 12 | 2 | Dernier accès |
| 14 | 2 | Premier cluster (2 octets de poids fort) |
| 16 | 2 | Heure de modification |
| 18 | 2 | Date de modification |
| 1A | 2 | Premier cluster (2 octets de poids faible) |
| 1C | 4 | Taille du fichier |

- *Nom de fichier* Nom par lequel le fichier est enraciné dans le système de fichiers, complété s'il y a lieu à 8 caractères (avec des espaces). Selon sa valeur l'octet enregistré en première position de cet attribut a l'une ou l'autre des significations que voici.

| Masque | Signification |
|---|---|
| 0x00 | L'entrée est disponible et aucune entrée ultérieure n'est utilisée. |
| 0xE5 | L'entrée a été utilisée mais le fichier a été effacé ; elle est donc disponible. |
| 0x2E | |

- *Extension* Extension du nom en texte, complété s'il y a lieu à 8 caractères (avec des espaces).

- *Attributs* Ensemble de caractéristiques auxquelles le fichier est subordonné : lecture seule, caché, système, et ainsi de suite.

| Masque | Signification |
|---|---|
| 0x01 | Lecture seule. Toute tentative de modifier le fichier se solde par un échec. |
| 0x02 | Fichier caché, exclu par conséquent des recherches normales. |
| 0x04 | Fichier système. Indique que le fichier fait partie du fonctionnemnt normal du système d'exploitation, et ne peut dès lors pas être déplacé physiquement (par exemple lors d'une défragmentation). |
| 0x08 | Étiquette de volume. |
| 0x10 | Sous-répertoire. Indique que la chaîne de cluster associée à cette entrée doit etre interprétée comme un sous--répertoire (plutot qu'un fichier). Les sous-répertoires ont une entrée de taille de fichier de zéro. |
| 0x20 | Fichier archive. L'option est utilisée pour indiquer si le fichier a été sauvegardé (archivé) ou non. |

- *Heure de création* Heure à laquelle le fichier a été créé, enregistrée au format DOS.
- *Date de création* Date à laquelle le fichier a été créé, enregistrée au format DOS.
- *Dernier accès* Date à laquelle le fichier a été accédé pour la dernière fois, enregistrée au format DOS. Dans la configuration par défaut, Windows enregistre la date de dernier accès de tous les fichiers que vous utilisez, y compris quand aucune modification n'a eu lieu.
- *Heure de modification* Heure à laquelle le fichier a été modifié pour la dernière fois, enregistrée au format DOS.(Dans la perspective du système de fichiers, la création d'un fichier est considérée comme une opération d'écriture, et partant une modification.)
- *Date de modification* Date à laquelle le fichier a été modifié pour la dernière fois, enregistrée au format DOS. (Dans la perspective du système de fichiers, la création d'un fichier est considérée comme une opération d'écriture, et partant une modification.)
- *Premier cluster* Numéro du premier cluster où réside le fichier.
- *Extension* Extension du nom en texte, complété s'il y a lieu à 3 caractères (avec des espaces).
- *Taille du fichier* Nombre d'octets que le fichier occupe.

## Sauvegarde de l'environnement d'amorçage

Pour éviter que les donnée de démarrage du volume ne constituent un point de défaillance unique, FAT32 gère un duplicata des trois premiers secteurs du disque, incluant l'enregistrement de démarrage et la structure FSInfo. Ainsi, dans l'éventualité où les informations de ces secteurs viendraient à être corrompues, une procédure de restauration peut facilement être menée à partir de la copie de sauvegarde. Le numéro du secteur où réside la copie de sauvegarde est enregistrée à la position 0x32 du secteur d'amorçage du disque concerné.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000000030  .. .. 06 00 .. .. .. .. .. .. .. .. .. .. .. ..  ................
.
.
.
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000000C00  EB 58 90 4D 53 44 4F 53 35 2E 30 00 02 20 30 09  ëX.MSDOS5.0.. 0.
000000C10  02 00 00 00 00 F8 00 00 3F 00 FF 00 00 08 00 00  .....ø..?.ÿ.....
000000C20  00 F8 B6 03 68 3B 00 00 00 00 00 00 02 00 00 00  .ø¶.h;..........
000000C30  01 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000C40  80 00 29 CF DF EC BC 4E 4F 20 4E 41 4D 45 20 20  €.)ÏßìNO NAME
.
.
.
```

# NTFS

## Noms Unicode

Tout comme Windows dans son ensemble, NTFS est totalement orienté Unicode, et emploie à ce titre ledit jeu de caractères pour l'ensemble des objets représentés, y compris les noms de fichiers, de répertoires et de volumes.

Les systèmes classiques de représentation des caractères utilisent un schéma à deux octets dans lequel certains caractères tiennent sur 8 bits et d'autres sur 16 bits, technique qui exige le chargement de diverses pages de codes pour créer les caractères disponibles. Unicode ayant de son coté une représentation unique associée à chaque caractère, il ne dépend pas de telle ou telle page de code chargée. Chaque nom de répertoire ou de fichier dans un chemin NTFS peut contenir jusqu'à 255 caractères Unicode.

## Versions de NTFS

Il y a eu plusieurs versions de NTFS. La première était NTFS 1.0 avec Windows NT 3.1 (1993). Les secondes et troisièmes versions (1.1 et 1.2) furent distribuées par l'intermédiaire, respectivement, de Windows NT 3.1 (1993) et Windows NT 4.0 (1996). Windows 2000 fit entrer la prise en charge de NTFS 3.0, et Windows XP (2001) de NTFS 3.1. Bien que les versions ultérieures de Windows aient ajouté de nouvelles fonctionnalités liées au système de fichiers, elles n'entraînèrent pas d'amendement de la fiche technique de NTFS.

Pour consulter les informations de version relatives à NTFS, utilisez la commande fsutil fsinfo ntfsinfo.

[subs="quotes"]() C:\>fsutil fsinfo ntfsinfo c: Numéro de série du volume NTFS : 0xb6c6f47ec6f43fe3 **Version NTFS : 3.1** Version LFS : 2.0 . . .

## Appropriation de fichier/dossier

Tout fichier ou dossier se voit dans la perspective NTFS rattaché à un détenteur (nommé en la circonstance propriétaire). Le propriétaire d'une ressource correspond par défaut au compte créateur de ladite ressource. Lorsqu'un utilisateur crée un fichier ou un dossier, il en devient de facto le propriétaire, ce qui laisse alors la possibilité de définir les autorisations appropriées pour que d'autres utilisateurs puissent y accéder.

Un utilisateur ne peut s'approprier une ressource qu'à condition de posséder l'autorisation spéciale Prendre possession. La notion de propriété à laquelle ramène un objet ne peut en outre pas être donnée, elle ne peut qu'être prise. Les administrateurs ont le pouvoir de prendre possession de pratiquement tout les objets, sauf ceux gérés par le système.

## Autorisations NTFS

Un aspect important du rôle imparti à tout administrateur consiste à fournir aux utilisateurs un moyen commode d'afficher et manipuler les fichiers et dossiers de l'ordinateur, tout en protégeant ceux-ci de tiers illégitimes. Dans la perspective NTFS, ces enjeux se voient concrétisés par l'intermédiaire de propriétés dites *autorisations*, lesquelles régissent l'accès aux données stockées sur le système de fichiers, et assurent in fine un contrôle optimisé des ressources.

Pour configurer les autorisations NTFS concernant un volume, un répertoire ou fichier, au moins l'une des conditions suivantes doit être satisfaite : (1) faire partie du groupe Administrateurs, (2) être le propriétaire de la ressource (2), ou (3) bénéficier de l'autorisation Contrôle total sur celle-ci.

Les Autorisations NTFS définissent les droits dont un utilisateur est titulaire vis à de ressources disque, que celles-ci soit accédé localement (C:\MaRessource) ou à partir du réseau (\\NomDeServeur\MaRessource). Corollaire de ce principe, elles doivent être appliqués explicitement à un fichier ou un répertoire afin d'en

accorder l'accès à un utilisateur. Autrement dit, si aucune autorisation sur un fichier n'a été concédée pour un utilisateur donné ou pour le groupe auquel il appartient, l'utilisateur n'a pas accès à ce fichier.

Sur un volume NTFS, tous les fichiers et dossiers ont une liste de contrôle d'accès discrétionnaire qui leur est associée. La DACL répertorie les comptes d'utilisateur et les groupes auxquels ont été attribués des autorisations d'accès à une ressource, ainsi que les autorisations elle-même. Chaque entrée au sein d'une DACL est appelée une entrée de contrôle d'accès (ACE). Un compte d'utilisateur, ou un groupe dont le compte d'utilisateur est membre, doit être défini en tant qu'entrée de contrôle d'accès dans la DACL pour envisager avoir accès à une ressource.

La liste qui suit énumère les autorisations de base en ce qui concerne des dossiers.

- *Affichage du contenu du dossier* L'utilisateur est en mesure de voir les noms des fichiers et des sous-répertoires du dossier, mais ne peut y accéder.
- *Lecture* L'utilisateur peut voir les fichiers et sous-répertoires du dossier, et afficher les propriétés qui s'y rapportent, telles que les autorisations, le propriétaire et les attributs.
- *Écriture* L'utilisateur peut créer de nouveaux fichiers et répertoires au sein du dossier, modifier les attributs du dossier, afficher les autorisations et le propriétaire du dossier.
- *Lecture et exécution* L'utilisateur peut effectuer toutes les actions permises par les autorisations Lecture et Affichage du contenu du dossier, et peut également parcourir les sous-dossiers.
- *Modification* L'utilisateur peut effectuer toutes les actions permises par les autorisations Écriture et Lecture et exécution, et également supprimer le répertoire.
- *Contrôle total* L'utilisateur peut effectuer toutes les actions permises par les autres autorisations de base, mais aussi modifier les autorisations et prendre possession (devenir propriétaire) du dossier.

Par défaut, la DACL d'un fichier hérite des mêmes entrées que le répertoire dans lequel il se situe (dossier parent). Bien que les autorisations concernant fichiers et dossiers s'appuient sur des principes, et ont des des réalités de fonctionnement, par nature très proches, il existe néanmoins de subtiles différences entre les deux. La liste qui suit répertorie les autorisations de base pour des des fichiers.

- *Lecture* L'utilisateur a la possibilité de lire le contenu du fichier et d'afficher ses propriétés, y compris ses autorisations, son propriétaire et ses attributs.
- *Écriture* L'utilisateur peut écraser le fichier (mais pas modifier son contenu), modifier ses attributs, son propriétaire et ses autorisations.
- *Lecture et exécution* L'utilisateur peut effectuer toutes les actions permises par l'autorisation Lecture et exécuter le ficher.
- *Modification* L'utilisateur peut effectuer toutes les actions permises par les autorisations Écriture, Lecture et exécution, modifier le contenu du fichier et supprimer ce dernier.
- *Contrôle total* L'utilisateur peut effectuer toutes les actions permises par les autres autorisations de base, mais aussi modifier les autorisations et le propriétaire du fichier.

Les autorisations sur un fichier sont toujours prioritaires sur celles qui s'appliquent au dossier qui le contient. Par exemple, si un utilisateur dispose de l'autorisation Lecture sur un dossier et de l'autorisation Modification sur un fichier stocké à cet emplacement, il peut dans tous les cas apporter des modifications au fichier et les enregistrer.

### Affichage des autorisations NTFS attribuées

Sur un volume NTFS, il est possible de voir quelles autorisations s'appliquent en faisant appel à l'onglet Sécurité de la boîte de dialogue Propriétés d'un fichier ou d'un répertoire.

1. Dans l'Explorateur Windows, naviguez jusqu'au fichier ou au dossier pour lequel vous souhaitez connaitre les autorisations.

2. Effectuez un clic secondaire sur le fichier ou le dossier en question puis cliquez sur Propriétés dans le menu contextuel.

3. Cliquez sur l'onglet Sécurité.

Lorsque vous affichez les attributions des autorisations, Windows identifie généralement les comptes d'utilisateur et les groupes par leur nom, mais s'il ne peut procéder de la sorte (tel ou tel nom n'a pu être résolu), montre alors le SID de l'entité concernée.

### Copie et déplacement de fichiers et de dossiers

Pour effectuer une opération de copie ou de déplacement vers une partition NTFS, l'utilisateur doit disposer des autorisations adéquates. Par exemple, le déplacement d'un fichier entre partitions NTFS n'est possible que si l'utilisateur dispose du droit Écriture dans le dossier de destination et de l'autorisation Modifier dans le dossier source. (Le droit de modification est requis du fait que, lors du déplacement d'un fichier ou d'un dossier, Windows procède à la suppression du fichier ou du dossier après l'avoir copié à l'emplacement cible.)

Les opérations de copie/déplacement de fichiers/dossiers ont potentiellement un effet sur les autorisations. Il est donc important de comprendre les modifications qui se produisent alors, et de connaître les quelques règles qui prévalent en la matière.

- Si vous copiez un fichier ou un dossier vers le même volume ou sur un volume différent, la copie du fichier ou du dossier hérite des autorisations de la destination.

- Si vous déplacez un fichier ou un dossier vers un volume différent, ce dernier hérite des autorisations de la destination. Par contre, si vous déplacez un fichier ou dossier sur le même volume, il y a conservation des autorisations d'origine.

- Que ce soit pour des copies ou des déplacements, si le volume destination n'est pas une partition NTFS, toutes les autorisations sur le fichier ou le dossier copié sont perdues, cela du fait que les partitions non-NTFS ne prennent pas en charge les autorisations NTFS.

Lorsque vous copiez un fichier ou un dossier, vous devenez propriétaire de la copie qui résulte de ces opérations.

## Indexation

L'architecture NTFS est ainsi faite qu'elle prend en charge l'indexation tant du contenu que des propriétés des fichiers des fichiers, incluant nom, chemin, taille, type, date de modification. Une telle exhaustivité permet au système de fichiers de trouver plus rapidement les fichiers qui répondent à certains critères, par exemple tous les fichiers audio d'un répertoire, tous les fichiers contenant une chaine de caractère donnée, etc.).

La plupart des application intégrées à Windows tirent partie, d'une façon ou d'une autre, des catalogues issues de l'indexation. L'Explorateur, par exemple, l'utilise pour suivre les modifications apportés aux fichiers. Microsoft Edge l'emploie en vue de présenter rapidement des résultats pour l'historique du navigateur dans la barre d'adresses.

Pour savoir quels répertoires sont considérés pour l'indexation, allez dans le panneau de configuration et cliquez sur Options d'indexation. La boîte de dialogue homonyme s'affiche alors. Les éléments indexés sont répertoriés dans la partie centrale de la fenêtre.

**Indexer des volumes.** Sauf spécification particulière, seuls les dossiers utilisateurs et le menu Démarrer sont pris en compte pour l'indexation, ce dans l'optique de privilégier les performances. Pour indexer d'autres emplacements, suivez les étapes décrites ci-après.

1. Dans le panneau de configuration, double-cliquez sur l'icône Options d'indexation.

2. Dans la fenêtre Options d'indexation, cliquez sur le bouton Modifier.

3. Dans la fenêtre Emplacements indexés, sélectionnez tout ou partie d'un élément à indexer, puis validez en cliquant sur le bouton OK.

**Indexer le contenu des fichiers.** Dans la configuration par défaut, Windows n'indexe (pour les emplacements indexés) que les noms et les attributs (propriétés) des fichiers, à l'exclusion, donc, de leur contenu. Pour indexer le contenu des fichiers, procédez comme suit :

1. Dans le panneau de configuration, double-cliquez sur l'icône Options d'indexation.

2. Dans la fenêtre Options d'indexation, cliquez sur le bouton Avancé.

3. Dans la fenêtre Options avancées, cliquez sur l'onglet Types de fichiers.

4. Dans la rubrique Comment ce fichier doit-il être indexé, optez pour l'option Indexer les propriétés et le contenu des fichiers, puis validez en cliquant sur le bouton OK.

## Compression NTFS

NTFS prend en charge de sorte à optimiser l'espace disque la compression de données fichier. Les procédures établies en la matière s'exécutant de manière transparente, les fichiers compressés peuvent être manipulés sans avoir à être décompressés via un programme externe. La décompression et la compression sont effectuées, respectivement, lors de la lecture ou de l'enregistrement d'un fichier et n'entraînent presque aucun coût supplémentaire.

Certains types de fichier peuvent être fortement compressés, conduisant de la sorte à des gains significatifs en terme d'espace disque. La compression d'autres types de fichier n'a, par contraste, que très peu d'effet.

Les applications agissent sur l'état de compression d'un fichier ou d'un répertoire en alimentant la fonction DeviceIoControl avec le code de contrôle FSCTL_SET_COMPRESSION, et consultent le même état à l'aide du code FSCTL_GET_COMPRESSION. Un fichier compressé est doté de l'attribut FILE_ATTRIBUTE_COMPRESSED, qui peut être consulté via la fonction GetFileAttributes.

### Compression en ligne de commandes

L'utilitaire intégré compact.exe permet de gérer la compression à l'invite de commandes.

- */C* Compresse le répertoire ou le fichier spécifié. Les répertoires seront marqués de telle sorte que tous les fichiers ajoutés par la suite soient compressés, sauf si le paramètre /EXE est spécifié.

- */U* Décompresse le répertoire ou le fichier spécifié. Les répertoires seront marqués de telle sorte que tous les fichiers ajoutés par la suite ne soient pas compressés. Dans l'éventualité où le paramètre /EXE est spécifié, seuls les fichiers compressés en tant qu'exécutables seront décompressés ; autrement, seuls les fichiers compressés NTFS le seront.

- */?* Affiche l'aide à l'invite de commandes.

- */A* Affiche les fichiers cachés ou système, qui ne sont Par défaut pas inclus.

- */I* Poursuit l'exécution de l'opération spécifiée même après que des erreurs aient été constaté. Par défaut, la commande compact s'arrête sitôt rencontrée une erreur.

- */Q* Ne reporte que les informations essentielles.

- */F* Force l'opération de compression sur tous les fichiers spécifiés, y compris ceux déjà compressés (qui sont ignorés par défaut).
- */EXE* Utilise la compression optimisée pour les fichiers exécutables. Les algorithmes pris en charge sont : XPRESS4K (valeur la plus rapide, et aussi celle par défaut), XPRESS8K, XPRESS16K et LZX (le plus compact).
- */CompactOs* Définit ou interroge l'état de compression du système.
- */WINDIR* Utilisé conjointement au paramètre /CompactOs:query lors de l'interrogation du système d'exploitation hors connexion. Spécifie le répertoire dans lequel Windows est installé.

En cas d'utilisation sans paramètre, compact affiche l'état de compression du répertoire actif et de tous les fichiers qu'il contient.

## Flots de données multiples

Dans NTFS, chaque chaque unité d'informations associée à un fichier (nom, propriétaire, données d'horodatages, contenu, etc.) est enracinée dans le système de fichiers sous forme d'attribut. Chaque attribut se compose d'un seul flot, en l'occurence une simple séquence d'octets. Les fichiers (et répertoires) peuvent contenir de multiples flots.

Un fichier NTFS a un seul flot de données par défaut, dépourvu de nom. Une application peut créer des flots de données supplémentaires et y accéder par leurs noms.

L'utilisation la plus répandue des flots de données multiples concerne les métadonnées. Ainsi, si vous naviguez à l'aide de l'Explorateur Windows parmi les propriétés d'un fichier, l'application vous permet d'associer au fichier des informations du genre titre, auteur, mots clé, etc. En interne, l'Explorateur (par extension le système d'exploitation) mémorise ces informations par le biais d'un flot spécifique, nommé Summary Information, qu'il ajoute au fichier.

Pour mieux compartimenter l'utilisation des flots de données multiples, chaque flot est doté de caractéristiques distinctes du fichier auquel il s'apparente, et a l'occurence une taille d'allocation (déterminant combien d'espace disque lui a été octroyé) et une taille réelle (comptabilisant le nombre d'octets qu'il occupe) qui lui sont propre. Chaque flot incorpore en plus de cela des sémantiques de verrouillage et de synchronisation, indispensables pour la prise en charge des accès concurrents.

Seule une minorité de commandes et de logiciels intégrés avec Windows sont conçus pour manipuler des flots nommés, dont echo et more.

## Remappage dynamique des clusters défectueux

Sur les systèmes dépourvues de fonctionnalités de tolérance aux pannes, toute tentative de manipuler des informations depuis un secteur disque défectueux se solde invariablement par la perte des données concernées - avec pour conséquence un comportement erratique de l'ordinateur. NTFS emploie de sorte à anticiper de tels genres de dysfonctionnement une méthode dite de *remappage des clusters*. Ainsi, en cas d'erreur de secteur défectueux, NTFS alloue automatiquement un nouveau cluster, qui remplace le cluster dans lequel réside le secteur problématique, et copie les données dans le cluster nouvellement créé. NTFS enregistre l'adresse des secteurs illisibles dans le fichier $Badclus. Ainsi, en cas d'erreur de secteur défectueux, NTFS alloue automatiquement un nouveau cluster, qui remplace le cluster dans lequel réside le secteur problématique, et copie les données dans le cluster nouvellement créé. NTFS enregistre l'adresse des secteurs illisibles dans le fichier $Badclus.

Bien que ne constituant pas une solution parfaite, le remappage dynamique des secteurs défectueux permet de réduire drastiquement les nuisances que peuvent occasionner des emplacements disque subitement inaccessibles ou impossible à écrire. Quand des secteurs défectueux sont détectés lors d'une opération d'une

E/S type lecture, NTFS tente de procéder aux ajustements nécessaires et, si le volume n'est pas configuré en tant que volume redondant, renvoie une erreur de lecture au programme appelant. Dans ce cas de figure, si les données qui se trouvaient dans ce cluster sont bel et bien perdues, le reste du fichier - de même que le système de fichiers - demeurent intact; charge au logiciel dont émane l'opération d'E/S de répondre de manière appropriée à la carence de données. Dans l'éventualité où l'erreur s'est produite suite à une opération d'écriture, NTFS écrit les données dans le nouveau cluster et aucune donnée n'est perdue.

## Fichiers clairsemés

En plus des méthodes classiques de compression et décompression des données, NTFS offre un autre type de mécanisme via lequel réduire l'encombrement global du stockage : les fichiers clairsemés (*sparse files*).

Quand un fichier est marqué comme clairsemé, NTFS n'alloue d'espace de volume qu'aux portions du fichier qui contiennent des données, et catalogue les autres portions (comprendre les zones vides d'un fichier) à l'aide de métadonnées. Lorsque de tels fichiers sont lus, les données non allouées sont retournées sous la forme de tampons remplis de zéro, conformément aux exigences en matière de sécurité.

En plus d'utiliser plus efficacement le disque, les fichiers clairsemés permettent de créer des fichiers volumineux même s'il n'existe pas suffisamment d'espace libre disponible pour ce faire. Cela est par exemple utile dans les systèmes où une stratégie de pré-allocation est souvent la règle (par exemple, création d'une image de machine virtuelle).

A l'instar des fichiers comprimés, les fichiers clairsemés sont gérés de façon transparente par NTFS. Les application indiquent qu'un fichier est clairsemé en spécifiant auprès de la fonction DeviceIoControl le code de contrôle FSCTL_SET_SPARSE. Elles utilisent le code FSCTL_SET_ZERO_DATA en vue de marquer une plage d'un fichier comme étant vide, et le code FSCTL_QUERY_ALLOCATED_RANGES afin de demander quelles parties d'un fichier le sont.

## Compression ZIP

Windows intègre une gestion native des archives au format ZIP, offrant de la sorte aux utilisateurs la possibilité de créer et manipuler de tels fichiers sans qu'il est besoin de recourir à des outils tiers. (S'agissant d'une norme, les fichiers compressés par Windows sont compatibles avec les autres logiciels utilisant le format ZIP.) Moins souple, mais pour autant pas moins prisée, que la compression NTFS, la compression ZIP a pour avantage une indépendance naturelle aux systèmes de fichier.

S'offrant de façon transparente à l'utilisateur, les fichiers portant l'extension ZIP sont dans l'Explorateur Windows considérés comme des dossiers (à l'instar des fichiers CAB). De ce fait, une large majorité des opérations valides dans le contexte de répertoires le sont également pour des fichiers ZIP : ouverture, copie, découpage, suppression, etc.

Pour créer un nouveau fichier compressé à partir de l'EXplorateur, sélectionnez le(s) fichier(s) ou dossier(s) à compressez puis, après un clic secondaire, choisissez Envoyer vers, Dossier compressé. Le fichier résultant adopte le nom du premier fichier ou dossier sélectionné, auquel est ajouté l'extension .ZIP. Les paramètres qui régissent la compression ne peuvent être personnalisés.

## Liens durs

Un lien dur permet à de multiples chemins de référencer les mêmes données disque, ouvrant de la sorte la possibilité à un fichier unique d'apparaitre en plusieurs endroits d'un volume NTFS.

Quel que soit le nombre de liens durs établis concernant un fichier, les entrées de répertoire associées pointent toutes vers le fichier unique enraciné dans le système de fichiers. En cela, essentiellement, diffèrent les liens durs des copies. Les liens durs ne peuvent correspondre qu'à des données existantes *et* accessibles à partir du même volume.

Les processus ont la possibilité de créer des liens durs avec la fonction CreateHardLink, les administrateurs par le biais des commandes fsutil et mklink. Avant de voir comment ces deux utilitaires prennent en charge les interactions avec des liens durs, commencez par créer un fichier et ajoutez-y du texte, soit à l'aide d'un logiciel spécifiquement adapté à cet usage, soit (comme ici) d'une fonction de redirection d'E/S.

```
C:\>echo "lorem ipsum" > lorem.txt
```

Sollicitez la commande mklink en spécifiant le commutateur /H.

```
C:\>mklink /H hardlink.txt lorem.txt
Liaison permanente créée pour hardlink.txt <<===>> lorem.txt
```

Si vous inventoriez le contenu du répertoire, vous pourrez remarquer que les deux fichiers sont identiques en tous points, et partagent en l'occurence les mêmes attributs (date de création, autorisations, etc.) ; seul leur nom, en définitive, diffèrent.

```
C:\>dir *.txt
 Le volume dans le lecteur C s'appelle Windows
 Le numéro de série du volume est D659-86F7

 Répertoire de C:\

2021-01-20  14:46                16 hardlink.txt
2021-01-20  14:46                16 lorem.txt
               2 fichier(s)               32 octets
               0 Rép(s)  142 915 411 968 octets libres
```

## Organisation de NTFS

Cette section décrit comment Windows régente et enracine sur le disque les différentes structures impliquées dans NTFS.

### Volumes

Figure centrale de toute organisation de stockage moderne, la notion de volume se traduit concrètement sous la forme d'une partition logique munie d'un système de fichiers, déterminé au formatage de tout ou partie du disque. (Une partition qui existe sans système de fichier n'entre donc pas dans cette définition.)

Un disque peut avoir un plusieurs volumes, chacun comprenant une série de fichiers, plus l'espace libre restant sur la partition.

### Clusters

Comme d'autres systèmes de fichier, NTFS gère l'espace disque à l'aide de *clusters*, correspondant à des ensembles de secteurs physiques. La taille de cluster sur un volume NTFS, ou facteur de cluster, est définie lors de la phase de formatage, et dépend de la taille du volume. Le nombre de secteurs dans un cluster est toujours une puissance de 2 (1 secteur, 2 secteurs, 4 secteurs, etc.). La taille du cluster est exprimée en octets.

NTFS utilise le cluster comme unité d'allocation essentiellement dans l'optique de garantir son indépendance vis-à-vis des tailles de secteurs physiques, ce qui lui permet de gérer efficacement de très gros disques (via une taille de cluster optimale), et à tout le moins, prendre en charge des disques qui, s'éloignant des standards, se basent sur une taille de secteur différente de 512 octets. Sur un volume contenant des fichiers de grande taille, l'emploi de clusters plus gros contribue à réduire la fragmentation et accélérer les commandes d'allocation, ce qui offre en définitive de meilleurs performances.

NTFS référence les emplacements physiques d'un disque au moyen de numéros de clusters logiques (LCN, Logical Cluster Number), lesquels servent à numéroter l'un après l'autre l'ensemble des clusters appartenant à

un volume (en partant du début vers la fin). Le LCN numéro 0 fait référence au premier cluster du volume (à quoi correspond, du reste, le secteur de démarrage), le LCN numéro 1 au second, et ainsi de suite. Pour convertir un LCN en adresse disque physique, NTFS multiplie le LCN par la taille de cluster en vigueur sur le volume.

Là où les LCN numérotent les clusters d'un volume, les numéros de clusters virtuels (LCN, Virtual Cluster Number) le font des clusters appartenant à un fichier particulier.

### Master File Table (MFT)

Dans la perspective NTFS, toutes les informations stockées sur le disque le sont par l'intermédiaire de fichiers, y compris les structures de données internes permettant de traiter les accès, les données d'amorçage, les états d'allocation ou encore les descripteurs de sécurité. Procéder ainsi (comprendre le fait de tout stocker dans des fichiers) facilite la localisation et la gestion des données, et permet, en cas de problème, de proposer un recours rapide et efficace (nous reviendrons sur ce point plus loin dans ce chapitre).

NTFS emploie pour stocker les informations concernant fichiers et dossiers une base de données appelée table de fichiers maîtresse (MFT, *Master File Table*), laquelle se présente sous forme d'une série d'enregistrements. Très logiquement, la MFT contient un enregistrement pour chaque ficher et dossier d'un volume, plus diverses informations servant à gérer le volume lui-même. Voici, sans pour l'instant entrer dans le détail, quelques types de données données que renferme la MFT :

- *Nom* Nom via lequel le fichier est enraciné dans le système de fichiers.
- *Taille*
- *Horodatages*
- *Attributs*
- *Droits d'accès (ACL)*
- *Liste des blocs (clusters) contenant le fichier*

Chaque enregistrement défini au sein de la MFT occupe 1 Ko, quelle que soit la taille de cluster. En principe, chaque enregistrement MFT correspond à un fichier différent. Toutefois, dans l'éventualité où un fichier a un nombre conséquent d'attributs, ou est très fragmenté, plusieurs enregistrements sont auquel cas monopolisés.

Compte tenu de propriétés de la MFT, les fichiers de petite taille (moins d'1 Ko) peuvent être entièrement stockés dans un enregistrement de ladite table, en utilisant auquel cas l'espace normalement réservé pour la liste des blocs (clusters). Il résulte de cette configuration deux avantages : d'une part une diminution de la perte d'espace occasionnée presque inexorablement par ce genre d'entité, d'autre part une optimisation des accès. De fait, ici, NTFS accède à la MFT et obtient les données immédiatement, sans rencontrer d'avantage d'indirection.

Quand une opération de suppression de fichiers a lieu, Windows modifie l'entrée MFT adéquate pour la faire passer de occupé à libre, indiquant dès lors la possibilité de la réutiliser ultérieurement (en l'occurrence pour un nouveau fichier). L'enregistrement MFT du fichier, du reste comme le contenu de ce fichier, demeure sur le disque. Ceci explique pourquoi le fichier MFT ne cesse de s'amplifier au fur et à mesure des opérations qui altérent la structure d'un volume NTFS.

Outre la MFT, chaque volume NTFS inclut un ensemble de fichiers de méta données contenant les informations donnant sa forme finale à la structure du système de fichiers. Chacun de ces fichiers a un nom qui débute par le signe $. Le tableau qui suit répertorie les enregistrements de la MFT associés aux fichiers de méta données NTFS.

***Tableau 17.2. Enregistrements MFT réservés***

| Index | Nom | Description |
|---|---|---|
| 0 | $Mft | Table maîtresse des fichiers |

| Index | Nom | Description |
|---|---|---|
| 1 | $MftMirr | MFT miroitée |
| 2 | $LogFile | Fichier de journalisation |
| 3 | $Volume | Fichier volume |
| 4 | $AttrDef | Table de définitions des attributs |
| 5 | $. | Répertoire racine |
| 6 | $Bitmap | Table d'allocation des clusters |
| 7 | $Boot | Fichier d'amorçage |
| 8 | $BadClus | Clusters défectueux |
| 9 | $Secure | Descripteurs de sécurité |
| 10 | $Upcase | Mappage des majuscules |
| 11 | $Extend | Méta données étendues |
| 12-23 | s/o | Inutilisé |
| 24 | $Extend\$Quota | Infomations de quota |
| 25 | $Extend\$ObjId | Suivi des ID d'objet |
| 26 | $Extend\$Reparse | Points reparse |
| 27 | $Extend\$RmMetadata | |
| 28 | $Extend\$RmMetadata\$Repair | |
| 29 | $Extend\$RmMetadata\$TxfLog | |
| 30 | $Extend\$RmMetadata\$Txf | |
| 31 | $Extend\$RmMetadata\$TxfLog\$Tops | |
| 32 | $Extend\$RmMetadata\$TxfLog\$TxfLog.blf | |
| 33+ | s/o | Fichiers et répertoires utilisateur |

- *Répertoire racine ($.)* Répertorie les fichiers et répertoires stockés à la racine de la structure de répertoires NTFS.

- *Définition des attributs (AttrDef)* Définit les types d'attributs pris en charge sur le volume et indique s'ils peuvent être indexés, rétablis après une opération de récupération système, ainsi que s'ils exigent un espace d'allocation (et donc des clusters) n'appartenant pas à la MFT.

- *Clusters défectueux (BadClus)* Garde trace de quels emplacements sur le volume ont été marqués comme défectueux.

- *Table d'allocation des clusters (Bitmap)* Catalogue l'ensemble des clusters dont est constitué le volume et indique pour chaque s'il est libre ou allouée à un fichier.

- *Fichier d'amorçage (Boot)* Contient les informations nécessaires au démarrage du système : adresse du chargeur, taille des secteurs, nombre total de secteurs, nombre de secteurs par cluster, nombre de cluster par enregistrement de la MFT, adresse du premier cluster de la MFT et de la MFT de sauvegarde, adresse du programme d'amorce NTFS, et ainsi de suite.

- *Fichier journal (LogFile)* Enregistre les opérations qui affectent la structure du volume NTFS, entre autres les créations et les duplications de fichier, ce dans l'optique d'offrir en cas de dysfonctionnement (panne de courant, incident système, etc.) des possibilités de récupération.

- *MFT miroitée (MftMirr)* Duplicata des premiers enregistrements de la MFT destiné à permettre la restauration d'une partition dont la MFT serait altérée.

- *Points reparse (Reparse)* Identifie les fichiers et répertoires contenant des points reparse.

- *Sécurité (Secure)* Base de données des descripteurs de sécurité du volume. Si les fichiers et répertoires NTFS ont des descripteurs de sécurité individuels, conformément cela dit aux standards qui prévalent en matière de

controle d'accès, NTFS centralise les paramètres dans un fichier commun, ce qui permet aux entités ayant les mêmes attributs de sécurité de référencer le même descripteur.

- *Mappage des majuscules (Upcase)* Table de traduction entre minuscules et majuscules.
- *Fichier de volume (Volume)* Informations se rapportant au volume: nom, version de NTFS avec laquelle il a été structuré, et état (corrompu ou non, amenant dans le premier cas à une intervention via l'utilitaire Chkdsk).

### Références de fichier

Chaque fichier enraciné dans un volume NTFS l'est au moyen d'une valeur appelée référence de fichier, laquelle se compose d'un numéro de fichier et d'un numéro de séquence. Le numéro de fichier correspond à la position de l'enregistrement associé au fichier dans la MFT (ou à la position de l'enregistrement de base, si le fichier a plusieurs enregistrements qui lui sont consacrées). Le numéro de séquence est utilisé est interne pour des contrôles.

### Visualisation des données de volume

Vous pouvez afficher des informations détaillés concernant un volume NTFS, entre autres l'emplacement et la taille de la MFT, au moyen de l'utilitaire ligne de commande intégré Fsutil.

```
C:\> fsutil fsinfo ntfsinfo c:
Numéro de série du volume NTFS :       0x82d65992d65986f7
Version NTFS    :                  3.1
Version LFS    :                  2.0
Nombre de secteurs :                  0x0000000022ee8fff
Nombre total de clusters :                 0x00000000045dd1ff
Clusters libres  :                  0x00000000025b5686
Total réservé :                  0x00000000000060fc
Octets par secteur  :               512
Octets par secteur physique :       4096
Octets par cluster :               4096
Octets par segment d'enreg. de fich.    : 1024
Clusters par segment d'enreg. de fich. : 0
Longueur de données valide MFT :           0x000000002c9c0000
LCN de démarrage MFT  :                  0x00000000000c0000
LCN de démarrage MFT2 :                  0x0000000000000002
Début de la zone MFT :                  0x0000000002547200
Fin de la zone MFT   :                  0x000000000254d900
Nombre d'étendues de réglage d'appareil max :     0
Nombre d'octets de réglage d'appareil max :       0x0
Nombre d'étendues de réglage de volume max :     62
Nombre d'octets de réglage de volume max :       0x40000000
Identificateur du Gestionnaire de ressources :        3B75B80A-CD24-11E8-89F0-D8CB8AA93022
```

À l'aide, encore, de la commande Fsutil, Vous pouvez également obtenir des statistiques détaillées sur les métadonnées et l'utilisation des fichiers utilisateur.

```
C:\> fsutil fsinfo statistics c:
Type de système de fichiers :     NTFS

UserFileReads :        1349229
UserFileReadBytes :    132431971328
UserDiskReads :        1204361
UserFileWrites :       712097
UserFileWriteBytes :   173602827776
UserDiskWrites :       749218
MetaDataReads :        209955
MetaDataReadBytes :    2174689280
MetaDataDiskReads :    254156
MetaDataWrites :       328097
MetaDataWriteBytes :   1861795840
```

```
MetaDataDiskWrites :   417429

MftReads :                      177476
MftReadBytes :                  2022883328
MftWrites :                     238353
MftWriteBytes :                 1291509760
Mft2Writes :                    0
Mft2WriteBytes :                0
RootIndexReads :                0
RootIndexReadBytes :            0
RootIndexWrites :               0
RootIndexWriteBytes :           0
BitmapReads :                   4586
BitmapReadBytes :               28733440
BitmapWrites :                  58930
BitmapWriteBytes :              350195712
MftBitmapReads :                22
MftBitmapReadBytes :            212992
MftBitmapWrites :               6707
MftBitmapWriteBytes :           30142464
UserIndexReads :                70128
UserIndexReadBytes :            694505472
UserIndexWrites :               88560
UserIndexWriteBytes :           442216448
LogFileReads :                  8
LogFileReadBytes :              32768
LogFileWrites :                 453842
LogFileWriteBytes :             5458968576
LogFileFull :                   16
DiskResourceFailure :           0
VolumeTrimCount :               7052
AvgVolumeTrimTime (ms) :        0
AvgVolumeTrimSize (Ko) :        35227
AvgVolumeTrimSpeed (Ko/s) :     249595702
VolumeTrimSkippedCount :        0
VolumeTrimSkippedSize (Ko) :    0
FileLevelTrimCount :            0
AvgFileLevelTrimTime (ms) :     0
AvgFileLevelTrimSize (Ko) :     0
AvgFileLevelTrimSpeed (Ko/s) :  0
NtfsFillStatInfoFromMftRecordCalledCount :                              0
NtfsFillStatInfoFromMftRecordBailedBecauseOfAttributeListCount :        0
NtfsFillStatInfoFromMftRecordBailedBecauseOfNonResReparsePointCount :   0
```

## Attributs de fichier NTFS

NTFS stocke les fichiers en tant qu'ensemble de paires attribut/valeur, l'une de ces paires représentant les données proprement dites du fichier. D'autres attributs comprennent le nom du fichier, des informations sur la sécurité, sur d'éventuelles données *reparse* associées, etc. NTFS ordonne les attributs définis au sein d'un enregistrement fichier par l'intermédiaire de valeurs dites codes de type. Le tableau qui suit répertorie les attributs envisageables pour un fichier.

***Tableau 17.3. Attributs de fichier NTFS***

| Code de type | Attribut | Nom | Description |
|---|---|---|---|
| 0x10 | Informations standard | $STANDARD_INFORMATION | Attributs fichier du genre lecture-seule, archive, etc. ; données d'horodatage (date de création et date de dernière modification). |
| 0x20 | Liste d'attributs | $ATTRIBUTE_LIST | Liste des attributs composant le fichier. |
| 0x30 | Nom de fichier | $FILE_NAME | Nom Unicode du fichier. |

| Code de type | Attribut | Nom | Description |
|---|---|---|---|
| 0x40 | ID d'objet | $OBJECT_ID | Identifiant du fichier ou répertoire, permettant d'en assurer le suivi dans l'éventualité où son nom ou son emplacement viendrait à changer. |
| 0x50 | Descripteur de sécurité | $SECURITY_DESCRIPTOR | Détermine qui peut accéder au fichier et de quelle manière. |
| 0x60 | Nom de volume | $VOLUME_NAME | Nom du volume. Seul le fichier de méta données $VOLUME incorporent cet attribut. |
| 0x70 | Informations de volume | $VOLUME_INFORMATION | Données de version de NTFS et informations concernant l'état du volume. Seul le fichier de méta données $VOLUME incorporent cet attribut. |
| 0x80 | Données | $DATA | Contenu du fichier. |
| 0x90 | Racine d'index | | |
| 0xA0 | Allocation d'index | | |
| 0xB0 | Bitmap | $BITMAP | |
| 0xCO | Données reparse | $REPARSE_POINT | Contient les données de point reparse du fichier. Jonctions (liens symboliques) et points de montage incluent cet attribut. |
| 0xD0, 0xE0 | Attributs étendus | $EA_INFORMATION, $EA | Deux attributs utilisés pour la rétro compatibilité avec les applications OS/2 (HPFS) |

Certains types d'attributs peuvent apparaître plusieurs fois pour un même fichier. C'est le cas, par exemple, quand un fichier a plusieurs noms ou plusieurs flots de données.

Quand un fichier est de petite taille, tous ses attributs (ainsi que leurs valeurs) sont stockés dans l'enregistrement correspondant, et auquel cas dits résidants. Un certain nombre d'attributs sont définis de la sorte (comprendre toujours résidants), par exemple le nom du fichier ($FILE_NAME) et les informations standards ($STANDARD_INFORMATION) associées.

Chaque attribut commence par un en-tete standard contenant des informations sur l'attribut, tels le type auquel il appartient, si la valeur associée est résidante ou non, etc.

## Secteur d'amorçage NTFS

```
Offset(h)   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000000000  EB 52 90 4E 54 46 53 20 20 20 20 00 02 08 00 00  ëR.NTFS    .....
0000000010  00 00 00 00 00 F8 00 00 3F 00 FF 00 00 28 07 00  .....ø..?.ÿ..(..
0000000020  00 00 00 00 80 00 80 00 FF 8F EE 22 00 00 00 00  ....€.€.ÿ.î"....
0000000030  00 00 0C 00 00 00 00 00 02 00 00 00 00 00 00 00  ................
0000000040  F6 00 00 00 01 00 00 00 F7 86 59 D6 92 59 D6 82  ö.......÷†YÖ'YÖ‚
0000000050  00 00 00 00 FA 33 C0 8E D0 BC 00 7C FB 68 C0 07  ....ú3ÀŽÐ¼.|ûhÀ.
0000000060  1F 1E 68 66 00 CB 88 16 0E 00 66 81 3E 03 00 4E  ..hf.Ë^...f.>..N
0000000070  54 46 53 75 15 B4 41 BB AA 55 CD 13 72 0C 81 FB  TFSu.´A»ªUÍ.r..û
0000000080  55 AA 75 06 F7 C1 01 00 75 03 E9 DD 00 1E 83 EC  Uªu.÷Á..u.éÝ..ƒì
0000000090  18 68 1A 00 B4 48 8A 16 0E 00 8B F4 16 1F CD 13  .h..´HŠ...‹ô..Í.
00000000A0  9F 83 C4 18 9E 58 1F 72 E1 3B 06 0B 00 75 DB A3  ŸƒÄ.žX.rá;...uÛ£
00000000B0  0F 00 C1 2E 0F 00 04 1E 5A 33 DB B9 00 20 2B C8  ..Á.....Z3Û¹. +È
00000000C0  66 FF 06 11 00 03 16 0F 00 8E C2 FF 06 16 00 E8  fÿ.......ŽÂÿ...è
00000000D0  4B 00 2B C8 77 EF B8 00 BB CD 1A 66 23 C0 75 2D  K.+Èwï¸.»Í.f#Àu-
00000000E0  66 81 FB 54 43 50 41 75 24 81 F9 02 01 72 1E 16  f.ûTCPAu$.ù..r..
00000000F0  68 07 BB 16 68 52 11 16 68 09 00 66 53 66 53 66  h.».hR..h..fSfSf
0000000100  55 16 16 16 68 B8 01 66 61 0E 07 CD 1A 33 C0 BF  U...h¸.fa..Í.3À¿
```

```
0000000110  0A 13 B9 F6 0C FC F3 AA E9 FE 01 90 90 66 60 1E  ..¹ö.üóªéþ...f`.
0000000120  06 66 A1 11 00 66 03 06 1C 00 1E 66 68 00 00 00  .f¡..f.....fh...
0000000130  00 66 50 06 53 68 01 00 68 10 00 B4 42 8A 16 0E  .fP.Sh..h..´BŠ..
0000000140  00 16 1F 8B F4 CD 13 66 59 5B 5A 66 59 66 59 1F  ...‹ôÍ.fY[ZfYfY.
0000000150  0F 82 16 00 66 FF 06 11 00 03 16 0F 00 8E C2 FF  .‚..fÿ.......ŽÂÿ
0000000160  0E 16 00 75 BC 07 1F 66 61 C3 A1 F6 01 E8 09 00  ...u¼..faÃ¡ö.è..
0000000170  A1 FA 01 E8 03 00 F4 EB FD 8B F0 AC 3C 00 74 09  ¡ú.è..ôëý‹ð¬<.t.
0000000180  B4 0E BB 07 00 CD 10 EB F2 C3 0D 0A 41 20 64 69  ´.»..Í.ëòÃ..A di
0000000190  73 6B 20 72 65 61 64 20 65 72 72 6F 72 20 6F 63  sk read error oc
00000001A0  63 75 72 72 65 64 00 0D 0A 42 4F 4F 54 4D 47 52  curred...BOOTMGR
00000001B0  20 69 73 20 63 6F 6D 70 72 65 73 73 65 64 00 0D   is compressed..
00000001C0  0A 50 72 65 73 73 20 43 74 72 6C 2B 41 6C 74 2B  .Press Ctrl+Alt+
00000001D0  44 65 6C 20 74 6F 20 72 65 73 74 61 72 74 0D 0A  Del to restart..
00000001E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000001F0  00 00 00 00 00 00 8A 01 A7 01 BF 01 00 00 55 AA  ......Š.§.¿...Uª
```

| Position(h) | Longueur(o) | Nom |
|---|---|---|
| 000 | 3 | Instruction de saut |
| 003 | 8 | Nom du fabricant |
| 00B | 2 | Octets par secteur |
| 00D | 1 | Secteurs par cluster |
| 00E | 2 | Secteurs réservés |
| 015 | 1 | Type de media |
| 018 | 2 | Secteurs par piste |
| 01A | 2 | Nombre de têtes |
| 01C | 4 | Secteurs cachés |
| 028 | 8 | Nombre de secteurs |
| 030 | 8 | Emplacement de la MFT |
| 038 | 8 | Emplacement de la MFT miroitée |
| 040 | 1 | Clusters/Octets par segment |
| 048 | 8 | Numéro de série du volume |
| 054 | 426 | Routine de démarrage |
| 1FE | 2 | Signature |

- **Instruction de saut** Saut inconditionnel visant à poursuivre l'exécution après les structures de données définies dans ce secteur. Exemple (offset 0-2) : EB 52 90, équivalent à un saut (jmp) vers le code machine exécutable située à l'offset 0x54.
- **Nom du fabricant** Ensemble de caractères employé en vue d'identifier le système de fichiers. Exemple (offset 3-10) : 4E 54 46 53 20 20 20 20, à quoi correspond l'acronyme NTFS suivi de quatres espaces terminaux.
- **Octets par secteur** Nombre d'octets que renferme un secteur de disque. Exemple : 00 02, soit 512 octets.
- **Secteurs par cluster** Nombre d'octets qu'encapscule un secteur de disque. Exemple : 08.
- **Secteurs réservés** Nombre de secteurs que le système d'exploitation a réservé au début du disque.
- **Type de media** Catégorie à laquelle s'apparente le support de stockage Exemple : F8, correspondant à un disque dur.
- **Secteurs par piste** Nombre de secteurs par piste. Exemple : 3F 00 (0x003F).
- **Nombre de têtes** Nombre de pistes par cylindre. Exemple : FF 00 (0x003F ou 255).
- **Secteurs cachés** Nombre de secteurs qui précèdent la partition.
- **Nombre de secteurs** Nombre de secteurs que recouvre la partition.

- **Emplacement de la MFT** Numéro du cluster où réside la table maitresse des fichiers (MFT).
- **Emplacement de la MFT miroitée** Numéro du cluster où réside la MFT miroitée.
- **Clusters/Octets par segment** Une valeur positive indique le nombre de clusters dans un segment d'enregistrement de fichier. Une valeur négative indique la quantité d'octets dans un segment d'enregistrement de fichier, auquel cas la taille est de 2 à la puissance de la valeur absolue. Exemple : F6 (-10), pour 2 puissance 10, 1024.
- **Numéro de série** Valeur unique assignée à la partition. Exemple : F7 86 59 D6, correspondant le numéro de série D659-86F7. (Bien que NTFS s'appuie en interne sur des numéros de série constitués de 8 octets, une large majorité des commandes Windows d'administration des disques, par exemple la commande vol, n'en montrent que quatre.)
- **Routine de démarrage** Séquence exécutable chargée de faire aller de l'avant le démarrage du système d'exploitation.
- **Signature** Marque la fin d'un enregistrement de démarrage.

## ID d'objet

Tout fichier ayant un ID d'objet dispose également des propriétés suivantes :

- **ID de volume (BirthVolumeID)** ID du volume à partir duquel le fichier a été créé.
- **ID d'origine (BirthObjectID)** ID de l'objet tel qu'il a été attribué au fichier au moment de sa création.
- **ID de domaine (DomainID)** ID du domaine à partir duquel le fichier a été créé.

Quand un fichier fait suite d'une opération de copie ou de déplacement vers un volume NTFS, un nouvel ID d'objet lui est automatiquement assigné. Les informations originelles concernant le fichier sont conservées à l'aide d'attributs spécifiquement établis en ce sens (BirthVolumeID et BirthObjectID).

NTFS fournit des méthodes permettant de manipuler fichiers et répertoires par leurs ID plutôt que par leur nom. Plusieurs utilitaires agissent de même, par exemple la commande mountvol qui, utilisée sans paramètre, affiche une liste d'ID correspondant aux volumes et points de montage disponibles.

```
C:\mountvol
.
.
.
    \\?\Volume{7dfe2ede-2186-4a17-961d-616fa7ce6d91}\
        C:\

    \\?\Volume{ff3edecb-a375-4cdf-b4c5-c0ce5ca1d0b3}\
        *** Aucun point de montage ***

    \\?\Volume{0d0987e9-d7c4-48ec-b6ab-0eaa57b0b526}\
        D:\

    \\?\Volume{8b21e38b-d16a-4fbd-b7de-5dacbd6b8adb}\
        *** Aucun point de montage ***

    \\?\Volume{038d9cb0-c804-4d79-8582-643f9109ff0b}\
        *** Aucun point de montage ***
```

Windows octroie aux applications la possibilité d'assigner un ID d'objet à un fichier par l'intermédiaire des codes de contrôle de système de fichiers FSCTL_CREATE_OR_GET_OBJECT_ID et FSCTL_SET_OBJECT_ID. Pour connaitre les ID d'objet, employez les codes de contrôle FSCTL_CREATE_OR_GET_OBJECT_ID et FSCTL_GET_OBJECT_ID. Le code de contrôle FSCTL_DELETE_OBJECT_ID permet de supprimer les ID d'objet.

## Gestion des fichiers et répertoires

*Tableau 17.4. Fonctions de gestion de fichiers*

| Fonction | Description |
|---|---|
| CreateDirectory | Crée un nouveau répertoire sur le chemin donné |
| GetFullPathName | Retourne le chemin complet et le nom d'un fichier |
| Get/SetCurrentDirectory | Retourne ou modifie le répertoire en cours |
| GetSystemDirectory | Retourne le chemin d'accès du dossier système de Windows |
| GetWindowsDirectory | Retourne le chemin d'accès du répertoire où Windows est installé |
| RemoveDirectory | Supprime un répertoire existant |
| ReadDirectoryChangesW | Informe l'appelant des changements qui ont lieu dans un répertoire donné |

## Common Log File System (CLFS)

Les fonctionnalités transactionnelles intégrées à Windows le sont à l'aide d'un composant spécialisé, dit Système commun de journalisation (CLFS, Common Log File System), dont la fonction consiste à garder trace des modifications apportés à des fichiers ou à des entrées, en vue de les annuler si nécessaire.

Bien que CLFS soit présenté en tant que système de fichiers, notez qu'il n'en est en réalité pas un, mais définit par contre une couche d'abstraction virtuelle au-dessus de NTFS en utilisant des flux (streams) et des conteneurs.

Chaque journal CLFS peut éventuellement être régi par un ensemble de stratégies de gestion configurable par le client. Pour plus d'informations, consultez la structure CLFS_MGMT_POLICY.

## Défragmentation

La version ligne de commande de Défragmenteur de disque offre des contrôles étendus sur le processus de défragmentation. Afin de mettre en oeuvre la commande pour un lecteur donné, saisissez defrag x: dans une invite de commandes, où x représente la lettre de l'unité ou un point monté d'un volume existant.

Les commutateurs suivants sont supportés :

- **-a** Analyse le lecteur ou le volume spécifié et affiche un récapitulatif du rapport d'analyse.
- **-c** Défragmente tous les volumes présents sur le disque.
- **-e** Défragmente tous les volumes présents sur le disque, à l'exception de ceux spécifiés.
- **-r** Effectue une défragmentation partielle en ne consolidant que des fragments de fichier dont la taille est inférieure à 64 Mo.
- **-u** Affiche à l'écran l'état d'avancement de l'opération.
- **-w** Effectue une défragmentation complète en consolidant tous les fragments de fichier, indépendamment de leur taille.

## Extensions de noms de fichiers

Une extension de nom de fichier (ou simplement extension de fichier, voire extension) est un ensemble de caractères enchâssé à un nom de fichier qui caractérise le type d'informations que le fichier renferme. Par exemple, dans le nom de fichier notepad.exe, l'extension .exe indique que le fichier accueille un programme exécutable (en l'occurence, ici, l'application Bloc-Notes).

Les logiciels, systèmes d'exploitation compris, appréhendent le format d'un fichier d'après l'extension qui lui est associée. Lors de l'accès à un fichier, Windows examine son extension et détermine l'application à exécuter en vue de sa prise en charge.

***Tableau 17.5. Extensions de fichiers courantes***

| Extension | Description | Logiciels compatibles |
|---|---|---|
| bat | fichier exécutable batch | natif |
| bmp | fichier bitmap | |
| dll | bibliothèque de liens dynamiques | natif |
| doc | fichier texte | Word, WordPad, Bloc-Notes |
| exe | image exécutable | natif |
| htm | fichier Hypertext Markup Language | Edge, Firefox, Chrome |
| iso | image CD ISO 9660 | natif |
| pdf | Portable Document Format | Acrobat |
| xls | fichier tableur | Excel |

Dans une large majorité des cas, l'extension de nom d'un fichier est attribuée par le programme ayant assuré la création du fichier concerné. Elle peut toutefois être modifiée a posteriori par l'utilisateur, de la même manière que le nom du fichier.

## Conversion d'un volume en NTFS

Windows intègre nativement plusieurs outils pour convertir des volumes FAT ou FAT32 en NTFS, dont l'utilitaire ligne de commandes Convert (Convert.exe), lequel fonctionne de manière non destructrice, et opère de ce fait en conservant la structure des fichiers et des répertoires sans perte de données.

Pour convertir, utilisez la syntaxe suivante dans invite de commandes élevée : convert volume /fs:ntfs, où volume est une lettre de lecteur (suivie de deux points), un chemin de lecteur ou un nom de volume. La syntaxe générale accompagnant l'utilitaire se présente comme suit :

```
convert [volume] /fs:ntfs [/v] [/cvtarea:nomfichier] [/nosecurity] [/x]
```

Voici à quoi correspondent ces options :

- **/FS:NTFS** Convertir le volume en NTFS.
- **/V** Active le mode verbeux, lequel rend compte de plus de détails à l'écran.
- **/F** Force le démontage du volume avant la conversion (si nécessaire).
- **/cvtarea:nomfichier** Spécifie un fichier à la racine du disque qui sera employé afin de stocker la MFT et les autres fichiers de métadonnées NTFS. L'emploi d'une telle option peut aboutir à un système de fichiers moins fragmenté au terme de la conversion.
- **/nosecurity** Définit les paramètres de sécurité NTFS de tous les fichiers et dossiers de sorte que le groupe Tout le monde puisse y accéder. Cette option met par nature l'ensemble du système de fichiers à la merci de quiconque peut accéder à la machine (en local ou à distance).

# Chapitre 18. Démarrage et arrêt

Le présent chapitre aborde en détail le processus de démarrage de Microsoft Windows. Nous verrons quelles opérations s'effectuent lors de la phase d'amorçage de l'ordinateur, quels mécanismes sont impliqués, et quelles options sont influentes en la matière. Nous passerons ensuite en revue les différents problèmes susceptibles de se manifester lors du démarrage, et comment les résoudre. Enfin, nous expliquerons ce qui arrive au cours de l'arrêt du système.

## BootRec

BootRec accepte les paramètres de ligne de commande suivants :

- **fixmbr** Écrit un enregistrement de démarrage principal sur la partition système.
- **fixboot** Écrit un nouveau secteur de démarrage sur la partition système.
- **scanos** Examine les disques à la recherche d'installations de Windows et rapporte les entrées actuellement absentes du stockage BCD.
- **rebuildbcd** Examine les disques à la recherche d'installations de Windows et procure un choix concernant les entrées à ajouter au stockage BCD.

## Dernière bonne configuration connue

La dernière bonne configuration connue est un mécanisme des plus utiles lorsqu'il s'agit de remettre d'aplomb un système qui ne démarre plus. Lorsque vous apportez des modifications sur l'ordinateur, les paramètres de registre qui en résultent ne sont pour la plupart pas conservés définitivement (comprendre pas irrévocablement). Certains paramétrages personnels sont en effet enregistrés à l'extinction de la machine, tandis que d'autres le sont au démarrage de la session. Le système a pour principe de considérer une configuration système comme stable si redémarrez l'ordinateur et arrivez jusqu'au Bureau Windows (à quoi correspond en interne une ouverture de session réussie). À ce moment, il enregistre cette configuration comme "dernière bonne connue" qui, un peu à la manière d'un point de restauration, servira si besoin est à retourner à un état stable et cohérent de la plate-forme.

En cas de recours à la dernière bonne configuration connue, le système d'exploitation commence tout d'abord par marquer la configuration qu'il employait pour son amorçage comme erronée en initialisant la valeur Failed sous HKLM\System\Select. Il modifie ensuite HKLM\System\Select\Current pour y mettre la valeur stockée dans HKLM\System\Select\LastKnownGood. Enfin, il actualise le lien symbolique HKLM\SYSTEM\CurrentControlSet de sorte à le faire pointer vers la configuration LastKnownGood. Supposons, à titre d'exemple, que que vous ayez installé un pilote introduisant un dysfonctionnement parmi la procédure de démarrage, lequel fait immanquablement s'effondrer le système. En pareil circonstance, vous pouvez en utilisant la dernière bonne configuration connue défaire vos changement. La clé du pilote récemment installé n'étant pas présente dans la sous-clé Services du jeu LastKnownGood, le système peut de ce fait démarrer sans problème.

## Mode sans échec

Le mode sans échec est une configuration spéciale d'amorçage prévue pour n'exécuter qu'un sous-ensemble des pilotes et des services configurés pour démarrer en mode normal. En s'appuyant sur les seuls pilotes et services qui sont indispensable à l'amorçage, Windows évite de la sorte de charger des pilotes tierce partie et/ou non essentiels qui pourraient faire s'effondrer le système.

La prise en charge du mode sans échec par Windows consiste essentiellement en trois variantes : Mode sans échec, Mode sans échec avec prise en charge réseau et Invite de commandes en mode sans échec. Le mode sans échec standard comprend le minimum de pilotes et de services indispensables au démarrage. Le mode sans échec avec prise en charge réseau ajoute à cela des pilotes et services réseau permettant d'accéder à Internet et à d'autres ordinateurs. Enfin, le mode sans échec avec invite de commandes est identique au mode sans échec standard à ceci près que Windows exécute l'application Invite de commandes (Cmd.exe) au lieu de l'explorateur comme shell quand le système bascule en mode graphique.

Windows inclut un quatrième mode sans échec, incarné en l'occurrence par le mode restauration Active Directory, très différent en réalité du mode sans échec standard ou réseau. Le mode de restauration Active Directory fait démarrer le système dans un mode où le service d'annuaire d'un contrôleur de domaine est hors ligne. Cela permet de faire une réparation sur la base de données ou de la sauvegarder depuis un support de sauvegarde. Dans cette configuration, tous les pilotes et services, exception faite du service Active Directory, sont chargés.

Pour démarrer votre ordinateur en mode sans échec, commencez par ouvrir les Paramètres de l'ordinateur (par exemple en appuyant sur les touches Win + I). Sélectionnez ensuite Mise à jour et sécurité, puis Récupération. Sous Démarrage avancé, sélectionnez Redémarrer maintenant. Après le redémarrage de votre ordinateur, dans l'écran Choisir une option, sélectionnez Dépannage, Options avancées, Paramètres de démarrage, Redémarrer. Après le redémarrage de votre ordinateur, plusieurs options s'affichent. Sélectionnez la touche 4 ou F4 pour démarrer votre ordinateur en mode sans échec.

Compte tenu de la nature des opérations généralement assurées dans le mode sans échec (de l'ordre du diagnostic), l'ouverture d'une session en mode sans échec n'actualise pas l'ensemble de contrôle LastKnownGood.

La configuration du mode sans échec se réalise via la clé HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot qui liste les pilotes et services minimaux à charger en mode standard (sous-clé Minimal) ou avec support réseau (sous-clé Network).

### Programmes utilisateur reconnaissant le mode sans échec

La présence au démarrage de paramètres de mode sans échec entraine une configuration spéciale de certaines valeurs de registre et variables d'environnement, servant de la sorte aux programmes utilisateur dans l'éventualité où ils adoptent un comportement différent selon le mode d'amorçage du système.

Le composant mode utilisateur SCM, implémenté par Services.exe, lit HKLM\SYSTEM\ CurrentControlSet \Control\SafeBoot\Option\OptionValue pour savoir si le système est en train de démarrer en mode sans échec. Userinit lit HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\Option\UseAlternateShell. Si cette valeur est définie, Userinit exécute le programme spécifié comme shell de l'utilisateur dans la valeur HKLM\SYSTEM \CurrentControlSet\Control\ SafeBoot\AlternateShell, par défaut Cmd.exe.

Les applications Windows ayant besoin d'effectuer certaines opérations quand le système démarre en mode sans échec emploient à cette intention la fonction Windows GetSystemMetrics(SM_CLEANBOOT). Les scripts batch, quant à eux, recherchent la variable d'environnement SAFEBOOT_OPTION, laquelle n'est définie que dans le mode sans échec.

## Processus de démarrage

La séquence de démarrage de Windows se compose des étapes que voici :

1. Phase d'autotest d'allumage (POST)
2. Phase initiale de démarrage
3. Phase du gestionnaire de démarrage Windows
4. Phase du chargeur de démarrage Windows
5. Phase de chargement du noyau
6. Phase d'ouverture de session

La liste qui suit énumère les composants impliqués lors du processus d'amorçage.

- *Code MBR* Charge les secteurs d'amorçage des partitions. Les instructions exécutées durant cette phase le sont en mode réel 16 bits.

- *Secteur d'amorçage* Lit le répertoire racine pour charger Bootmgr. Les instructions exécutées durant cette phase le sont en mode réel 16 bits.
- *Bootmgr* Lit le stockage BCD, affiche le menu de démarrage et prépare l'exécution pour différents utilitaires de démarrage pré-système d'exploitation, tels que Diagnostics de la mémoire (Memtest.exe). Sur une installation 32 bits, bascule en mode protégé 32 bits ; sur une installation 64 bits, bascule en mode étendu 64 bits.
- *Chargeur de démarrage (Winload.exe)* Charge Ntoskrnl.exe, les dépendances dont ce module a besoin (Hal.dll, Kdcom.dll et autres), et les pilotes indispensables pour amorcer le système.
- *Dispostif de reprise (Winresume.exe)* Charge les données de veille prolongée (Hiberfil.sys) et configure le système à l'état enregistrée avant la mise en veille.
- *Exécutif et noyau (Ntoskrnl.exe)* Initialise les sous-systèmes de l'exécutif et les pilotes marqués boot-start et system-start, prépare le système à l'exécution d'applications natives et donne lieu au processus hébergeant le Gestionnaire de session (Smss.exe).
- *Gestionnaire de session (Smss.exe)* Charge le sous-système Windows, incluant Win32k.sys, Csrss.exe et Wininit.exe (session 0). Démarre les processus Csrss.exe et Winlogon (session 1 et au-delà). Smms se présente sous la forme d'une application native.
- *Wininit.exe* Démarre le SCM, le sous-système de sécurité locale (LSASS) et le gestionnaire de session locale (LSM).
- *Winlogon.exe*
- *Logonui.exe* Affiche la boite de dialogue d'ouverture de session interactive.
- *Services.exe* Charge et initialise les pilotes et services Windows automatiques.

## Phase d'autotest d'allumage

Dès le moment où vous allumez votre ordinateur, son processeur commence par exécuter les instructions contenues dans le BIOS (*Basic Inuput/Output System*) ou le EFI (*Extensible Firmware Interface*). Les BIOS et EFI, deux types de microprogrammes (*firmware*) renferment le code dépendant du processeur qui démarre l'ordinateur quel que soit le système d'exploitation installé. Le premier ensemble d'instructions de démarrage est l'autotest d'allumage (POST, Power-on Self Test), qui effectue la configuration primaire de l'ordinateur.

La séquence POST est responsable des fonctions et diagnostics suivants :

- Repérages initiaux concernant le matériel, comme la détermination de la quantité de mémoire vive présente ou la vérification de la présence des périphériques nécessaires au démarrage d'un système d'exploitation, par exemple un disque dur.
- Contrôles initiaux du matériel, comme la vérification du bus système et des connecteurs d'extension
- Vérification de la mémoire vidéo et des signaux des commandant l'affichage

Si le démarrage échoue avant ou pendant la POST, l'ordinateur ne va pas plus loin. Dans l'éventualité où la séquence POST détecte un élément défectueux ou manquant, elle le signale généralement à l'aide d'une série de bips. Les signaux sont émis sur le haut-parleur interne. Aucune carte son n'est utilisée.

## Phase du chargeur de démarrage Windows

Winload :

1. Vérification de l'intégrité du processus Winload.exe en comparant la signature générée à partir de l'image en mémoire du processus à celle donnée dans le fichier catalogue de signature (nt5.cat). Si les signatures ne correspondent pas, le système s'arrête. Une exception à cette règle est cependant prévue pour le cas où un débogueur mode noyau contrôle la machine (un avertissement est auquel cas affiché sur la console).

2. Chargement des DLL pshed.dll, bootvid.dll, clfs.sys et ci.dll (dans cet ordre).
3. Chargement des images appropriées du noyau et de la couche d'abstraction matérielle (Ntoskrnl.exe et Hal.dll par défaut). Si Winload n'arrive pas à charger l'un ou l'autre de ces fichiers, le message "Windows n'a pas pu démarrer car le fichier suivant est manquant ou endommagé", suivi du nom du fichier incriminé.
4. Lecture du fichier de fontes VGA (par défaut, vgaoem.fon). Si l'opération échoue, le même message d'erreur que celui donné à l'étape précédente est affiché.
5. Lecture des fichiers de support de langue nationale (NLS), utilisés pour l'internationalisation. Par défaut, ces fichiers comprennent l_intl.nls, c_1252.nls, et c_437.nls.
6. Chargement de la ruche SYSTEM, \Windows\System32\Config\System, et montage de celle-ci dans HKLM \SYSTEM.
7. Lecture de la ruche SYSTEM et repérage de tous les pilotes à charger pour accomplir l'amorçage. Ces pilotes sont indiqués dans le registre par une valeur Start de SERVICE_BOOT_START (0).

Ainsi se termine le rôle de Winload dans le processus d'amorçage, lequel passe alors le relais à la fonction principale de Ntoskrnl.exe (KiSystemStartup) afin de mener plus avant l'initialisation du système.

## Initialisation du noyau

L'initialisation du noyau et des sous-systèmes de l'exécutif se compose de deux phases distinctes, mais fortement couplées. La première phase, appelée phase 0, a pour but la mise en place des structures rudimentaires indispensables pour la suite. La seconde phase, appelée phase 1, donne quant à elle lieu à l'ensemble minimal des services requis pour orchestrer la plateforme (sans toutefois encore la rendre accessible à l'utilisateur).

1. Désactivation des interruptions, évitant ainsi que des événements externes ne viennent interrompre la procédure.
2. Configuration des intervalles d'horloge.
3. Initialisation du gestionnaire d'objets.
4. Initialisation du gestionnaire de processus.
5. Initialisation du gestionnaire Plug and Play.
6. Initialisation de la file DPC du processeur.

Phase 1 :

1. Le pilote vidéo d'amorçage (Bootvid.dll) est appelé pour afficher l'écran de démarrage. Si l'option de démarrage quietboot a été spécifiée, cette étape n'a pas lieu.
2. Initialisation du gestionnaire d'alimentation.
3. Lecture de l'heure système et enregistrement de la valeur retournée en tant qu'heure d'amorçage du système.
4. Sur une machine multiprocesseur, initialisation et démarrage de l'exécution des autres processeurs. Ceci fait, le masque d'affinité du processus System est modifié de sorte qu'il puisse fonctionner sur tous les processeurs.
5. Le gestionnaire d'objets crée le répertoire racine de l'espace de noms (\), le répertoire des dénominations des types d'objets (\ObjectTypes), le répertoire de mappage des noms de périphérique style DOS (\Global??) et le lien symbolique \DosDevices qui pointe vers ce dernier.
6. Le noyau initialise les structures de données de l'ordonnanceur et la table de ventilation (dispatch) des services système.

7. Le moniteur de référence sécurité crée le répertoire \Security dans l'espace de noms du gestionnaire d'objets.

### Gestionnaire de sessions (Smss)

1. Création d'un objet port LPC (\SmApiPort) et de quelques threads chargés à terme de répondre aux requêtes client (par exemple pour créer une session).
2. Définition des liens symboliques pour les noms de périphériques style MS-DOS (COM1, LPT1, etc.).
3. Création du répertoire \Sessions dans l'espace de noms du gestionnaire d'objets.
4. Exécution des programmes définis dans HKLM\SYSTEM\CurrentControlSet\Control\Session Manager \BootExecute. En principe, ladite valeur contient une commande pour exécuter Autochk.
5. Prise en compte des opérations différées de renommage et de suppression de fichier. La configuration de ces commandes se trouve sous HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\PendingFileRenameOperations et HKLM\SYSTEM\ CurrentControlSet\Control\Session Manager \PendingFileRenameOperations2.
6. Création du répertoire \KnownDlls dans l'espace de noms du gestionnaire d'objets, repérage des DLL connues et création pour celles-ci d'objets section au sein dudit répertoire.
7. Ouverture, ou éventuellement création, des fichiers de pagination spécifiés sous HKLM\SYSTEM \CurrentControlSet\ Control\Session Manager\Memory Management\PagingFiles.
8. Chargement de la partie mode noyau du sous-système Windows (Win32k.sys).
9. Démarrage du processus du sous-système Windows, à savoir Csrss. (Les autres sous-systèmes sont configurés pour démarrer à la demande.)
10. Démarrage du processus d'ouverture de session (Winlogon).

### Winlogon

1. Création des objets station fenêtre et bureau initiaux.
2. Exécution de l'autorité de sécurité locale (Lsass.exe)
3. Exécution du gestionnaire de contrôle des services (Services.exe)

### Userinit

Userinit initialise l'environnement de l'utilisateur quand celui-ci ouvre une session.

1. Chargement du profil et traitement des scripts utilisateur.
2. Lancement du shell spécifié dans HKCU\SOFTWARE\Microsoft\Windows NT\Current Version\ Winlogon \Shell. Si cette valeur n'existe pas (ce qui est généralement le cas), Userinit donne lieu au shell spécifié dans HKLM\SOFTWARE\Microsoft\Windows NT\Current Version\Winlogon\Shell, par défaut Explorer.exe.

## BCD

La base de données BCD laisse place à trois types de composants :

- Des magasins (stores)
- Des objets
- Des éléments

Le magasin constitue le grade le plus élevé de la hiérarchie. Dans la perspective BCD, chaque magasin comporte un ensemble d'objets, lesquels renferment à leur tour d'autres objets, appelés dans ce contexte éléments.

Il existe un magasin spécial dit système. La clé de registre HKLM\BCD00000000 agit en tant que duplicata du magasin système.

Plusieurs des outils d'administration standard livrés avec Windows forment un paravent sur le magasin BCD, y compris Configuration du système.

Le moyen le plus commode de configurer le magasin BCD est d'utiliser l'utilitaire de configuration du système Microsoft (Msconfig), dont l'onglet Démarrer fait écho à à la plupart des paramètres en lien avec l'amorçage de l'ordinateur (délai d'attente, journalisation, etc.). une autre manière de gérer le magasin BCD consiste à le faire depuis la ligne de commandes, cela à l'aide de l'utilitaire BCDedit.

**Agir sur le magasin système ou non.** Dans la configuration par défaut, les modifications apportées par BCDedit le sont au niveau du magasin système, c'est-à-dire celui utilisé au démarrage de la machine. Pour interagir sur un BCD non système, recourez à l'option /store, admise dans la plupart des commandes BCDedit, afin de spécifier le BCD à utiliser. L'exemple que voici illustre comment afficher le contenu du BCD enregistré dans un ficher bcdbackup.bcd présent à la racine du disque : bcdedit /enum /store c:\bcdbackup.bcd.

***Tableau 18.1. Options BCD***

| Élément BCD | Signification |
|---|---|
| baudrate | Active le déboggage mode noyau et redéfinit la vitesse la vitesse de transmission du port de communication utilisé pour le débogage noyau distant. |
| bootems | Active ou désactive les services de gestion d'urgence (EMS, Emergency Management Services). |
| bootdebug | Active ou désactive le débogueur de démarrage. |
| burnmemory | Spécifie une quantité de mémoire physique (en Mo) que Windows ne peut pas utiliser. Exemple : burnmemory=128 pour réduire de 128 Mo la mémoire physique pour Windows. Cette option facilite la résolution des problèmes relatifs à l'épuisement de la RAM. |
| bootsequence | Définit une séquence de démarrage unique (comprendre que seul le prochain amorçage du système est concerné, mais pas les suivants). |
| debug | Active le déboggage mode noyau. |
| debugstart | Les options envisageables sont Active, AutoEnable et Disable. |
| firstmegabytepolicy | Définit la stratégie de premier Mo, correspondant à 1 Mo de mémoire que la couche d'abstraction matérielle (HAL) emploie afin de limiter les corruptions causées par le BIOS lors des transitions d'alimentation. |
| groupsize | Controle le nombre maximal de processeurs logiques susceptibles de faire partie d'un groupe (maximum 64). |
| hal | Remplace le nom de fichier par défaut de l'image de HAL (Hal.exe). |
| halbreakpoint | Force HAL à honorer un point d'arrêt au moment de son initialisation. |
| hypervisordebugport | Spécifie le port série employé comme port de débogage. |
| hypervisordisableslat | Contrôle la prise en charge de la fonction de traduction d'adresses de second niveau (SLAT, Second Level Address Translation). |
| kernel | Remplace le nom de fichier par défaut de l'image du noyau (Ntoskrnl.exe). Cette option est utile pour alterner entre un noyau controlé (incluant des codes de débogage) et un noyau libre (version commercialisée normale). |
| keyringaddress | Spécifie l'adresse physique où se trouve le jeu de clés BitLocker. |
| lastknowngood | Amorce le système avec la dernière bonne configuration connue, nonobstant le jeu de contrôles actuel. |
| nocrashautoreboot | Désactive le redémarrage automatique sur incident. |
| quietboot | Empêche l'initialisation du pilote vidéo VGA chargé de l'affichage des informations de progression au démarrage de Windows. |
| recoverysequence | Définit la séquence de récupération à employer. |

| Élément BCD | Signification |
|---|---|
| relocatephysical | Addresse physique à laquelle doit être redirigé la mémoire physique d'un noeud d'accès mémoire non uniforme (NUMA) automatiquement sélectionné. |
| truncatememory | Force Windows à ne pas utiliser la mémoire physique au-delà d'un seuil spécifié (en Mo). |
| msi | Détermine le niveau de prise en charge des interruptions signalées par message. |
| maxgroup | Maximise le nombre de groupes de processeurs créés lors de la configuration de la topologie NUMA. |
| nolowmem | Force Windows à ne pas utiliser les quatre premiers Go de mémoire physique. Exige la version PAE du noyau Windows et que l'ordinateur soit pourvu de plus de 4 Go de mémoire. |
| numproc | Spécifie le nombre de processeurs utilisables sur un configuration multi processeur. Exemple : numproc=2 sur une machine quadri processeur force Windows à n'utiliser que deux des quatre processeurs disponibles. |
| nx | Contrôle l'état de la protection non exécution. |
| optionsedit | Active l'éditeur d'options du gestionnaire de démarrage. Avec cette option, le système permet à l'utilisateur de spécifier de manière interactive des options de ligne de commande et des commutateurs pour le démarrage actuel. |
| osdevice | Spécifie le périphérique sur lequel le système d'exploitation est installé. |
| safeboot | Spécifie les paramètres d'amorçage en mode sans échec. Les options suivantes sont disponibles : Minimal, Network, DsRepair.- Les flags Minimal et Network correspondent respectivement au mode sans échec standard et au mode sans échec avec prise en charge réseau. Le flag DsRepair est utilisé pour les contrôleurs de domaine uniquement. |
| safebootalternateshell | Indique à Windows d'utiliser le programme spécifié par l'intermédiaire de la clé HKLM\SYSTEM\CurrentControlSet \Control\SafeBoot\Option\UseAlternateShell en tant que shell de l'utilisateur (Cmd.exe par défaut). Correspond à l'entrée "mode sans échec avec invite de commandes" dans le menu Options de démarrage avancées. |
| sos | Affiche les noms des pilotes de périphériques au fur et à mesure de leur chargement. |
| systemroot | Spécifie le chemin d'accès au répertoire dans lequel Windows est installé. |
| usefirmwarepcisettings | Empêche Windows d'assigner dynamiquement des ressources IO/IRQ aux périphériques PCI et laisse lesdits périphériques configurés par le BIOS. |
| vga | Force Windows à utiliser le pilote VGA standard pour les opérations en mode graphique. Cette option est surtout utile en vue de supprimer ou mettre à jour un pilote vidéo posant problème ou revenir à sa version précédente. |
| xsavedisable | Désactive la prise en charge de la fonctionnalité XSAVE. |
| xsavepolicy | Force le système (Hwpolicy.sys) à adopter la stratégie XSAVE donnée. |
| x2apicpolicy | Contrôle l'utilisation du mode APIC étendu. Le système utilise par défaut le mode APIC étendu s'il est disponible. |

*Tableau 18.2. Objets BCD*

| GUID | Nom symbolique | Nom convivial |
|---|---|---|
| {0CE4991B-E6B3-4B16-B23C--5E0D9250E5D9} | GUID_EMS_SETTINGS_GROUP | {emssettings} |
| {1AFA9C49-16AB-4A5C-4A90--212802DA9460} | GUID_RESUME_LOADER_SETTINGS_-GROUP | {resumeloadersettings} |

| GUID | Nom symbolique | Nom convivial |
|---|---|---|
| {1CAE1EB7-A0DF-4D4D-9851--4860E34EF535} | GUID_DEFAULT_BOOT_ENTRY | {default} |
| {313E8EED-7098-4586-A9BF--309C61F8D449} | GUID_KERNEL_DEBUGGER_SETTINGS_-GROUP | {kerneldbgsettings} |
| {4636856E-540F-4170-A130--A84776F4C654} | GUID_DEBUGGER_SETTINGS_GROUP | {dbgsettings}<br>{eventsettings} |
| {466F5A88-0AF2-4F76-9038--095B170DC21C} | GUID_WINDOWS_LEGACY_NTLDR | {legacy}<br>{ntldr} |
| {5189B25C-5558-4BF2-BCA4--289B11BD29E2} | GUID_BAD_MEMORY_GROUP | {badmemory} |
| {6EFB52BF-1766-41DB-A6B3--0EE5EFF72BD7} | GUID_BOOT_LOADER_SETTINGS_GROUP | {bootloadersettings} |
| {7254A080-1510-4E85-AC0F--E7FB3D444736} | GUID_WINDOWS_SETUP_EFI | |
| {7EA2E1AC-2E61-4728-AAA3--896D9D0A9F0E} | GUID_GLOBAL_SETTINGS_GROUP | {globalsettings} |
| {7FF607E0-4395-11DB-B0DE--0800200C9A66} | GUID_HYPERVISOR_SETTINGS_GROUP | {hypervisorsettings} |
| {9DEA862C-5CDD-4E70-ACC1--F32B344D4795} | GUID_WINDOWS_BOOTMGR | {bootmgr} |
| {A1943BBC-EA85-487C-97C7--C9EDE908A38A} | GUID_WINDOWS_OS_TARGET_TEMPLATE_-PCAT | |
| {A5A30FA2-3D06-4E9F-B5F4--A01DF9D1FCBA} | GUID_FIRMWARE_BOOTMGR | {fwbootmgr} |
| {AE5534E0-A924-466C-B836--758539A3EE3A} | GUID_WINDOWS_SETUP_RAMDISK_-OPTIONS | {ramdiskoptions} |
| {B012B84D-C47C-4ED5-B722--C0C42163E569} | GUID_WINDOWS_OS_TARGET_TEMPLATE_-EFI | |
| {B2721D73-1DB4-4C62-BF78--C548A880142D} | GUID_WINDOWS_MEMORY_TESTER | {memdiag} |
| {CBD971BF-B7B8-4885-951A--FA03044F5D71} | GUID_WINDOWS_SETUP_PCAT | |
| {FA926493-6F1C-4193-A414--58F0B2456D1E} | GUID_CURRENT_BOOT_ENTRY | {current} |

## Modes de démarrage

Windows offre trois modes de démarrage :

- *Démarrage normal* Sert pour le fonctionnement normal du système. Dans ce mode, le système charge tous les pilotes de périphérique, tous les services automatiques, et exécute toutes les applications de démarrage.
- *Démarrage en mode diagnostic* Sert pour le dépannage du système ou la mise en oeuvre d'activités de réponse à un incident. Dans ce mode, le système ne charge que les pilotes et services indispensables, cela afin de les éliminer des causes probables d'un dysfonctionnement.
- *Démarrage sélectif* Sert pour l'identification des causes de problèmes. Dans ce mode, le système s'appuie sur une configuration de démarrage personnalisée selon une méthode sélective des services et autres éléments de démarrage. Les onglets Services et Démarrage permettent d'activer/désactiver les services et applications concernés.

Windows emploie par défaut l'option Démarrage normal. Voici la procédure pour utiliser un autre mode de démarrage.

1. Démarrez Configuration du système.

2. Dans l'onglet Démarrage, cochez Démarrage en mode diagnostic ou Démarrage sélectif. L'option Démarrage sélectif s'accompagne de différents choix se ramenant aux éléments à considérer lors de l'amorçage.

   - *Charger les services système* Demande au système de charger les services Windows au démarrage. Utilisez dans cette éventualité les paramètres de l'onglet Services pour spécifier les services à démarrer.

   - *Charger les éléments de démarrage* Demande au système d'exécuter les applications censées démarrer à l'amorçage. Utilisez dans cette éventualité les paramètres de l'onglet Démarrage en vue d'activer ou désactiver les applications.

   - *Utiliser la configuration de démarrage d'origine* Force le système à prendre en compte la configuration de démarrage originelle, et non celle issue de modifications antérieures.

3. Cliquez sur le bouton OK puis réamorcez l'ordinateur.

## Options d'amorçage

L'onglet Démarrer de Configuration du système permet de sélectionner un système d'exploitation au démarrage et de lui appliquer certains paramètres, cela en vue d'en préparer la personnalisation ou le dépannage. Les options suivantes concernent directement la manipulation des rubriques de système d'exploitation (emplacements des partitions d'amorçage et méthodes d'amorçage utilisées).

- *Par défaut* Fait de la partition d'amorçage sélectionnée la partition par défaut.

- *Délai* Définit la durée pendant laquelle Windows attend en affichant le menu de démarrage avant d'utiliser la partition d'amorçage par défaut.

- *Supprimer* Supprime une rubrique de système d'exploitation. (Sur un ordinateur doté d'un seul système d'exploitation, les boutons Par défaut et Supprimer sont estompés, les actions correspondantes n'étant pas permises.)

- *Options avancées* Configure la prise en compte du matériel de l'ordinateur, dont par exemple le nombre de processeurs, la quantité de mémoire vive, les verrous PCI, HAL.

Les options de démarrage suivantes sont définies.

- *Démarrage sécurisé* Démarre l'ordinateur en mode sécurisé avec des paramètres supplémentaires permettant de choisir entre Minimal, Réseau, Autre environnement et Réparer Active Directory. Le mode minimal est équivalent à un mode sans échec sans prise en charge du réseau ; le mode réseau à un mode sans échec avec prise en charge du réseau ; le mode mode autre environnement à une invite de commande en mode sans échec sans prise en charge du réseau ; et enfin, le mode réparer, au mode de restauration des services d'annuaire (dsrepair).

- *Ne pas démarrer l'interface utilisateur graphique* Démarre l'ordinateur en mode caractère, sans charger les composants graphiques du système d'exploitation. Dans cette configuration, Windows présente à l'utilisateur une invite de commande.

- *Vidéo de base* Démarre l'ordinateur en mode VGA et force Windows à employer le pilote VGA minimal au lieu des pilotes spécifiques à la carte vidéo installés sur la machine.

- *Journaliser le démarrage* Consigne dans un journal les événements clé du démarrage. Le fichier résultant de la journalisation de l'amorçage est par défaut %SystemRoot%\Ntbtlog.txt.

- *Informations sur le démarrage du système* Affiche des informations qui permettent de voir le détail des opérations impliquées dans le processus de démarrage, par exemple le nom des pilotes chargés.

## ReadyBoot

L'extraction anticipée intelligente (comprendre le prefetcher logique standard) n'est pas la seule méthode qu'emploie Windows en vue d'accélérer le processus de démarrage. Ainsi, lorsque la configuration le permet (700 Mo ou plus de RAM), le système d'exploitation s'appuie pendant la phase d'amorçage sur un cache situé en RAM.

A l'issue de la phase de démarrage du système, le service ReadyBoost calcule un plan de mise en cache pour le démarrage suivant, ce sur la base des informations de suivi de fichier des cinq démarrages précédents. Les données enregistrées pour l'occasion le sont en tant que fichiers d'extension .fx dans le répertoire \Windows \Prefetch\Readyboot, tandis que le plan de cache lui même est conservé dans le registre sous la clé HKLM \SYSTEM\CurrentControlSet\Services\Rdyboost\Parameters.

## Modification du délai d'affichage

Il est possible de modifier la valeur du délai d'affichage associé au système d'exploitation par défaut à l'aide du commutateur /timeout de la commande bcdedit. Fixez la valeur au délai souhaité (exprimé en secondes) comme suit :

```
bcdedit /timeout 15
```

Pour démarrer automatiquement sur le système d'exploitation par défaut, régler le délai à 0 secondes.

## Modification temporaire de la séquence de démarrage

Il peut, en certaines circonstances, être utile d'amorcer un système d'exploitation particulier une fois avant de revenir à l'ordre de démarrage par défaut. Pour ce faire, employez le commutateur /bootsequence de la commande bcdedit, suivi de l'identificateur du système d'exploitation à mettre en route après avoir relancé l'ordinateur.

Au redémarrage de l'ordinateur, celui-ci sélectionnera le système d'exploitation spécifié uniquement pour cette session. Lors de tout démarrage ultérieur, Windows emploiera l'ordre de démarrage par défaut original.

## Afficher le contenu du BCD

Pour afficher la liste des entrées que renferme le magasin des données de configuration de démarrage, tapez simplement bcdedit dans une invite de commandes élevée. Vous pouvez également utiliser la syntaxe suivante :

```
bcdedit /enum
```

Dans l'éventualité où l'ordinateur ne contient qu'un seul système d'exploitation, vous devriez voir s'afficher deux entrées :

- Gestionnaire de démarrage Windows
- Chargeur de démarrage Windows

Si vous souhaitez afficher les identificateurs sous leur forme complète, utilisez le commutateur /v.

Pour voir les entrées BCD pour tous les utilitaires de démarrage et les valeurs des paramètres, saisissez bcdedit /enum all. Cette commande donne la liste en mode prolixe de l'ensemble des entrées BCD quel que soit leur état.

- *Reprendre à partir de la mise en veille prolongée* L'entrée Reprendre à partir de la mise en veille prolongée fait le lien avec la configuration actuelle du dispositif de reprise. L'utilitaire de démarrage de pré-système d'exploitation en charge du contrôle de la reprise est généralement Winresume (\\Windows

\System32\Winresume.exe). Les données de veille prolongée, comme spécifié dans le paramètre FilePath, sont hébergées dans le fichier Hiberfil.sys situé dans le répertoire racine du périphérique où Windows est installé (paramètre OSDevice, correspondant au volume C dans la plupart des configurations).

- *Windows Legacy OS loader* L'entrée Windows Legacy OS loader rend visible la configuration actuelle pour le chargement de versions antérieures de Windows. Le paramètre Device fait référence à la partition par défaut à employer, par exemple C:, et le paramètre Path définit le chemin par défaut vers l'utilitaire de chargement, comme Ntldr.

- *Paramètres EMS* L'entrée Paramètres EMS procure une vue sur les paramètres de configuration employés lors du démarrage avec les services de gestion d'urgence (EMS, Emergency Management Services).

- *Paramètres du débogueur* L'entrée Paramètres du débogueur montre les options en vigueur lorsque le démarrage du système se déroule sous le contrôle d'un débogueur. Vous pouvez voir quels sont les paramètres de débogage actifs en entrant bcdedit /debugsettings.

## Performances de démarrage

### Causes courantes de mauvaises performances de démarrage

Cette section traite des différents facteurs susceptibles de dégrader les performances de démarrage du système, présentant pour chaque problème les outils et les méthodes les mieux adaptés.

- *Services ne signalant pas leur démarrage* Dès lors que le contrôleur de services (SCM) envoie une commande de démarrage, il est attendu que le service concerné réponde à la requête dans les plus brefs délais. Un service qui ne répond pas immédiatement à une demande de démarrage oblige le système à attendre qu'un délai imparti soit écoulé.

- *Stratégies de groupe synchrones* Les stratégies de groupe peuvent quand elles sont appliquées de manière synchrone entrainer des retards plus ou moins longs affectant d'une part le démarrage de l'ordinateur, d'autre part et par la suite le processus d'ouverture de session utilisateur. Si vous implémentez des stratégies de groupe, considérez l'utilisation du mode asynchrone.

- *Analyseurs de démarrage* Certaines solutions de sécurité informatique (antivirus notamment) intègrent des fonctions d'analyse qui peuvent ralentir considérablement les performances de démarrage du système. Ces fonctions, implémentées pour la plupart en tant que pilotes de filtre, peuvent être surveillées à l'aide d'une trace de démarrage ETL (Event Trace Log).

- *Superfetch désactivé* Superfetch est un procédé de gestion de la mémoire destinée à améliorer la vitesse d'ouverture et le temps de réponse des programmes les plus utilisés. Si le service Superfetch est désactivé, de telles optimisations ne sont dès lors plus possibles.

- *Retards réseau* Des retards peuvent être induits par un réseau ou une carte réseau momentanément indisponible, et encore par des connexions à des ressources telles que les disques, les profils itinérants ou les stratégies de groupe (voir point plus haut).

- *Lenteur support de stockage* Les performances constatées lors du démarrage de l'ordinateur dépendent le plus souvent des performances du disque (dont découle la vitesse à laquelle le système d'exploitation manipule les fichiers nécessaires pour l'amorçage). Si une telle éventualité est envisageable, optez pour une unité de stockage de type SSD, plus rapide et plus fiable qu'un disque dur traditionnel.

- *Trop d'éléments de démarrage* Utilisez des outils tels que Autoruns pour détecter et gérer tous les pilotes, services et autres applications (par exemple, images configurés pour démarrer automatiquement) exécutés au démarrage ou suite à une ouverture de session.

## Débusquer les applications lancées au démarrage

### Dossier Démarrage

Pour afficher les raccourcis d'applications configurées pour s'exécuter au démarrage, ouvrez le contrôle Exécuter, saisissez la commande shell:startup puis validez. Le dossier Démarrage s'ouvre alors.

### Registre

Sollicitez l'Éditeur du Registre (ou tout autre utilitaire du même genre) et développez les entrées HKCU, Software, Microsoft, Windows, CurrentVersion, en enfin Run. Pour désactiver une entrée, effectuez un clic secondaire sur son nom dans le volet droit puis sélectionnez Supprimer dans le menu contextuel.

### Désactiver les programmes de démarrage via l'application Paramètres

Dans l'application Paramètres, cliquez sur l'icône Applications puis sélectionnez la section Démarrage, laquelle regroupe les logiciels et applications s'exécutant automatiquement lors de la connexion. Utilisez les interrupteurs pour désactiver chacune des applications dont vous estimez ne pas avoir besoin.

### Désactiver les programmes de démarrage via le gestionnaire de tâches

Dans la fenêtre du Gestionnaire des Tâches, sélectionnez l'onglet Démarrage. Les applications lancées au démarrage s'affichent par ordre alphabétique. Pour désactiver une des applications listées, cliquez dessus avec le bouton droit de la souris puis sélectionnez Désactiver dans le menu contextuel nouvellement apparu. (En fonction de l'utilité du programme en question, vous avez également la possibilité de le supprimer de façon classique.)

Les informations montrées par l'onglet Démarrage du Gestionnaire de Tâches le sont sous forme de colonnes. L'organisation par défaut affiche quatre colonnes, avec le nom, l'éditeur, l'état et l'impact du démarrage pour chaque application de démarrage indiquée.

- *Nom* Affiche le nom de l'application. Cette colonne est la seule qui ne puisse être masquée.
- *Éditeur* – Affiche le nom de l'entité (personne, association, société ou organisme) ayant publié le logiciel.
- *Statut* Indique si un programme est activé ou désactivé pendant la procédure de démarrage.
- *Impact du démarrage* Donne une idée générale de l'importance de l'impact de chaque programme: élevé (marqué comme Haut), moyen, faible (marqué comme Bas) ou nul (marqué comme Aucun). Toute nouvelle application ajoutée à la liste affiche initialement l'état Indéterminé, ce jusqu'à ce que suffisamment d'informations le concernant aient été accumulées pour l'inclure dans l'une des catégories susmentionnées.

Les autres colonnes susceptibles d'être ajoutées sont :

- *Type de Démarrage* Indique la manière dont l'application est programmée pour démarrer, soit en utilisant soit le Registre, soit le dossier de démarrage de Windows.
- *E/S disque au démarrage* Quantité d'octets (en Mo) requises par les E/S émanant de l'application lors de son démarrage.
- *UC au démarrage* Indique le temps de fonctionnement du processeur requis par l'application au démarrage.
- *En cours d'exécution* Signale si l'application se trouve ou non en cours d'exécution.
- *Ligne de commande* Spécifie la ligne de commande ayant accompagné l'application.

## Options d'amorçage BCD

Options du gestionnaire de démarrage (Bootmgr) :

- **bcddevice** Périphérique (sous la forme GUID) où réside la base de données de configuration de démarrage.
- **bcdfilepath** Chemin vers la base de données de configuration de démarrage (habituellement \Boot\BCD).
- **customactions** Actions personnalisées à effectuer lorsqu'une séquence de clavier spécifique a été saisie.
- **displaybootmenu** Détermine si le gestionnaire de démarrage affiche le menu de démarrage, où se présente un choix entre différentes options comme comme le démarrage normal, le mode sans échec, etc., ou si Windows démarre automatiquement en utilisant l'option par défaut définie.
- **displayorder** Ordre dans lequel le gestionnaire de démarrage affiche le menu de démarrage multiple.
- **keyringaddress** Adresse physique où réside sur disque lu jeu de clés BitLocker.
- **noerrordisplay** Réduit au silence la sortie des erreurs rencontrées lors du processus de démarrage.
- **resume** Dicte au système d »exploitation s'il faut ou non qu'il effectue une reprise à partir de l'hibernation lors du démarrage.
- **resumeobject** Application de démarrage (sous la forme GUID) en charge de la sortie de veille prolongée (généralement Winresume.exe).
- **timeout** Laps de temps (en secondes) durant lequel le menu de démarrage est affiché avant que le système ne choisisse automatiquement l'option par défaut.
- **toolsdisplayorder** Ordre dans lequel le gestionnaire de démarrage affiche le menu des outils.

Options du chargeur de démarrage (Winload) :

- **advancedoptions** Contrôle la manière dont Windows réagit en cas d'échec de démarrage, soit en tentant automatiquement une récupération, soit en offrant à l'utilisateur la possibilité d'accéder à des options de démarrage avancées pour une intervention manuelle.
- **bootlog** Journalise les actions effectuées lors de la procédure d'amorçage dans le fichier %SystemRoot% \Ntbtlog.txt.
- **bootux** Expérience graphique pour accompagner le processus de démarrage : aucune (écran noir), minimale (barre de progression), ou standard (animation du logo Windows).
- **bootstatuspolicy** Contrôle de quelle façon le chargeur de démarrage se comporte vis-à-vis de l'affichage des erreurs.
- **clustermodeaddressing** Nombre maximal de processeurs susceptibles d'être regroupés ensemble dans une même unité de traitement pour la gestion des interruptions dans un système multiprocesseur.
- **configflags** Valeurs de configuration spécifiques au processeur se rapportant à des fonctionnalités telles la gestion de la puissance ou d'autres paramètres de performance.
- **debug** Active le débogage mode noyau.
- **dbgtransport** Remplace les transports de débogage du noyau par défaut (Kdcom.dll, Kd1394, Kdusb.dll) par un fichier spécifique, permettant ainsi l'utilisation de méthodes de débogage non pris en charge par la configuration standard.

- **hal** Remplace le fichier par défaut (hal.dll) pour la couche d'abstraction matérielle.
- **perfmem** Taille du tampon réservé pour la journalisation des performances.
- **nointegritychecks** Désactive les contrôles d'intégrité qu'effectue Windows lors du chargement des pilotes. Supprimé automatiquement au prochain redémarrage.
- **onecpu** Force Windows à n'utiliser qu'un seul processeur parmi les multiples disponibles sur un système multi processeur.

Options pour applications de démarrage :

- **avoidlowmemory** Force le système à ignorer autant que possible les adresses physiques en deçà d'une valeur donnée.
- **badmemoryaccess** Force l'utilisation des pages de mémoire reconnues défectueuses.
- **channel** Canal IEEE 1394 par lequel transiteront les communications de débogage noyau. Il peut s'agir de n'importe quel nombre compris entre 0 (valeur par défaut) et 62.
- **debugaddress** Adresse matérielle du port série (COM) employé pour le débogage.
- **debugport** Active le déboggage mode noyau et redéfinit le port série auquel est connecté un hôte de debogeur noyau à distance. Par défaut, Windows emploie le second port série (COM2), s'il existe, le premier (COM1) dans le cas contraire.
- **debugtype** Méthode utilisée pour interagir avec le débogueur noyau, indiquant si la communication s'effectue via port série (cas par défaut), FireWire ou USB.
- **emsbaudrate** Taux de transmission à utiliser pour les communications EMS (Emergency Management Services).
- **emsport** Port série (COM) en conjonction des services EMS.
- **extendedinput** Offre aux applications de démarrage la possibilité de recourir aux fonctionnalités d'entrée étendues du BIOS.
- **firstmegabytepolicy** Contrôle la manière dont est géré le premier mégaoctet de la mémoire physique pendant les phases de démarrage et de reprise du système.
- **graphicsmodedisabled** Contrôle la capacité des applications de démarrage à exécuter des opérations graphiques. Cette option peut s'avérer utile dans des circonstances spécifiques où restreindre l'utilisation des ressources lors du démarrage du système est nécessaire, ou lorsque des problèmes de compatibilité ou de stabilité surviennent avec des applications graphiques au démarrage.
- **graphicsresolution** Résolution d'affichage utilisée pendant le processus de démarrage du système d'exploitation.
- **integrityservices** Gère la vérification de l'intégrité du code s'exécutant en mode noyau, assurant ainsi que seuls les pilotes et les composants système signés sont autorisés à être chargés.
- **locale** Langue et autres paramètres régionaux (format de date, heure, devises, etc.).
- **noumex** Ignore les exceptions en provenance du mode utilisateur tandis que le débogage noyau est activé. aidant ainsi à isoler plus facilement les problèmes liés strictement aux sphères privilégiés du système.
- **novesa** Empêche l'utilisation des modes d'affichage VESA.

- **recoveryenabled** Ouvre la possibilité d'une série d'actions que Windows peut entreprendre pour remettre le système en état après un problème. Cela peut inclure des tentatives de réparation automatique, le retour à un point de restauration précédent, ou le démarrage en mode sans échec.
- **targetname** Nom de la cible dans l'éventualité d'un débogage par USB.
- **testsigning** Contrôle la manière dont le système autorise le chargement de pilotes non signés ou signés localement.

## Arrêt

Du point de vue du sous-système Windows, la phase d'extinction de l'ordinateur commence dès l'entrée dans ExitWindowsEx, dont émane en premier lieu un message RPC demandant à Csrss de prendre en charge l'arrêt. Csrss commence par satisfaire partiellement cette requête en empruntant l'identité de l'appelant, puis en envoyant un message à Winlogon. Winlogon prend alors l'identité de l'utilisateur courant (qui peut ou non disposer du même contexte de sécurité que l'utilisateur ayant initié l'arrêt) et sollicite ExitWindowsEx à une seconde reprise. À nouveau, cet appel entraine l'envoi d'un message à Csrss.

Une fois établi que la demande de fermeture du système provient de Winlogon, Csrss parcourt tous les processus de la session de l'utilisateur interactif dans l'ordre inverse de leur *niveau d'arrêt*, à quoi correspond un niveau de priorité qu'emploie le système pour déterminer quand un processus peut se terminer vis-à-vis des autres. Les processus peuvent influencer l'ordre dans lequel ils se terminent spécifiant un niveau d'arrêt à l'aide de la fonction SetProcessShutdownParameters. Les niveaux d'arrêt valides vont de 0 à 1023, où 0 est le plus élevé (terminé en dernier) et 1023 est le plus bas (terminé en premier). Par exemple, un processus avec un niveau d'arrêt de 42 sera arrêté après un processus avec un niveau de 666.

Csrss gère la fermeture des processus qui possèdent des fenêtres en envoyant des messages spécifiques pour vérifier s'ils peuvent se terminer, puis en les attendant pendant un délai configuré avant de forcer leur fermeture si nécessaire. Premièrement, Csrss envoie le message `WM_QUERYENDSESSION` à chaque thread du processus auquel se rattache une boucle de messages Windows. Si le thread répond en retournant TRUE, signe qu'il est prêt à se fermer immédiatement, alors l'arrêt du système peut continuer. Csrss envoie ensuite le message `WM_ENDSESSION` au thread, lui ordonnant de se terminer. Csrss attend alors un certain nombre de secondes (défini par la valeur de registre HungAppTimeout sous HKCU\Control Panel\Desktop, par défaut 5 000 millisecondes, soit 5 secondes) pour que l'opération ait lieu - c'est-à-dire pour que le thread se termine. Dans l'éventualité où le thread ne se termine pas avant expiration du délai imparti, Csrss assombrit l'écran et affiche une notification visuelle détaillant les programmes en cause, offrant par la même occasion à l'utilisateur la possibilité de forcer leur arrêt ou d'annuler la demande de fermeture du système. Dans le cas où le thread se termine effectivement avant la fin du délai, Crsss continue d'envoyer les paires de messages aux autres threads concernés. Une fois que tous ont pris fin, Crsss termine le processus dont ils dépendent et passe au processus suivant dans l'ordre convenu plus tôt.

Pour les applications à interface textuelle (processus console), Csrss envoie au gestionnaire de contrôle de la console idoine le code `CTRL_LOGOFF_EVENT`. Si le gestionnaire répond FALSE, cela indique que l'application refuse de se fermer immédiatement. En réponse, Csrss met sans autre forme de procès fin au processus récalcitrant. Autrement, si le gestionnaire répond TRUE ou ne répond pas avant le délai spécifié dans HKCU\Control Panel \Desktop\WaitToKillAppTimeout (par défaut 20 000 millisecondes, soit 20 secondes), Windows affiche un écran indiquant que certains programmes doivent encore se fermer.

L'étape suivante intervient après que tous les processus de la session de l'utilisateur interactif se soient terminés, et occasionne un passage de relais de Winlogon à Wininit, plus haut dans la hiérarchie des composants que Windows met en oeuvre. En la circonstance, Wininit appelle à nouveau ExitWindowsEx, mais cette fois dans le contexte du processus système, ce qui envoie un message RPC au processus Csrss qu'héberge la session 0 sur le système. Conséquemment, Csrss recherche tous les processus appartenant au contexte système et envoie la combinaison `WM_QUERYENDSESSION`/`WM_ENDSESSION` aux threads ayant une interface graphique. (Notez que, conformément aux bonnes pratiques, aucune application GUI ne devrait être hébergée, même temporairement, dans la session 0.) Au lieu d'envoyer `CTRL_LOGOFF_EVENT`, cependant, il envoie le code

`CTRL_SHUTDOWN_EVENT` aux applications console ayant des gestionnaires de contrôle déclarés, incluant le gestionnaire de contrôle des services (SCM) et par extension tous les services qui se sont inscrits pour recevoir les notifications d'arrêt.

Contrairement aux processus utilisateur, dont la fin peut être brutalement précipitée s'ils ne répondent pas dans les délais prescrits, les processus système, dotés d'un rôle crucial dans le maintien de la stabilité du système, ne sont pas soumis à des exigences aussi drastiques. En réalité, nombre d'entre eux restent même en cours d'exécution jusqu'à l'arrêt complet de l'ordinateur : Smss, Wininit, Services, Lsass et d'autres. Dans les faits, si l'ou ou l'autre de ces processus, tous considérés critique, devait s'arrêter avant que le système ne soit mis hors tension, le noyau Windows provoquerait un effondrement du système (code `CRITICAL_OBJECT_TERMINATION`), opération qui pourrait causer divers autres problèmes liés à l'arrêt inattendu du système.

Après notification au processus exécutés dans la session 0 que le système est en train de s'arrêter, Winlogon parachève le processus d'arrêt en passant la main à la fonction NtShutdownSystem, qui appelle elle-même NtSetSystemPowerState pour coordonner l'arrêt des pilotes et du reste des sous-systèmes de l'exécutif :

- Envoi de paquets d'E/S d'arrêt depuis le gestionnaire d'E/S vers les pilotes de périphériques ayant demandé la notification des arrêts, leur laissant ainsi une chance de mener à bien d'éventuelles tâches spéciales avant que Windows ne cesse d'occuper l'ordinateur, par exemple démanteler des ressources, sauvegarder des données, remettre le matériel à un état spécifique, et ainsi de suite.
- Écriture des pages de mémoire modifiées contenant des données fichier vers leurs fichiers respectifs sur le volume de stockage, évitant de la sorte toute perte de données.
- Enregistrement des modifications apportées au registre vers les fichiers ruche correspondants.
- Notification aux pilotes de systèmes de fichiers de l'arrêt imminent du système afin qu'ils accomplissent les mesures nécessaires. Cela inclut notamment le fait de sauvegarder les données en mémoire cache ou modifiées vers les périphériques de stockage sous-jacents.

En dernier lieu, l'arrêt du système se conclut dans le gestionnaire d'alimentation, qui finalise le processus en mettant l'ordinateur hors tension, en le redémarrant, etc.

| **Fonctions** | **Services** | **Routines** |
|---|---|---|
| ExitWindowsEx | | |
| GetProcessShutdownParameters | | |
| SetProcessShutdownParameters | | |
| | NtShutdownSystem | |
| SetSystemPowerState | NtSetSystemPowerState | |

| **Élément** | **Lieu** | **Autres références** |
|---|---|---|
| ShutdownFlags | CSR_PROCESS | SetProcessShutdownParameters |
| ShutdownLevel | CSR_PROCESS | SetProcessShutdownParameters |

# Chapitre 19. Gestionnaire de cache

À la lisière du gestionnaire de mémoire et des dispositifs bas niveau contrôlant processus et stockage, le gestionnaire de cache offre des fonctionnalités de cache de données à tous les pilotes de système de fichiers Windows (tant locaux que réseau). Dans ce chapitre, nous expliquerons en quoi consiste les services de mise en cache : quelles dimensions ils revêtent, quelles structures les représentent, quelles fonctions les animent, et ainsi de suite.

## Fonctionnalités clé du gestionnaire de cache

Cette section couvre les fonctionnalités majeures qui voient le jour au sein du gestionnaire de cache.

- *Centralisation des techniques de mise en cache* Le gestionnaire de cache fournit des services globaux de cache à NTFS et aux autres pilotes de système de fichiers, que ces pilotes soient locaux ou de réseau (serveurs et redirecteurs). Ledit composant expose pour ce faire au gestionnaire de mémoire virtuelle une interface spécialisée, qui plus est cohérente à l'échelle du système, dont l'objectif premier est d'éviter à chaque système de fichiers de mettre en oeuvre ses propres schémas de cache.

- *Optimisation des E/S* Quand un processus tente d'accéder à une partie de fichier qui n'est pas dans le cache, le gestionnaire de mémoire virtuelle appelle le pilote de système de fichiers approprié, lequel sollicite en retour le pilote disque de sorte à obtenir le contenu du fichier à partir du volume idoine. Le gestionnaire de cache optimise dès ce moment les entrées/sorties disque en utilisant un ensemble de threads système dévolues à transférer, en tache de fond (et toujours par l'intermédiaire du gestionnaire de mémoire), les contenus stratégiques du disque vers le cache.

### Cache système unifié et centralisé

Un aspect particulièrement remarquable du gestionnaire de cache Windows tient aux relations qu'il entretient avec d'autres d'autres composants fondamentaux du système, dont notamment le gestionnaire d'E/S. Tandis que certains systèmes d'exploitation s'appuie sur chaque système de fichiers individuel pour mettre en cache les données, ce qui implique de dupliquer du code, Windows prend le contre-pied de cette approche et offre un système de cache centralisé capable de traiter des données indépendamment de leur type (contenu d'un fichier, activités d'E/S, méta données de système de fichiers, etc.) et du support qui les accueille (disques durs locaux, serveurs réseau, lecteurs optiques, etc.).

### Cache basé sur les blocs virtuels

Maints systèmes de cache opèrent sur la base de blocs logiques, et ont de cette façon conscience de quels blocs d'une partition disque se trouvent dans le cache. Windows emploie, par contraste, une méthode dite *cache basé sur les blocs virtuels*, fondée en l'occurence sur des fichiers plutôt que sur des blocs bruts. Dans cette approche, le gestionnaire de cache sait quelles parties de quels fichiers sont dans le cache. Cette façon de procéder offre deux avantages majeurs :

Cette façon de procéder offre deux avantages majeurs :

- Elle ouvre la possibilité de prédire les sous-ensembles d'un fichier qui ont le plus de chance d'être sollicités lors des opérations d'E/S ultérieures, concrétisant de la sorte ce que l'on appelle des lectures anticipées intelligentes. (Bien que les lectures anticipées soit possible avec la mise en cache de blocs logiques, la mise en cache de blocs virtuels rend le procédé plus simple à implémenter et plus flexible.)

- Pour des requêtes concernant des données qui sont déjà dans le cache, elle permet au système d'E/S d'éviter le recours au système de fichiers (Fast I/O). De telles requêtes peuvent ainsi être satisfaites plus rapidement et de façon plus efficace.

### Cohérence du cache

Une fonction majeure d'un agent de mise en cache est d'assurer que tout processus accédant à des données du cache lise la version la plus récente de ces données. Du point de vue agnostique (comprendre non Windows), une telle exigence parait difficile à satisfaire, d'autant plus si l'on considère le cas où plusieurs processus interagissent avec le même fichier. En réalité, ce problème potentiel ne se produit pas sous Windows ; cela dans la mesure où tant le gestionnaire de cache que les autres composants, y compris les applications utilisateur, s'appuient sur les mêmes services de mappage de fichier du système de gestion de la mémoire. Comme le gestionnaire de mémoire garantit qu'il n'a qu'une seule représentation de chaque fichier mappé (quel que soit le nombre de vues mappées), il mappe toutes les vues d'un fichier vers un même ensemble de pages en mémoire physique. C'est donc en définitive l'étroite surface de contact entre fonctionnalités mémoire et fonctionnalités de cache qui permet de réaliser (et de maintenir) la cohérence du cache.

### Cache et systèmes de fichiers récupérables

L'un de principaux atouts du gestionnaire de cache tient à la compatibilité qu'il offre vis-à-vis des systèmes de fichiers récupérables, tel NTFS, conçus pour reconstruire la structure des volumes disque après un incident système. De telles fonctionnalités impliquent que toutes les opérations susceptibles d'altérer la structure du système de fichiers (allocation d'espace pour de nouveaux fichiers, redimensionnement de fichiers, etc.) soient implémentées de manière atomique, et s'effectuent de la sorte soit complètement, soit pas du tout. En vue de prendre en charge cette caractéristique, un système de fichiers récupérable gère un fichier journal dans lequel il enregistre chaque modification qu'il entend apporter avant d'écrire la modification sur le volume. Dans l'éventualité d'une défaillance, le système est par ce biais en mesure d'exploiter les informations du journal pour procéder aux arrangements nécessaire sur la structure du volume.

Pour qu'un volume soit récupérable, il est impératif que chaque enregistrement du fichier journal documentant une modification du volume soit écrit sur disque avant que la modification en elle-même n'aie lieu. Compte tenu que les écritures disque sont aussi sujettes à la mise en cache, le gestionnaire de cache et le système de fichiers doivent coopérer que les phénomènes suivants se déroulent dans l'ordre indiqué :

1. Le système de fichiers écrit un enregistrement de journal qui documente la modification qu'il s'apprête à effectuer.

2. Le système de fichiers sollicite le gestionnaire de cache pour vidanger sur disque l'enregistrement du fichier journal.

3. Le système de fichiers écrit dans le cache la modification de volume.

4. Le gestionnaire de cache vidange sur le disque les méta données modifiées.

## Écritures immédiates du cache

Certaines applications ne souffrant aucun délai entre la modification d'un fichier et son enregistrement sur le disque, le gestionnaire de cache permet de faire de la vidange immédiate du cache pour un fichier individuel. Les opérations de niveau fichier prises en compte dans ce schéma de cache se déroulent alors de façon continue, les modifications étant enregistrées sur disque au fur et à mesure qu'elles apparaissent. Pour activer la vidange immédiate du cache, spécifiez le flag FILE_FLAG_WRITE_THROUGH lors de l'appel à CreateFile. Une autre alternative consiste à vidanger explicitement un fichier ouvert, ceci par l'intermédiaire de la fonction FlushFileBuffers. Contrairement à CreateFile, qui opère sur le plan global, FlushFileBuffers suppose de la solliciter chaque fois que la nécessité d'écrire les données sur le disque se fait sentir.

## Désactivation de la mise en cache

Pour diverses raisons qui leur appartiennent, les applications peuvent soustraire leurs fichiers de la mise en cache en employant lors de l'appel à CreateFile un flag spécifique à ces aspects : FILE_FLAG_NO_BUFFERING. En

pareil cas, le gestionnaire d'E/S génère un IRP représentant la requête et l'envoie au pilote de système de fichiers. Ce dernier envoie une requête directement vers le pilote du périphérique de stockage (ici, un pilote disque) en court-circuitant le cache du système de fichiers et effectue finalement l'opération d'E/S (lecture ou écriture) proprement dite.

| Fonctions | Services | Routines |
|---|---|---|
| CreateFile (dwFlagsAndAttributes= FILE_-FLAG_NO_BUFFERING) | NtCreateFile (CreateOptions = FILE_NO_-INTERMEDIATE_BUFFERING) | IoCreateFile(Ex) (CreateOptions = FILE_-NO_INTERMEDIATE_BUFFERING) |
| | NtOpenFile (OpenOptions = FILE_NO_-INTERMEDIATE_BUFFERING) | IoCreateFile(Ex) (CreateOptions = FILE_-NO_INTERMEDIATE_BUFFERING) |

## Structures de données du cache

Le gestionnaire de cache emploie un certain nombre de structures de données pour la gestion des fichiers mis en cache.

- Chaque fenêtre de 256 Ko du cache système est décrite par un VACB (Virtual Address Control Block).
- Chaque instance ouverte d'un fichier mis en cache ouvert séparément a un plan de cache privé (*private cache map*), qui contient les données servant à contrôler les lectures anticipées. Voir structure de données FILE_OBJECT.
- Chaque fichier mis en cache a un plan de cache partagé, qui décrit l'état du fichier mis en cache. Voir structure de données SHARED_CACHE_MAP.

Les sections qui suivent vont décrire ces structures et les relations que chacune tisse avec les autres.

### Structures de données globales du cache

Le gestionnaire de cache piste l'état des vues du cache système à l'aide d'un tableau de structures de données appelées VACB (Virtual Address Control Block). Lors de l'initialisation du système, le gestionnaire de cache alloue depuis le pool non paginé une quantité de mémoire suffisante pour contenir tous les VACB exigés par la description du cache système. Il enregistre ensuite l'adresse du tableau ainsi formé dans la variable CcVacbs. Chaque VACB représente une vue de 256 Ko du cache système.

**Structure VACB.**

```
lkd> dt nt!_VACB
   +0x000 BaseAddress      : Ptr64 Void
   +0x008 SharedCacheMap   : Ptr64 _SHARED_CACHE_MAP
   +0x010 Overlay          : <anonymous-tag>
   +0x020 ArrayHead        : Ptr64 _VACB_ARRAY_HEADER
```

- *Compteur d'actifs (ActiveCount)* Nombre de références à la vue, c'est-à-dire le nombre de lectures ou écritures la concernant. Pendant une opération d'E/S sur un fichier, le compteur de références du VACB se rapportant au fichier est incrémenté ; il est ensuite décrémenté à l'issue de l'opération.
- *Adresse de base (BaseAddress)* Adresse virtuelle des données du cache système. Voir routine MmMapViewInSystemCache.
- *Emplacement dans le fichier (FileOffset)* Emplacement dans le fichier à partir duquel commence la vue.
- *Plan de cache partagé (SharedCacheMap)* Pointeur vers la structure plan de cache partagé, laquelle identifie le fichier mis en cache.

***Variables système se rapportant aux VACB***

- CcNumberOfFreeVacbs Nombre de VACB disponibles pour l'allocation. Voir routine CcInsertVacbArray.

- CcNumberOfMappedVacbs
- CcVacbArraysHighestUsedIndex
- CcVacbFreeList Voir type de données LIST_ENTRY.
- CcVacbSpinLock Voir type de données PKSPIN_LOCK.

## Structures de données de niveau fichier du cache

Chaque handle ouvert pour un fichier a un objet fichier correspondant. Lors de la mise en cache du fichier, l'objet fichier est mis à jour de façon à pointer vers une structure plan de cache privé qui contient l'emplacement des deux dernières lectures concernant l'instance du fichier en cours d'accès, cela afin que le gestionnaire de cache puisse prédire quelles sont les données qu'un processus est susceptible de lire ultérieurement. Pour plus d'informations, consultez la section \<<lecture-anticipee>\>.

Chaque fichier mis en cache (par opposition à un objet fichier) a une structure plan de cache partagé (shared cache map) qui décrit l'état du fichier mis en cache.

Pour être constamment au fait des vues qui sont mappées dans le cache système pour un fichier donné, le gestionnaire de cache gère un tableau de pointeurs de VACB, appelé tableau des index de VACB.

**SHARED_CACHE_MAP.**

```
lkd> dt nt!_SHARED_CACHE_MAP
   +0x000 NodeTypeCode     : Int2B
   +0x002 NodeByteSize     : Int2B
   +0x004 OpenCount        : Uint4B
   +0x008 FileSize         : _LARGE_INTEGER
   +0x010 BcbList          : _LIST_ENTRY
   +0x020 SectionSize      : _LARGE_INTEGER
   +0x028 ValidDataLength  : _LARGE_INTEGER
   +0x030 ValidDataGoal    : _LARGE_INTEGER
   +0x038 InitialVacbs     : [4] Ptr64 _VACB
   +0x058 Vacbs            : Ptr64 Ptr64 _VACB
   +0x060 FileObjectFastRef : _EX_FAST_REF
   +0x068 VacbLock         : _EX_PUSH_LOCK
   +0x070 DirtyPages       : Uint4B
   +0x078 LoggedStreamLinks : _LIST_ENTRY
   +0x088 SharedCacheMapLinks : _LIST_ENTRY
   +0x098 Flags            : Uint4B
   +0x09c Status           : Int4B
   +0x0a0 Mbcb             : Ptr64 _MBCB
   +0x0a8 Section          : Ptr64 Void
   +0x0b0 CreateEvent      : Ptr64 _KEVENT
   +0x0b8 WaitOnActiveCount : Ptr64 _KEVENT
   +0x0c0 PagesToWrite     : Uint4B
   +0x0c8 BeyondLastFlush  : Int8B
   +0x0d0 Callbacks        : Ptr64 _CACHE_MANAGER_CALLBACKS
   +0x0d8 LazyWriteContext : Ptr64 Void
   +0x0e0 PrivateList      : _LIST_ENTRY
   +0x0f0 V1               : _LOGGED_STREAM_CALLBACK_V1
   +0x0f0 V2               : _LOGGED_STREAM_CALLBACK_V2
   +0x100 LargestLSN       : _LARGE_INTEGER
   +0x108 DirtyPageThreshold : Uint4B
   +0x10c LazyWritePassCount : Uint4B
   +0x110 UninitializeEvent : Ptr64 _CACHE_UNINITIALIZE_EVENT
   +0x118 BcbLock          : _FAST_MUTEX
   +0x150 LastUnmapBehindOffset : _LARGE_INTEGER
   +0x158 Event            : _KEVENT
   +0x170 HighWaterMappingOffset : _LARGE_INTEGER
   +0x178 PrivateCacheMap  : _PRIVATE_CACHE_MAP
```

```
+0x1f0 WriteBehindWorkQueueEntry : Ptr64 Void
+0x1f8 VolumeCacheMap   : Ptr64 _VOLUME_CACHE_MAP
+0x200 ProcImagePathHash : Uint4B
+0x204 WritesInProgress : Uint4B
+0x208 AsyncReadRequestCount : Uint4B
+0x210 Partition        : Ptr64 _CC_PARTITION
+0x218 InternalRefCount : Uint4B
+0x21c NumMappedVacb    : Uint4B
+0x220 NumActiveVacb    : Uint4B
```

- *DirtyPageThreshold* Voir routine CcSetDirtyPageThreshold.
- *NodeTypeCode* Voir routine CcInitializeCacheMap.

## Taille du cache

Comme la plupart des calculs concernant la gestion de la mémoire, la taille virtuelle du cache système dépend d'un certain nombre de facteurs :

- Architecture supportée (32 ou 64 bits)
- Quantité de mémoire physique installée
- Géographie de l'espace d'adressage (comprendre l'emploi d'options spéciales qui modifient de façon substantielle celle par défaut)
- Version (client ou serveur) de Windows exécutée

### Taille virtuelle du cache

Sur les versions 32 bits de Windows, la taille virtuelle du cache système est essentiellement fonction de la dimension de l'espace d'adressage système (2 Go). Cette zone de mémoire étant partagée avec d'autres ressources, y compris les entrées les tables de page et entrées de table de page, le pool paginé et non paginé, la quantité de mémoire dévolue seulement à la gestion de cache est en réalité beaucoup plus petite. Sur les versions de Windows orientées 64 bits, la taille virtuelle maximale du cache système est de 1024 Go (1 To).

### Taille du working set du cache

Le cache système n'a pas de working set qui lui est dévolu en propre, mais partage une unique plage de travail système qui renferme les données issues de plusieurs composants, y compris cache, pool paginé, code paginable et vues mappées du système (noyau et pilotes), etc.

## Lecture anticipée

Le gestionnaire de cache de Windows utilise le principe de localité spatiale pour faire de la lecture anticipée (*read ahead*), et de la sorte prédire quelles sont les données qu'un processus est susceptible de lire à l'occasion de ses prochaines requêtes d'E/S.

Une première forme de lecture anticipée, appelée préchargement virtuel, a lieu quand le gestionnaire de mémoire doit résoudre un défaut de page, et effectue à cette occasion une lecture de plusieurs pages proches de celles accédées explicitement. Pour les applications qui font de l'accès séquentiel, cette méthode a pour avantage majeur une réduction des accès disque, d'où une amélioration évidente des performances. L'API Windows CreateFile, en vue d'améliorer d'autant plus la prise en compte de ce modèle, rend visible un flag qui permet d'indiquer un accès séquentiel vers l'avant sur le fichier : FILE_FLAG_SEQUENTIAL_SCAN. Le fait de lire le bloc

suivant d'un fichier n'offre de bénéfices que dans une perspective séquentielle. Aussi, pour étendre l'utilité de la lecture anticipée aux cas des accès éparpillés, le gestionnaire de cache gère un historique des deux dernières requêtes de lecture dans le plan de cache privé du fichier en cours d'accès. Cette façon de procéder porte le nom de lecture anticipée asynchrone avec historique. Dans cette configuration, le gestionnaire de cache extrapole à partir des lectures faites par l'appelant un scénario, et agit conséquemment à ces critères. Par exemple, si l'appelant lit la page 1000 puis la page 2000, le gestionnaire de cache part du principe que la page suivante demandée sera la page 3000, qu'il charge donc à l'avance. Les applications qui accèdent aléatoirement à un fichier, et n'ont en l'occurence pas de modèle de lecture prévisible, peuvent souligner cette spécificité auprès du gestionnaire de cache via l'emploi du flag FILE_FLAG_RANDOM_ACCESS lors de l'appel à CreateFile. En la circonstance, le gestionnaire de cache désactive la lecture anticipée, cessant alors de prédire l'emplacement de la prochaine prochaine lecture, et suspend l'ensemble de ses activités en ce qui concerne les vues inactives du fichier. (Dans la configuration par défaut, les vues inactives d'un fichier sont démappées au fur et à mesure que d'autres vues deviennent actives.)

La lecture anticipée faite par le gestionnaire de cache est asynchrone, car effectuée dans un thread distinct de celui de l'appelant et se déroulant en parallèle avec l'exécution de ce dernier.

## Fast I/O

Chaque fois que possible, les séquences d'opérations (lectures et écritures) mobilisées en ce qui concerne des fichiers mis en cache sont pris en charge par le biais d'un mécanisme haut débit appelé Fast I/O, lequel offre la possibilité d'interagir avec un fichier sans obligation de générer un IRP pour représenter chaque requête. Lorsque les activités d'E/S pour un fichier sont portés par Fast I/O, les pilotes de système de fichiers utilisent le gestionnaire de cache de sorte à accéder aux données fichier en copiant tout simplement vers ou depuis les pages mappées vers le fichier réel concerné.

Compte tenu de la dynamique entourant la mise en cache des fichiers, Fast I/O n'a pas lieu de façon systématique. Par exemple, le premier accès (lecture ou écriture) dans un fichier exige du système la configuration du fichier pour la mise en cache, ce qui exclut de fait le recours à une telle méthode d'optimisation. En plus de cela, si l'appelant s'appuie sur des E/S asynchrones, Fast I/O n'entre pas en scène, car dans l'éventualité contraire, l'appelant risquerait d'être bloqué pendant des opérations d'E/S de pagination requises par la copie du tampon vers et depuis le cache système, ce qui irait à l'encontre du caractère asynchrone de l'E/S. Outre ces différents aspects, même pour E/S synchrone, la probabilité du traitement Fast I/O n'est pas encore tout à fait établie, puisque le pilote de système de fichiers peut encore de son propre chef décider de ne pas employer ledit mécanisme, par exemple si le fichier a une plage d'octets verrouillée (suite à des appels aux fonctions LockFile et UnlockFile). Bien que très étroitement lié au sous-système de mémoire virtuelle, le gestionnaire de cache n'ayant pas connaissance de quelles portions de quels fichiers sont verrouillées, le pilote de système de fichiers a pour obligation de contrôler la validité du tampon impliqué, ce qui impose un IRP.

Voici dans les grandes lignes ce qui se revient lors du traitement d'une E/S avec Fast I/O.

1. Un thread effectue une opération de lecture ou d'écriture.

2. Si le fichier est présent dans le cache et que l'E/S est de nature synchrone, la gestionnaire d'E/S appelle en lui passant la requête la routine Fast I/O du pilote de système de fichiers. Si le fichier n'est pas dans le cache, le pilote du système de fichiers configure le fichier pour la mise en cache, de sorte que les accès ultérieurs puissent être pris en charge via Fast I/O.

3. Si la routine Fast I/O du pilote de système de fichiers détermine que Fast I/O est possible, elle appelle la routine de lecture ou d'écriture du gestionnaire de cache pour accéder aux données dans le cache. Dans l'éventualité où Fast I/O n'est pas possible, le pilote de système de fichiers rend la main au système d'E/S, qui crée alors un IRP et appelle en définitive la routine de lecture ou d'écriture classique du système de fichiers.

4. Le gestionnaire de cache traduit l'emplacement demandé dans le fichier en une adresse virtuelle dans le cache.

5. Pour les lectures, le gestionnaire de cache copie les données du cache dans le tampon du processus dont émane la requête d'E/S ; pour les écritures, il copie les données du tampon dans le cache. Les options suivantes sont à considérer.

   - Pour les lectures, on actualise les informations de lecture anticipée dans le plan de cache privé de l'instance du fichier ouvert - sauf si le mode aléatoire (FILE_FLAG_RANDOM_ACCESS) a été spécifié lors de l'ouverture du fichier, lequel mode empêche toute prévisibilité des emplacements susceptibles d'être demandés lors de prochaines opérations d'E/S.

   - Pour les écritures, on arme le bit de modification de chaque page modifiée du cache afin que l'écrivain paresseux (lazy writer) soit amené dans un futur plus ou moins proche à les vidanger sur le disque. Pour les fichiers ouverts en demandant des écritures immédiates du cache (flag FILE_FLAG_WRITE_THROUGH), on vidange toutes les modifications.

Dans la perspective Fast I/O, aucun IRP n'étant impliqué, l'opération ne peut pas bloquer, et ne peut non plus être placée dans la file d'attente d'un thread. De ce fait, lorsque l'opération atteint le système de fichiers, elle échoue si les informations ne sont pas présentes dans le cache. Le gestionnaire d'E/S tente alors l'opération en utilisant le chemin normal des IRP.

Les compteurs de performances et variables système du tableau suivant permettent d'examiner les activités Fast I/O du système.

***Tableau 19.1. Champs KPRCB sous-jacents aux activités de lecture rapide***

| Champ | Compteur | Description |
|---|---|---|
| CcFastReadWait | Cache:Lectures rapides sync/s | Lectures synchrones ayant été traitées en tant que requêtes Fast I/O |
| CcFastReadNoWait | Cache:Lectures rapides async/s | Lectures asynchrones ayant été traitées en tant que requêtes Fast I/O.[a] |
| CcFastReadNotPossible | Cache:Impossibilités de lectures rapides/s | Requêtes Fast I/O n'ayant pu être satisfaites [b] |
| CcFastReadResourceMiss | Cache:Insuccès de lectures rapides/s | Requêtes Fast I/O n'ayant pu être satisfaites pour cause de conflits de ressources [c] |

[a]Étant donné qu'il n'y a pas de lectures rapides asynchrones dans Windows, la valeur de ce compteur est toujours nulle.

[b]La question du traitement d'une E/S avec ou sans Fast I/O est du ressort du pilote de système de fichiers. Un fichier contenant une plage d'octets verrouillée ne peut par exemple pas bénéficier des E/S rapides.

[c]Cette situation peut se produire avec FAT, pas avec NTFS.

## Interfaces avec le système de fichiers

Lors du premier accès (lecture ou écriture) aux données d'un pilote, le pilote du système de fichiers est chargé de déterminer s'il existe une partie du fichier qui réside dans le cache. Dans l'éventualité du contraire, le pilote du système de fichiers doit solliciter le gestionnaire de cache dans l'optique qu'il crée les structures de données nécessaires pour la mise en cache dudit fichier (CcInitializeCacheMap). Dans la plupart des systèmes de fichiers, la mise en cache des fichiers est active par défaut, mais peut être désactivée en spécifiant le flag FILE_NO_INTERMEDIATE_BUFFERING parmi les options de création du fichier.

Une fois le fichier configuré pour le cache, le pilote du système de fichiers a le choix pour accéder aux données du fichier entre plusieurs sous-ensembles des fonctionnalités du gestionnaire de cache, à savoir :

- *Copie vers et depuis le cache* La méthode copie, peut-être la naive parmi celles proposées, a pour effet de copier les données utilisateur entre les tampons du cache, qui siègent dans l'espace espace système, et un tampon de processus, situé dans l'espace utilisateur.

- *Mappage et épinglage* La méthode de mappage et épinglage, considérée comme la plus optimale, utilise des adresses virtuelles pour lire et écrire les données directement dans les tampons du cache.

- *Accès direct à la mémoire* La méthode d'accès à la mémoire physique véhicule des adresses physiques.

## Copie vers et depuis le cache

Le cache système résidant dans l'espace système, il est de ce fait mappé dans l'espace d'adressage de chaque processus. Cependant, comme c'est le cas de toutes les pages de l'espace système, les pages du cache ne sont pas accessibles depuis le mode utilisateur. Par conséquent, les E/S fichier qui émane d'une application utilisateur concernant des fichiers mis en cache doivent être prises en charge par des routines mode noyau qui copient les données entre les tampons du cache (situés comme nous l'avons dit dans l'espace système) et les tampons de l'application (situées dans l'espace d'adressage du processus). Le tableau qui suit énumère les routines que les pilotes de système de fichiers peuvent employer à cette fin. .Routines de de copie vers et depuis le cache

| Routine | Description |
|---|---|
| CcCopyRead | Copie une plage d'octets du cache système vers un tampon utilisateur |
| CcFastCopyRead | Variante optimisée de CcCopyRead, mais limitée à des emplacements fichier de 32 bits et à des lectures synchrones |
| CcCopyWrite | Copie une plage d'octets d'un tampon utilisateur vers le cache système |
| CcFastCopyWrite | Variante optimisée de CcCopyWrite, mais limitée à des emplacements fichier de 32 bits et à des lectures synchrones |

*Tableau 19.2. Eléments auxiliaires sous-jacents aux lectures dans le cache*

| Élément | Lieu | Autres références |
|---|---|---|
| CcCopyReadNoWait | SYSTEM_PERFORMANCE_INFORMATION | NtQuerySystemInformation (SystemInformationClass = SystemPerformanceInformation) |
| CcCopyReadNoWaitMiss | SYSTEM_PERFORMANCE_INFORMATION | NtQuerySystemInformation (SystemInformationClass = SystemPerformanceInformation) |
| CcCopyReadWaitMiss | SYSTEM_PERFORMANCE_INFORMATION | NtQuerySystemInformation (SystemInformationClass = SystemPerformanceInformation) |
| CcCopyReadWait | KPRCB | CcCopyRead (Wait = TRUE) |
| CcCopyReadNoWait | KPRCB | CcCopyRead (Wait = FALSE) |
| CcCopyReadNoWaitMiss | KPRCB | CcCopyRead |
| CcCopyReadWaitMiss | KPRCB | |

## Mappage et épinglage

Les interfaces de mappage/épinglage forment un procédé d'optimisation par lequel le gestionnaire de cache offre la possibilité aux pilotes de système de fichiers de modifier des données directement dans le cache système, sans passer par des tampons intermédiaires.

*Tableau 19.3. Interfaces de mappage/épinglage*

| Routine | Description |
|---|---|
| CcMapData | Mappe la plage d'octets pour des accès en lecture |
| CcPinRead | Mappe la plage d'octets pour des accès en lecture/écriture et l'épingle |
| CcPinMappedData | Epingle un tampon précédemment mappé |
| CcSetDirtyPinnedData | Informe le gestionnaire de cache que les données ont été modifiées |
| CcPreparePinWrite | Mappe et épingle la plage d'octets pour des accès en écriture (les lectures ne sont pas valides) |
| CcUnpinData | Libère les pages de sorte qu'elles puissent être supprimées de la mémoire |

## Cache et interfaces DMA

Le gestionnaire de cache procure une troisième interface vers les données du cache, appelée DMA (Direct Memory Access), dont les fonctions servent à lire ou à écrire les pages du cache sans passer par des tampons intermédiaires.

L'interface DMA retourne au système de fichiers les adresses physiques des données utilisateur mises en cache (au lieu des adresses virtuelles que retournent les interfaces de mappage et d'épinglage).

Les quelques routines présentées au tableau suivant implémentent l'interface DMA du gestionnaire de cache.

***Tableau 19.4. Fonctions de l'interface DMA***

| Fonction | Description |
|---|---|
| CcMdlRead | Retourne une MDL décrivant une plage d'octets spécifiée |
| CcMdlReadComplete | Libère la MDL |
| CcMdlWrite | Retourne une MDL décrivant une plage d'octets spécifiée |
| CcMdlWriteComplete | Libère la MDL et marque la plage comme étant modifiée |
| CcMdlWriteAbort | |
| CcPrepareMdlWrite | |

## Cache avec écritures en différé

Le gestionnaire de cache implémente un cache avec écritures en différé (*write-back*). Cela signifie que les données écrites dans des fichiers sont d'abord stockées en mémoire dans des pages du cache, où elles s'accumulent le temps nécessaire (comprendre jusqu'à être assez nombreuses) pour motiver une vidange du contenu du cache sur le disque. Procéder ainsi présente l'avantage, d'une part de diminuer le nombre global des E/S disque, d'autre part de retarder les IRP émis à destination des pilotes.

Le choix de la fréquence de vidange du cache est une question délicate. Si le cache est vidangé trop souvent, les performances du système seront diminuées par des E/S superflues. S'il l'est trop rarement, le système risque de se trouver à court de mémoire physique (parce qu'il y a trop de pages modifiées en mémoire), voire en cas d'incident de perdre des données fichier modifiées.

### Lazy Writer

Pour minimiser les risques inhérents à une stratégie de cache à écriture différée, et de la sorte réduire la fenêtre de vulnérabilité entre le moment où des modifications de fichier sont apportés dans le cache puis répercutées sur le disque, Windows sollicite à intervalles réguliers une fonction spéciale du gestionnaire de cache, dite écrivain paresseux (*lazy writer*), exécutée dans un thread auxiliaire système.

### Désactivation des écritures en différé pour un fichier

L'écrivain paresseux considère les fichiers temporaires (FILE_ATTRIBUTE_TEMPORARY dans l'API Windows) avec une attention spéciale, et n'écrit sur disque les pages modifiées apparentées à de tels fichiers que si les circonstances l'exigent (manque de mémoire physique) ou dans le cas d'une vidange explicite du fichier (FlushFileBuffers).

## Cache et mémoire virtuelle

Comme il opère sur une base virtuelle, le gestionnaire de cache se voit confier un sous-ensemble des espaces d'adressage virtuels du système, région qu'il divise en portions de 256 Ko, nommées *vues* (voir figure ci-dessous), consacrées au stockage des données mises en cache.

Lors de la première E/S (lecture ou écriture) sur un fichier, le gestionnaire de cache mappe dans une fenêtre libre de l'espace d'adressage du cache système une vue de 256 Ko de la région du fichier qui contient les données demandées. Par exemple, dans l'éventualité d'une opération s'effectuant à partir de l'emplacement 300 000 octets d'un fichier, la vue qui sera alors mappée commencera à la position 262 144 (seconde région alignée sur 256 Ko) dudit fichier, et s'étendra sur 256 Ko.

Le gestionnaire de cache mappe les vues des fichiers dans des fenêtres de l'espace d'adressage du cache selon le principe du tourniquet (*round-robin*) : la première vue demandée est mappée dans la première fenêtre, la deuxième vue dans la seconde fenêtre, et ainsi de suite.

Les vues des fichiers mappées dans des fenêtres de l'espace d'adressage du cache peuvent être à l'état actif ou inactif, selon qu'une E/S en cours s'adresse à la vue concernée. Le gestionnaire de cache garantit qu'une vue est mappée tant qu'elle est active (encore qu'une vue puisse rester mappée après devenue inactive).

Les pages qui correspondent à des vues de fichiers manipulés de façon séquentielle (FILE_FLAG_SEQUENTIAL_SCAN spécifié lors de l'appel à CreateFile ou si le système estime que la stratégie s'y prête) sont démappées automatiquement à mesure que le gestionnaire de cache mappe de nouvelles vues pour le fichier. Une telle façon de procéder encourage la réutilisation desdites pages, et empêche tout spécialement une copie de fichier de monopoliser plus d'une petite partie de la mémoire physique. Les pages qui correspondent à des fichiers qui n'ont pas de modèle de lecture prévisible (FILE_FLAG_RANDOM_ACCESS) empêchent le gestionnaire de cache de démapper automatiquement les vues correspondantes.

Dans l'éventualité où le gestionnaire de cache a besoin de mapper une vue d'un fichier et qu'il n'existe plus pour ce faire de fenêtres libres, il démappe la vue inactive qui est mappée depuis le plus longtemps et emploie cette fenêtre.

# Chapitre 20. Interceptions

Ce chapitre traite de la façon dont Windows met en oeuvre le traitement des interceptions, à savoir interruptions, exceptions et services système.

## Gestion des interruptions

```
lkd> !idt

Dumping IDT: fffff80245bdd000

00: fffff802486b2100 nt!KiDivideErrorFaultShadow
01: fffff802486b2180 nt!KiDebugTrapOrFaultShadow Stack = 0xFFFFF80245BE19D0
02: fffff802486b2240 nt!KiNmiInterruptShadow Stack = 0xFFFFF80245BE17D0
03: fffff802486b22c0 nt!KiBreakpointTrapShadow
04: fffff802486b2340 nt!KiOverflowTrapShadow
05: fffff802486b23c0 nt!KiBoundFaultShadow
06: fffff802486b2440 nt!KiInvalidOpcodeFaultShadow
07: fffff802486b24c0 nt!KiNpxNotAvailableFaultShadow
08: fffff802486b2540 nt!KiDoubleFaultAbortShadow Stack = 0xFFFFF80245BE13D0
09: fffff802486b25c0 nt!KiNpxSegmentOverrunAbortShadow
0a: fffff802486b2640 nt!KiInvalidTssFaultShadow
0b: fffff802486b26c0 nt!KiSegmentNotPresentFaultShadow
0c: fffff802486b2740 nt!KiStackFaultShadow
0d: fffff802486b27c0 nt!KiGeneralProtectionFaultShadow
0e: fffff802486b2840 nt!KiPageFaultShadow
10: fffff802486b28c0 nt!KiFloatingErrorFaultShadow
11: fffff802486b2940 nt!KiAlignmentFaultShadow
12: fffff802486b29c0 nt!KiMcheckAbortShadow Stack = 0xFFFFF80245BE15D0
13: fffff802486b2ac0 nt!KiXmmExceptionShadow
14: fffff802486b2b40 nt!KiVirtualizationExceptionShadow
15: fffff802486b2bc0 nt!KiControlProtectionFaultShadow
1f: fffff802486b2c40 nt!KiApcInterruptShadow
20: fffff802486b2cc0 nt!KiSwInterruptShadow
29: fffff802486b2d40 nt!KiRaiseSecurityCheckFailureShadow
2c: fffff802486b2dc0 nt!KiRaiseAssertionShadow
2d: fffff802486b2e40 nt!KiDebugServiceTrapShadow
2f: fffff802486b2f40 nt!KiDpcInterruptShadow
30: fffff802486b2fc0 nt!KiHvInterruptShadow
31: fffff802486b3040 nt!KiVmbusInterrupt0Shadow
...

lkd> dt nt!_KINTERRUPT ffff8181dcd39b40
   +0x000 Type             : 0n22
   +0x002 Size             : 0n256
   +0x008 InterruptListEntry : _LIST_ENTRY [ 0x00000000`00000000 - 0x00000000`00000000 ]
   +0x018 ServiceRoutine   : 0xfffff80b`a34c5e70     unsigned char  i8042prt!I8042KeyboardInterruptService+0
   +0x020 MessageServiceRoutine : (null)
   +0x028 MessageIndex     : 0
   +0x030 ServiceContext   : 0xffff930b`8cd2a9f0 Void
   +0x038 SpinLock         : 0
   +0x040 TickCount        : 0
   +0x048 ActualLock       : 0xffff930b`8cd2ab50  -> 0
   +0x050 DispatchAddress  : 0xfffff801`9bfe9810     void  nt!KiInterruptDispatch+0
   +0x058 Vector           : 0x80
   +0x05c Irql             : 0x8 ''
   +0x05d SynchronizeIrql  : 0x8 ''
   +0x05e FloatingSave     : 0 ''
   +0x05f Connected        : 0x1 ''
   +0x060 Number           : 0
   +0x064 ShareVector      : 0 ''
   +0x065 EmulateActiveBoth : 0 ''
   +0x066 ActiveCount      : 0
   +0x068 InternalState    : 0n0
```

```
+0x06c Mode             : 1 ( Latched )
+0x070 Polarity         : 0 ( InterruptPolarityUnknown )
+0x074 ServiceCount     : 0
+0x078 DispatchCount    : 0
+0x080 PassiveEvent     : (null)
+0x088 TrapFrame        : 0xfffffce04`7e217a80 _KTRAP_FRAME
+0x090 DisconnectData   : (null)
+0x098 ServiceThread    : (null)
+0x0a0 ConnectionData   : 0xffff930b`8cdefde0 _INTERRUPT_CONNECTION_DATA
+0x0a8 IntTrackEntry    : 0xffff930b`8cdbfdc0 Void
+0x0b0 IsrDpcStats      : _ISRDPCSTATS
+0x0f0 RedirectObject   : (null)
+0x0f8 Padding          : [8]  ""
```

- *Mode* Voir structure KINTERRUPTMODE.
- *Polarity* Voir structure KINTERRUPTPOLARITY.
- *ISR (ServiceRoutine)* Adresse de la routine de service en charge de l'interruption. Voir routine IoConnectInterrupt et IoConnectInterruptEx.
- *Verrou tournant (SpinLock)* Voir routine KeAcquireInterruptSpinLock, KeReleaseInterruptSpinLock.
- *Type de l'objet (Type)* Valeur qui correspond au type de l'objet tel qu'il est définit dans la perspective du noyau. Voir champ InterruptObject parmi l'énumération KOBJECTS.
- *Vecteur d'interruption (Vector)*
- *IsrDpcStats* Enregistre différentes métriques relatives au traitement de l'interruption, par exemple l'heure de début et la durée d'exécution de la routine de service associée. Voir structure ISRDPCSTATS.

## Contrôleurs d'interruption x86

La plupart des systèmes x86 s'appuient soit sur le contrôleur PIC (Programmable Interrupt Controller) i8259A, soit sur une variante du contrôleur APIC (Advanced Programmable Interrupt Controller) i82489. La majorité des ordinateurs modernes incluent un APIC, donnant de la sorte aux systèmes d'exploitation la possibilité de bénéficier d'un tel dispositif pour le contrôle des interruptions. La norme PIC tire ses origines du PC IBM originel. Les PIC ne fonctionnent qu'avec les systèmes mono processeur et disposent de 15 lignes d'interruption - un chiffre qui provient en réalité de deux contrôleurs mis en cascade selon le modèle maître-esclave. Les APIC fonctionnent avec les systèmes multi processeur et ont 256 lignes d'interruption.

En vue de conserver la compatibilité avec les systèmes d'exploitation mono processeur, les APIC offrent du code d'amorçage qui démarre une machine multi processeur en mode mono processeur, et intègrent un mode de compatibilité PIC à 15 interruptions (lesquelles interruptions sont dans cette configuration livrées uniquement au processeur principal, seul actif parmi tous ceux disponibles).

La technologie APIC s'articule autour de plusieurs composants matériels majeurs, chacun responsable de fournir une fonction précise : un APIC d'E/S (IO-APIC) qui reçoit les interruptions émanant des périphériques, et des APIC locaux qui reçoivent les interruptions de l'APIC d'E/S sur un bus privé (ce qui tend à favoriser un échange rapide des informations) et qui interrompent le processeur auquel ils sont associés. L'objectif premier de l'APIC d'E/S est d'implémenter des algorithmes efficaces et évolutifs en ce qui concerne le routage des interruptions. Il collecte à ce titre les interruptions en provenance de multiples sources, et assure l'acheminement de ces dernières auprès d'un ou plusieurs processeurs.

Les APIC locaux permettent la gestion des interruptions au niveau du processeur local. Dans la perspective d'un système multi processeur, chacun des APIC offre aussi la possibilité de générer des interruptions destinées à des processeurs voisins ; ces messages sont alors appelés interruptions inter-processeur (IPI, Inter-processor Interrupt). Un autre avantage procuré par les APIC locaux est l'intégration de timers haute précision.

## Contrôleurs d'interruption x64

L'architecture x64 assurant un maximum de compatibilité avec les systèmes d'exploitation x86, les systèmes x86 fournissent sensiblement les mêmes contrôleurs que le x86. Une différence significative, toutefois, réside dans le fait que les versions de Windows calibrées pour les plateformes x64 ne fonctionnent tout bonnement pas sur des machines dépourvues d'APIC.

## Interruptions matérielles

Sur les plateformes matérielles animées par Windows (comme du reste par tout autre système d'exploitation), les interruptions externes d'E/S arrivent sur l'une des lignes d'un contrôleur d'interruption. Le contrôleur interrompt ensuite le processeur, qui demande alors au contrôleur quelle est la requête d'interruption (IRQ). Le contrôleur traduit l'IRQ en un numéro d'interruption, lequel se rapporte à une structure appelée table de ventilation d'interruption (IDT), puis transfère le contrôle à la routine idoine de ventilation d'interruption.

Chaque processeur dispose de sa propre table de ventilation d'interruption, de sorte que les différents processeurs peuvent exécuter des ISR différentes si besoin est. Ainsi, dans un système multi processeur, chaque processeur reçoit l'interruption d'horloge, mais un seul processeur actualise l'horloge en réponse à cette interruption. En outre, certaines configurations système peuvent exiger que ce soit tel ou tel processeur particulier qui traite telle ou telle interruption matérielle.

# Gestion des services système

Sur les processeurs x86 antérieurs au Pentium II, Windows emploie de sorte à jeter un pont entre l'espace d'adressage utilisateur et l'espace d'adressage système l'instruction générique 2e, laquelle entraine une interception qui force le thread en cours d'exécution à basculer en mode noyau et à entrer dans le dispatcher de service système. Un argument numérique passé au registre EAX du processeur indique un numéro correspondant en interne au service sollicité. Le registre EBX pointe quant à lui vers la liste des paramètres que l'appelant souhaite transmettre audit service. Compte tenu de sa nature, à savoir une interruption logicielle parmi d'autres, l'instruction 2e peut avoir un impact plus ou moins sévère sur les performances du système. D'où le besoin d'un chemin optimisé entre le mode utilisateur et le mode noyau (et vice-versa), auquel vient en l'occurence répondre deux déclinaisons d'une même figure : sysenter sur les plateformes Intel, et syscall chez AMD.

Sur les processeurs Pentium II et supérieurs, Windows emploie l'instruction spéciale *sysenter*, spécifiquement définie pour la prise en charge rapide des appels système. À la différence des interruptions logicielles, dont les modalités de traitement impliquent des coûts additionnels élevés (essentiellement du fait des lectures des différentes structures de données impliquées), l'instruction sysenter offre un maximum de performances lors des différentes transitions entre anneaux de protection, cela notamment grâce à l'emploi de registres spécifiques (MSR, Model-specific register) qui renferment de base toutes les informations nécessaires à identifier promptement la routine dispatcher de service système. L'exécution de l'instruction force le passage en mode noyau et l'exécution du dispatcher. Le numéro de service système est passé dans le registre EAX du processeur, le registre EBX pointant vers la liste des arguments de l'appelant. Pour revenir au mode utilisateur, le dispatcher exécute généralement l'instruction sysexit.

***Tableau 20.1. Services système***

| Service | Fonction |
|---|---|
| NtAcceptConnectPort | |
| NtAccessCheck | |
| NtAccessCheckAndAuditAlarm | |
| NtAccessCheckByType | |
| NtAccessCheckByTypeAndAuditAlarm | |
| NtAccessCheckByTypeResultList | |
| NtAccessCheckByTypeResultListAndAuditAlarm | |
| NtAccessCheckByTypeResultListAndAuditAlarmByHandle | |
| NtAcquireCMFViewOwnership | |

| | |
|---|---|
| NtAcquireProcessActivityReference | |
| NtAddAtom | GlobalAddAtom |
| NtAddAtomEx | |
| NtAddBootEntry | |
| NtAddDriverEntry | |
| NtAdjustGroupsToken | |
| NtAdjustPrivilegesToken | |
| NtAdjustTokenClaimsAndDeviceGroups | |
| NtAlertResumeThread | |
| NtAlertThread | |
| NtAlertThreadByThreadId | |
| NtAllocateLocallyUniqueId | |
| NtAllocateReserveObject | |
| NtAllocateUserPhysicalPages | AllocateUserPhysicalPages |
| NtAllocateUuids | |
| NtAllocateVirtualMemory | VirtualAlloc(Ex) |
| NtAlpcAcceptConnectPort | |
| NtAlpcCancelMessage | |
| NtAlpcConnectPort | |
| NtAlpcConnectPortEx | |
| NtAlpcCreatePort | |
| NtAlpcCreatePortSection | |
| NtAlpcCreateResourceReserve | |
| NtAlpcCreateSectionView | |
| NtAlpcCreateSecurityContext | |
| NtAlpcDeletePortSection | |
| NtAlpcDeleteResourceReserve | |
| NtAlpcDeleteSectionView | |
| NtAlpcDeleteSecurityContext | |
| NtAlpcDisconnectPort | |
| NtAlpcImpersonateClientContainerOfPort | |
| NtAlpcImpersonateClientOfPort | |
| NtAlpcOpenSenderProcess | |
| NtAlpcOpenSenderThread | |
| NtAlpcQueryInformation | |
| NtAlpcQueryInformationMessage | |
| NtAlpcRevokeSecurityContext | |
| NtAlpcSendWaitReceivePort | |
| NtAlpcSetInformation | |
| NtApphelpCacheControl | |
| NtAreMappedFilesTheSame | |
| NtAssignProcessToJobObject | |
| NtAssociateWaitCompletionPacket | |
| NtCallEnclave | |
| NtCallbackReturn | |
| NtCancelDeviceWakeupRequest | CancelDeviceWakeupRequest |
| NtCancelIoFile | CancelIo |
| NtCancelIoFileEx | CancelIoEx |

| | |
|---|---|
| NtCancelSynchronousIoFile | |
| NtCancelTimer | CancelWaitableTimer |
| NtCancelTimer2 | |
| NtCancelWaitCompletionPacket | |
| NtClearAllSavepointsTransaction | |
| NtClearEvent | |
| NtClearSavepointTransaction | |
| NtClose | CloseHandle |
| NtCloseObjectAuditAlarm | |
| NtCommitComplete | |
| NtCommitEnlistment | |
| NtCommitRegistryTransaction | |
| NtCommitTransaction | |
| NtCompactKeys | |
| NtCompareObjects | |
| NtCompareSigningLevels | |
| NtCompareTokens | |
| NtCompleteConnectPort | |
| NtCompressKey | |
| NtConnectPort | |
| NtContinue | |
| NtConvertBetweenAuxiliaryCounterAndPerformanceCounter | |
| NtCreateDebugObject | |
| NtCreateDirectoryObject | |
| NtCreateDirectoryObjectEx | |
| NtCreateEnclave | |
| NtCreateEnlistment | |
| NtCreateEvent | CreateEvent |
| NtCreateEventPair | |
| NtCreateFile | CreateFile |
| NtCreateIRTimer | |
| NtCreateIoCompletion | CreateIoCompletionPort |
| NtCreateJobObject | |
| NtCreateJobSet | |
| NtCreateKey | |
| NtCreateKeyTransacted | |
| NtCreateKeyedEvent | |
| NtCreateLowBoxToken | |
| NtCreateMailslotFile | |
| NtCreateMutant | |
| NtCreateNamedPipeFile | |
| NtCreatePagingFile | |
| NtCreatePartition | |
| NtCreatePort | |
| NtCreatePrivateNamespace | |
| NtCreateProcess | CreateProcess, CreateProcessAsUser |
| NtCreateProcessEx | |
| NtCreateProfile | |

| | |
|---|---|
| NtCreateProfileEx | |
| NtCreateRegistryTransaction | |
| NtCreateResourceManager | |
| NtCreateSection | |
| NtCreateSemaphore | CreateSemaphore |
| NtCreateSymbolicLinkObject | |
| NtCreateThread | CreateRemoteThread, CreateThread |
| NtCreateThreadEx | |
| NtCreateTimer | CreateWaitableTimer |
| NtCreateTimer2 | |
| NtCreateToken | CreateToken |
| NtCreateTokenEx | |
| NtCreateTransaction | |
| NtCreateTransactionManager | |
| NtCreateUserProcess | |
| NtCreateWaitCompletionPacket | |
| NtCreateWaitablePort | |
| NtCreateWnfStateName | |
| NtCreateWorkerFactory | |
| NtDebugActiveProcess | |
| NtDebugContinue | |
| NtDelayExecution | Sleep |
| NtDeleteAtom | GlobalDeleteAtom |
| NtDeleteBootEntry | |
| NtDeleteDriverEntry | |
| NtDeleteFile | |
| NtDeleteKey | RegDeleteKey |
| NtDeleteObjectAuditAlarm | ObjectDeleteAuditAlarm |
| NtDeletePrivateNamespace | |
| NtDeleteValueKey | RegDeleteValue |
| NtDeleteWnfStateData | |
| NtDeleteWnfStateName | |
| NtDeviceIoControlFile | DeviceIoControl |
| NtDisableLastKnownGood | |
| NtDisplayString | |
| NtDrawText | |
| NtDuplicateObject | DuplicateHandle |
| NtDuplicateToken | DuplicateToken(Ex) |
| NtEnableLastKnownGood | |
| NtEnumerateBootEntries | |
| NtEnumerateDriverEntries | |
| NtEnumerateKey | RegEnumKey, RegEnumKeyEx |
| NtEnumerateSystemEnvironmentValuesEx | |
| NtEnumerateTransactionObject | |
| NtEnumerateValueKey | |
| NtExtendSection | |
| NtFilterBootOption | |
| NtFilterToken | CreateRestrictedToken |

| | |
|---|---|
| NtFilterTokenEx | |
| NtFindAtom | |
| NtFlushBuffersFile | FlushFileBuffers |
| NtFlushBuffersFileEx | |
| NtFlushInstallUILanguage | |
| NtFlushInstructionCache | FlushInstructionCache |
| NtFlushKey | RegFlushKey |
| NtFlushProcessWriteBuffers | |
| NtFlushVirtualMemory | FlushViewOfFile |
| NtFlushWriteBuffer | |
| NtFreeUserPhysicalPages | FreeUserPhysicalPages |
| NtFreeVirtualMemory | VirtualFree(Ex), BaseFreeThreadStack |
| NtFreezeRegistry | |
| NtFreezeTransactions | |
| NtFsControlFile | DeviceIoControl |
| NtGetCachedSigningLevel | |
| NtGetCompleteWnfStateSubscription | |
| NtGetContextThread | GetThreadContext |
| NtGetCurrentProcessorNumber | |
| NtGetCurrentProcessorNumberEx | |
| NtGetDevicePowerState | GetDevicePowerState |
| NtGetMUIRegistryInfo | |
| NtGetNextProcess | |
| NtGetNextThread | |
| NtGetNlsSectionPtr | |
| NtGetNotificationResourceManager | |
| NtGetPlugPlayEvent | |
| NtGetWriteWatch | GetWriteWatch |
| NtImpersonateAnonymousToken | |
| NtImpersonateClientOfPort | |
| NtImpersonateThread | |
| NtInitializeEnclave | |
| NtInitializeNlsFiles | |
| NtInitializeRegistry | |
| NtInitiatePowerAction | SetSystemPowerState |
| NtIsProcessInJob | IsProcessInJob |
| NtIsSystemResumeAutomatic | IsSystemResumeAutomatic |
| NtIsUILanguageComitted | |
| NtListTransactions | |
| NtListenPort | |
| NtLoadDriver | |
| NtLoadEnclaveData | |
| NtLoadHotPatch | |
| NtLoadKey | RegLoadKey |
| NtLoadKey2 | |
| NtLoadKeyEx | |
| NtLockFile | |
| NtLockProductActivationKeys | |

| | |
|---|---|
| NtLockRegistryKey | |
| NtLockVirtualMemory | |
| NtMakePermanentObject | |
| NtMakeTemporaryObject | |
| NtManagePartition | |
| NtMapCMFModule | |
| NtMapUserPhysicalPages | |
| NtMapUserPhysicalPagesScatter | |
| NtMapViewOfSection | |
| NtMarshallTransaction | |
| NtModifyBootEntry | |
| NtModifyDriverEntry | |
| NtNotifyChangeDirectoryFile | |
| NtNotifyChangeDirectoryFileEx | |
| NtNotifyChangeKey | RegNotifyChangeKeyValue |
| NtNotifyChangeMultipleKeys | |
| NtNotifyChangeSession | |
| NtOpenDirectoryObject | BaseGetNamedObjectDirectory |
| NtOpenEnlistment | |
| NtOpenEvent | OpenEvent |
| NtOpenEventPair | |
| NtOpenFile | OpenFile |
| NtOpenIoCompletion | |
| NtOpenJobObject | OpenJobObject |
| NtOpenKey | RegOpenKey |
| NtOpenKeyEx | RegOpenKeyEx |
| NtOpenKeyTransacted | |
| NtOpenKeyTransactedEx | |
| NtOpenKeyedEvent | |
| NtOpenMutant | OpenMutex |
| NtOpenObjectAuditAlarm | ObjectOpenAuditAlarm |
| NtOpenPartition | |
| NtOpenPrivateNamespace | |
| NtOpenProcess | OpenProcess |
| NtOpenProcessToken | OpenProcessToken |
| NtOpenProcessTokenEx | |
| NtOpenRegistryTransaction | |
| NtOpenResourceManager | |
| NtOpenSection | OpenFileMapping |
| NtOpenSemaphore | OpenSemaphore |
| NtOpenSession | |
| NtOpenSymbolicLinkObject | |
| NtOpenThread | OpenThread |
| NtOpenThreadToken | OpenThreadToken |
| NtOpenThreadTokenEx | |
| NtOpenTimer | OpenWaitableTimer |
| NtOpenTransaction | |
| NtOpenTransactionManager | |

| | |
|---|---|
| NtPlugPlayControl | |
| NtPowerInformation | GetSystemPowerStatus |
| NtPrePrepareComplete | |
| NtPrePrepareEnlistment | |
| NtPrepareComplete | |
| NtPrepareEnlistment | |
| NtPrivilegeCheck | PrivilegeCheck |
| NtPrivilegeObjectAuditAlarm | ObjectPrivilegeAuditAlarm |
| NtPrivilegedServiceAuditAlarm | |
| NtPropagationComplete | |
| NtPropagationFailed | |
| NtProtectVirtualMemory | VirtualProtect(Ex) |
| NtPullTransaction | |
| NtPulseEvent | PulseEvent |
| NtQueryAttributesFile | |
| NtQueryAuxiliaryCounterFrequency | |
| NtQueryBootEntryOrder | |
| NtQueryBootOptions | |
| NtQueryDebugFilterState | |
| NtQueryDefaultLocale | GetSystemDefaultLCID, GetUserDefaultLCID |
| NtQueryDefaultUILanguage | GetUserDefaultUILanguage |
| NtQueryDirectoryFile | |
| NtQueryDirectoryFileEx | |
| NtQueryDirectoryObject | |
| NtQueryDriverEntryOrder | |
| NtQueryEaFile | |
| NtQueryEvent | QueryMemoryResourceNotification |
| NtQueryFullAttributesFile | GetFileAttributesEx |
| NtQueryInformationAtom | |
| NtQueryInformationByName | |
| NtQueryInformationEnlistment | |
| NtQueryInformationFile | GetCompressedFileSize, GetFileAttributes, GetFileSize, GetFileSizeEx, GetFileTime, GetMailslotInfo, GetNamedPipeHandleState, GetNamedPipeInfo |
| NtQueryInformationJobObject | |
| NtQueryInformationPort | |
| NtQueryInformationProcess | GetExitCodeProcess |
| NtQueryInformationResourceManager | |
| NtQueryInformationThread | GetExitCodeThread, GetProcessIdOfThread, GetThreadId, GetThreadIOPendingFlag, GetThreadPriority, GetThreadPriorityBoost, GetThreadTimes, GetThreadSelectorEntry |
| NtQueryInformationToken | GetTokenInformation |
| NtQueryInformationTransaction | |
| NtQueryInformationTransactionManager | |
| NtQueryInformationWorkerFactory | |
| NtQueryInstallUILanguage | GetSystemDefaultUILanguage |
| NtQueryIntervalProfile | |
| NtQueryIoCompletion | |
| NtQueryKey | RegQueryInfoKey |

| | |
|---|---|
| NtQueryLicenseValue | |
| NtQueryMultipleValueKey | |
| NtQueryMutant | |
| NtQueryObject | GetHandleInformation |
| NtQueryOpenSubKeys | |
| NtQueryOpenSubKeysEx | |
| NtQueryPerformanceCounter | |
| NtQueryPortInformationProcess | |
| NtQueryQuotaInformationFile | |
| NtQuerySection | |
| NtQuerySecurityAttributesToken | |
| NtQuerySecurityObject | |
| NtQuerySecurityPolicy | |
| NtQuerySemaphore | |
| NtQuerySymbolicLinkObject | |
| NtQuerySystemEnvironmentValue | |
| NtQuerySystemEnvironmentValueEx | |
| NtQuerySystemInformation | GetSystemFileCacheSize, GetSystemTimeAdjustment, GetSystemTimes, |
| NtQuerySystemInformationEx | |
| NtQuerySystemTime | |
| NtQueryTimer | |
| NtQueryTimerResolution | |
| NtQueryValueKey | |
| NtQueryVirtualMemory | VirtualQuery |
| NtQueryVolumeInformationFile | GetFileType |
| NtQueryWnfStateData | |
| NtQueryWnfStateNameInformation | |
| NtQueueApcThread | QueueUserAPC |
| NtQueueApcThreadEx | |
| NtRaiseException | |
| NtRaiseHardError | |
| NtReadFile | ReadFile, ReadFileEx |
| NtReadFileScatter | ReadFileScatter |
| NtReadOnlyEnlistment | |
| NtReadRequestData | |
| NtReadVirtualMemory | ReadProcessMemory |
| NtRecoverEnlistment | |
| NtRecoverResourceManager | |
| NtRecoverTransactionManager | |
| NtRegisterProtocolAddressInformation | |
| NtRegisterThreadTerminatePort | |
| NtReleaseCMFViewOwnership | |
| NtReleaseKeyedEvent | |
| NtReleaseMutant | |
| NtReleaseSemaphore | ReleaseSemaphore |
| NtReleaseWorkerFactoryWorker | |
| NtRemoveIoCompletion | GetQueuedCompletionStatus |

| | |
|---|---|
| NtRemoveIoCompletionEx | |
| NtRemoveProcessDebug | |
| NtRenameKey | |
| NtRenameTransactionManager | |
| NtReplaceKey | |
| NtReplacePartitionUnit | |
| NtReplyPort | |
| NtReplyWaitReceivePort | |
| NtReplyWaitReceivePortEx | |
| NtReplyWaitReplyPort | |
| NtRequestDeviceWakeup | RequestDeviceWakeup |
| NtRequestPort | |
| NtRequestWaitReplyPort | |
| NtRequestWakeupLatency | RequestWakeupLatency |
| NtResetEvent | ResetEvent |
| NtResetWriteWatch | ResetWriteWatch |
| NtRestoreKey | |
| NtResumeProcess | |
| NtResumeThread | ResumeThread |
| NtRevertContainerImpersonation | |
| NtRollbackComplete | |
| NtRollbackEnlistment | |
| NtRollbackRegistryTransaction | |
| NtRollbackSavepointTransaction | |
| NtRollbackTransaction | |
| NtRollforwardTransactionManager | |
| NtSaveKey | |
| NtSaveKeyEx | |
| NtSaveMergedKeys | |
| NtSavepointComplete | |
| NtSavepointTransaction | |
| NtSecureConnectPort | |
| NtSerializeBoot | |
| NtSetBootEntryOrder | |
| NtSetBootOptions | |
| NtSetCachedSigningLevel | |
| NtSetCachedSigningLevel2 | |
| NtSetContextThread | SetThreadContext |
| NtSetDebugFilterState | |
| NtSetDefaultHardErrorPort | |
| NtSetDefaultLocale | |
| NtSetDefaultUILanguage | |
| NtSetDriverEntryOrder | |
| NtSetEaFile | |
| NtSetEvent | SetEvent |
| NtSetEventBoostPriority | |
| NtSetHighEventPair | |
| NtSetHighWaitLowEventPair | |

| | |
|---|---|
| NtSetIRTimer | |
| NtSetInformationDebugObject | |
| NtSetInformationEnlistment | |
| NtSetInformationFile | SetEndOfFile, SetFileAttributes, SetFilePointer, SetFileTime, SetFileValidData |
| NtSetInformationJobObject | |
| NtSetInformationKey | |
| NtSetInformationObject | SetHandleInformation |
| NtSetInformationProcess | |
| NtSetInformationResourceManager | |
| NtSetInformationSymbolicLink | |
| NtSetInformationThread | SetThreadAffinityMask, SetThreadIdealProcessor, SetThreadPriority, SetThreadPriorityBoost |
| NtSetInformationToken | |
| NtSetInformationTransaction | |
| NtSetInformationTransactionManager | |
| NtSetInformationVirtualMemory | |
| NtSetInformationWorkerFactory | |
| NtSetIntervalProfile | |
| NtSetIoCompletion | PostQueuedCompletionStatus |
| NtSetIoCompletionEx | |
| NtSetLdtEntries | |
| NtSetLowEventPair | |
| NtSetLowWaitHighEventPair | |
| NtSetQuotaInformationFile | |
| NtSetSecurityObject | RegSetKeySecurity, SetFileSecurity, SetKernelObjectSecurity, SetUserObjectSecurity |
| NtSetSystemEnvironmentValue | |
| NtSetSystemEnvironmentValueEx | |
| NtSetSystemInformation | SetSystemTimeAdjustment |
| NtSetSystemPowerState | |
| NtSetSystemTime | SetLocalTime, SetSystemTime |
| NtSetThreadExecutionState | SetThreadExecutionState |
| NtSetTimer | |
| NtSetTimer2 | |
| NtSetTimerEx | |
| NtSetTimerResolution | |
| NtSetUuidSeed | |
| NtSetValueKey | RegSetValue, RegSetValueEx |
| NtSetVolumeInformationFile | |
| NtSetWnfProcessNotificationEvent | |
| NtShutdownSystem | ExitWindows(Ex), InitiateSystemShutdown(Ex) |
| NtShutdownWorkerFactory | |
| NtSignalAndWaitForSingleObject | SignalObjectAndWait |
| NtSinglePhaseReject | |
| NtStartProfile | |
| NtStartTm | |
| NtStopProfile | |
| NtSubscribeWnfStateChange | |

| | |
|---|---|
| NtSuspendProcess | |
| NtSuspendThread | SuspendThread |
| NtSystemDebugControl | |
| NtTerminateEnclave | |
| NtTerminateJobObject | |
| NtTerminateProcess | |
| NtTerminateThread | TerminateThread, ExitThread |
| NtTestAlert | |
| NtThawRegistry | |
| NtThawTransactions | |
| NtTraceControl | |
| NtTraceEvent | |
| NtTranslateFilePath | |
| NtUmsThreadYield | |
| NtUnloadDriver | |
| NtUnloadKey | |
| NtUnloadKey2 | |
| NtUnloadKeyEx | |
| NtUnlockFile | |
| NtUnlockVirtualMemory | |
| NtUnmapViewOfSection | MapViewOfFile(Ex) |
| NtUnmapViewOfSectionEx | |
| NtUnsubscribeWnfStateChange | |
| NtUpdateWnfStateData | |
| NtVdmControl | |
| NtWaitForAlertByThreadId | |
| NtWaitForDebugEvent | |
| NtWaitForKeyedEvent | |
| NtWaitForMultipleObjects | WaitForMultipleObjects(Ex) |
| NtWaitForMultipleObjects32 | |
| NtWaitForSingleObject | WaitForSingleObject(Ex) |
| NtWaitForWnfNotifications | |
| NtWaitForWorkViaWorkerFactory | |
| NtWaitHighEventPair | |
| NtWaitLowEventPair | |
| NtWorkerFactoryWorkerReady | |
| NtWriteFile | |
| NtWriteFileGather | |
| NtWriteRequestData | |
| NtWriteVirtualMemory | WriteProcessMemory |
| NtYieldExecution | SwitchToThread |

## Interruptions logicielles

Bien que le matériel soit par nature la principale source d'émission de la plupart des interruptions, le noyau Windows génère également des interruptions pour toutes sortes de tâches :

- Traitement des interruptions temporellement non critiques
- Traitement des expirations de minuterie (timer)

- Exécution asynchrone d'une procédure dans le contexte d'un thread donné (APC)
- Prise en charge des opérations asynchrones d'E/S

## DPC (Deferred Procedure Call)

Les appels de procédure différés (DPC, Deferred Procedure Call) sont des interruptions logicielles utilisées de telle sorte à ajourner le traitement d'autres interruptions.

Le noyau emploie des DPC essentiellement pour traiter les expirations de minuterie et les expirations de quantum des threads. Les pilotes de périphérique utilisent quant à eux des DPC en vue de parachever les requêtes d'E/S.

Les DPC sont implémentés sous forme d'objets DPC créés lorsqu'un pilote ou un autre programme s'exécutant dans l'espace noyau en fait la demande. Chaque DPC se matérialise au niveau noyau sous la forme d'une structure KPCD (voir plus bas). Les routines DPC en attente d'exécution sont stockées dans des files spécialisées (il y a une file par processeur), appelées files DPC. Pour une meilleur flexibilité, Windows permet aux pilotes et autre code système d'interagir sur différents aspects de la prise en charge des DPC, leur laissant ainsi la possibilité de contrôler la file à laquelle le système attribue un DPC, l'emplacement du DPC dans la file et le moment où cette dernière est traitée.

Chaque objet périphérique a un objet DPC qui lui est automatiquement assigné. En interne, cette association, concrétisée par le champ nommé Dpc dans la structure DEVICEOBJECT, n'est objectivement complète que quand un pilote enregistre une routine DPC de traitement d'interruption (DpcForIsr), vu que ce n'est qu'à cette condidition que le gestionnaire d'E/S initialise ledit objet Dans l'éventualité où plus d'un DPC est nécessaire, un pilote peut en créer via la routine KeInsertQueueDpc.

Normalement, tout pilote qui a une routine de service d'interruption a au moins une routine DPC de traitement d'interruption, ne serait ce que pour compléter le traitement de quelques-unes des reqûetes d'E/S qui lui sont adressées (comprendre que l'appel à IoCompleteRequest au sein d'une ISR est impossible). En règle générale, la raison fondamentale pour un pilote de s'appuyer sur l'objet DPC par défaut, sur des objets DPC personnalisés, ou sur les deux, dépend de la nature du périphérique que commande le pilote, et des E/S que ce dernier met en oeuvre.

Par défaut, le noyau place les objets DPC à la fin de la file DPC du processeur ayant exécuté le thread initiateur de l'appel, et ce dans l'ordre où ils sont apparus. Un pilote peut toutefois modifier ce comportement en spécifiant une priorité DPC (basse, moyenne ou haute, le standard étant moyenne), et en réclamant l'exécution d'un DPC sur un processeur en particulier (on parle auquel cas de DPC *ciblé*). Quand un DPC a une priorité basse ou moyenne, le noyau insère l'objet DPC à la fin de la file ; si le DPC a une priorité haute, le noyau insère l'objet DPC au début de la file.

Quand la configuration IRQL du processeur est actualisée au niveau DPC/dispatch, le noyau traite les DPC, ce qu'il fait en extrayant les objets DPC de la file du processeur courant jusqu'à ce que la file soit vide (comprendre que le noyau draine la file, sauf s'il en est empêché par une priorité système supérieure), en appelant tour à tour la fonction appropriée. Ce n'est que quand la file sera vide que le noyau laissera l'IRQL chuter au dessous du niveau DPC/dispatch, redonnant ainsi libre cours à l'exécution normale des threads.

Une autre manière pour les priorités DPC d'affecter le comportement du système concerne le drainage de la file DPC, que le noyau initie généralement via une interruption de niveau DPC/dispatch (DISPATCHLEVEL). Le noyau ne génère ce genre d'interruption que si le DPC est adressé au même processeur que celui exécutant l'ISR et a une priorité moyenne ou haute. Si le DPC a une priorité basse, l'émission d'une telle interruption n'est faite que si le nombre de requêtes DPC en suspens pour le processeur dépasse un certain seuil. Dans l'éventualité où le DPC est ciblé et affecté d'une priorité haute, le noyau demande au processeur approprié de drainer immédiatement sa file (une interruption inter-processeur est alors envoyée). Si la priorité est moyenne ou basse, le nombre des DPC mis en file sur le processeur cible doit excéder la taille de la file DPC pour que le noyau déclenche une interruption DPC/dispatch. Le thread inactif draine aussi la file DPC du processeur sur lequel il est exécuté. Pour

plus de clarté vis-à-vis du lien existant entre objets DPC et les priorités qui les animent, le tableau qui suit résume les circonstances donnant lieu au drainage de la file DPC.

| Priorité | Insertion | Émission DPC/dispatch | IPI (DPC ciblé) |
|---|---|---|---|
| Basse | Fin de la file DPC | NON | NON |
| Moyenne | Fin de la file DPC | OUI | NON |
| Haute | Début de la file DPC | OUI | OUI |

Comme les threads mode utilisateur sont exécutés à un niveau IRQL très bas, il y a de fortes chances qu'un DPC interrompe l'exécution d'un thread utilisateur ordinaire. Notez que, au contraire des routines APC exécutées dans le contexte d'un thread donné, les routines DPC le sont sans aucune considération du thread en cours. Cela signifie que, quand une routine DPC est exécutée, elle n'a aucune connaissance de l'espace d'adressage du processus dans lequel elle a pris place. Par conséquent, les routines DPC peuvent solliciter des fonctions noyau, mais pas des services système. Elles ne peuvent pas non être la source de défauts de page ni créer ou attendre des objets dispatcher. Elles peuvent toutefois accéder aux adresses mémoire système non paginées, cela dans la mesure où celles-ci ne génèrent pas de défauts de page et sont mappés quel que soit le processus en cours.

```
lkd> dt nt!_KDPC
   +0x000 TargetInfoAsUlong : Uint4B
   +0x000 Type             : UChar
   +0x001 Importance       : UChar
   +0x002 Number           : Uint2B
   +0x008 DpcListEntry     : _SINGLE_LIST_ENTRY
   +0x010 ProcessorHistory : Uint8B
   +0x018 DeferredRoutine  : Ptr64     void
   +0x020 DeferredContext  : Ptr64 Void
   +0x028 SystemArgument1  : Ptr64 Void
   +0x030 SystemArgument2  : Ptr64 Void
   +0x038 DpcData          : Ptr64 Void
```

- *Routine DPC (DeferredRoutine)* Adresse de la routine à laquelle le DPC est associé. Voir routine KeInitializeDpc, paramètre DeferredRoutine.
- *DpcListEntry* Voir routine KeInsertQueueDpc.
- *Priorité DPC (Importance)* Affecte la position où un objet DPC est inséré dans la file d'attente et si une interruption logicielle de niveau DPC/dispatch est générée ou non lorsque l'objet est mis en file. Voir routine KeSetImportanceDpc.
- *Processeur (Number)* Numéro du processeur sur lequel le DPC doit être mis en file d'attente et exécuté. Voir routine KeSetTargetProcessorDpc.
- *Catégorisation (Type)* Catégorise la structure en tant qu'objet noyau. Voir types énumérés DpcObject et ThreadedDpcObject parmi l'énumération KOBJECTS ; routines KeInitializeDpc et KeInitializeThreadedDpc.

## Gestion des exceptions

Contrairement aux interruptions, qui sont asynchrones et peuvent à ce titre se manifester à n'importe quel moment, les exceptions sont des conditions qui résultent directement de l'exécution du programme en cours, et découlent en cela d'événements dont le caractère particulier requiert une prise en charge spéciale. Exemples d'exception : la division par zéro, les erreurs d'adressage, l'accès à une instruction non conforme, et d'autres.

On distingue 3 types d'exceptions : les trappes, les fautes et les abandons. La classification dans l'une ou l'autre de ces catégories est fonction de ce que l'unité de contrôle du micro-processeur enregistre lorsqu'il lève l'exception au sommet de la pile noyau, à quoi correspond en définitive la possibilité de « rejouer" (comprendre exécuter de nouveau) l'instruction à la source de l'exception.

- Les exceptions de type faute (fault) entrainent que la valeur sauvée du registre pointeur d'instruction est l'adresse de l'instruction fautive. Le gestionnaire d'exception idoine peut dès lors corriger le problème, par

exemple dans le cas d'un défaut de page alimenter la mémoire physique en les trames de page appropriés. Passée cette étape, toutes les circonstances sont normalement réunies afin que l'exécution puisse reprendre sans autres complications.

- Les exceptions de type trappe (trap) induisent que la valeur sauvée du registre pointeur d'instruction est l'adresse de l'instruction suivant celle fautive. Les trappes sont principalement utilisées dans le cadre du débogage.

- Les exceptions de type abandon (abort) correspondent à une erreur grave. En raison de la nature des dysfonctionnements auxquels fait à cette occasion face l'unité de contrôle du micro-processeur, l'opération en cause ne peut pas clairement être identifiée (il est autrement dit impossible de remonter à l'origine des événements qui se sont accomplis) et le gestionnaire d'exception d'abandon correspondant n'a d'autre choix que mettre fin au processus affecté.

Windows emploie afin d'assurer aux logiciels un environnement optimal une forme spéciale de gestion des exceptions, dit traitement structuré (SEH, St ructured Exception Handling), qui permet aux applications de garder le contrôle en cas d'exception, et de la sorte remédier aux conditions dont un dysfonctionnement émane. Notez que, même si le traitement des exceptions peut s'effectuer dans un langage de programmation muni des extensions idoines (c'est le cas, de C++, via le bloc try-except), il s'agit avant tout d'un mécanisme système, qui n'est donc pas spécifique au langage - ni, par ailleurs, à la plateforme matérielle.

Sur les systèmes x86, toutes les exceptions ont des numéros d'interruption prédéfinis dans l'IDT. Le tableau suivant montre les exceptions x86 avec leurs numéros d'interruption associés.

***Tableau 20.2. Exceptions x86***

| No. | Exception |
|---|---|
| 0 | Erreur de division |
| 1 | Débogage |
| 2 | NMI |
| 3 | Point d'arrêt |
| 4 | Débordement (overflow) |
| 5 | Vérification de limites |
| 6 | Code d'opération non valide |
| 7 | Périphérique non disponible |
| 8 | Faute double |
| 9 | Débordement de coprocesseur |
| 10 | TSS (Task State Segment) non valide |
| 11 | Segment non présent |
| 12 | Exception de pile |
| 13 | Protection générale |
| 14 | Défaut de page |
| 15 | Réservé |
| 16 | Virgule flottante |
| 17 | Contrôle d'alignement |

Quand une exception a lieu, se produit alors en interne une chaîne d'événements de plus ou moins grande ampleur. En premier lieu, le processeur transfère le contrôle au gestionnaire d'interceptions noyau, lequel crée une trame d'interception renfermant toutes les informations requises pour reprendre l'exécution au stade où elle s'était arrêté si l'exception est résolue. Le gestionnaire d'interceptions crée en plus de cela un enregistrement exception qui contient la cause de l'exception, et passe le relais au dispatcher d'exceptions. Si l'exception s'est produite en mode noyau, le dispatcher d'exceptions se contente de solliciter le gestionnaire d'exceptions approprié pour traiter l'exception. Dans l'éventualité où l'exception est survenue en mode utilisateur, le gestionnaire d'exceptions s'appuie sur le sous-système Windows.

## Traitement structuré des exceptions (SEH, Structured Exception Handling)

Pour offrir aux applications la possibilité de mieux anticiper les dysfonctionnements susceptibles de se produire durant leur exécution, Windows implémente un modèle spécial de gestion des exceptions, dit traitement structuré des exceptions (SEH, Structured Exception Handling), via lequel prendre efficacement en charge différentes situations exceptionnelles, telles que les fautes d'accès ou les opérations invalides.

L'une des plus-values les plus manifestes du traitement structuré des exceptions est de permettre aux applications de garder le contrôle suite à des événements qui termineraient sinon leur exécution, cela afin (dans l'idéal) d'apporter une solution au problème qui est survenu, au pire d'en minimiser les effets (libération correcte des ressources mémoire, etc.).

Lorsqu'une exception est levée, l'exécution est interrompue et le contrôle donné au gestionnaire d'exceptions le plus récemment installé. Ledit gestionnaire peut effectuer l'une des trois opérations suivantes :

- Il ne reconnaît pas l'exception. Le système est en la circonstance chargée de rechercher un autre gestionnaire susceptible de traiter l'exception.
- Il reconnaît l'exception, mais la fait disparaître (sous entendu l'invisibilise).
- Il reconnaît l'exception et la gère.

Les mécanismes à l'oeuvre dans le traitement structuré des exceptions sont étroitement liés à un processus appelé "déroulement de la pile" (stack unwinding)

Lorsque le handler d'un SEH est appelé, un certain nombre d'informations concernant l'exception sont mis en place sur la pile du thread ayant déclenché l'exception, à savoir les structures que voici : EXCEPTIONRECORD, CONTEXT et enfin EXCEPTIONPOINTERS.

Sur les systèmes x86, les gestionnaires d'exception définis dans la perspective SEH le sont sous forme d'une liste chainée propre à chaque thread. Lorsqu'un nouveau gestionnaire est ajoutée à la liste, le noeud représentant ledit gestionnaire devient la tete de la liste. En interne, chaque gestionnaire est enregistré sous forme d'une structure EXCEPTIONREGISTRATION, dont un des attributs enregistre l'adresse de la fonction appelée à s'exécuter du moment qu'une exception a lieu.

***Tableau 20.3. Traitement structuré des exceptions***

| Élément / Type | Lieu |
|---|---|
| ExceptionList<br>EXCEPTIONREGISTRATIONRECORD | NT_TIB |

# Chapitre 21. Gestion du temps
## Dates et heures DOS

Les systèmes fondées sur MS-DOS, incluant FAT32, ZIP, et d'autres, enregistrent dates et heures sous une forme compacte.

***Tableau 21.1. Format d'enregistrement de l'heure***

| Poids du bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Signification | h | h | h | h | h | m | m | m | m | m | m | s | s | s | s | s |

Dans ce format, hhhhh correspond au nombre d'heures écoulées depuis 0 heure, mmmmmm au nombre de minutes écoulées depuis le début de l'heure et sssss à la moitié du nombre de secondes écoulées depuis le début de la minute. Exemple avec la valeur 8DC7 :

| Poids du bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Signification | h | h | h | h | h | m | m | m | m | m | m | s | s | s | s | s |
| Hexa | 8 | | | | D | | | | C | | | | 7 | | | |
| Binaire | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |

Dans cet exemple, hhhhh vaut 10001 (17), mmmmmm vaut 101110 (46), et sssss vaut 00111 (7). L'heure est donc 17:46:14. Une logique similaire est appliquée en ce qui concerne la représentation des dates.

***Tableau 21.2. Format d'enregistrement de la date***

| Poids du bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Signification | a | a | a | a | a | a | m | m | m | m | j | j | j | j | j | |

Ici, aaaaaaa correspond au nombre d'années écoulées depuis 1980, mmmm au numéro du mois dans l'année et jjjjj au nombre de jours depuis le début du mois. Exemple avec la valeur 4B52 :

| Poids du bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Signification | a | a | a | a | a | a | m | m | m | m | j | j | j | j | j | |
| Hexa | 5 | | | | 2 | | | | 4 | | | | B | | | |
| Binaire | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |

Dans cet exemple, aaaaaaa vaut 0101001 (41), mmmm vaut 0010 (2), et jjjjj vaut 01011 (11), à quoi correspond 2021-02-11.

# Chapitre 22. Techniques de débogage

## Pseudo registres

Les tailles, les noms et les comportements des registres d'un processeur dépendent étroitement de la plateforme matérielle sous-jacente. Pour surmonter ce problème, le débogueur fournit divers pseudo-registres, qui restent valides quel que soit l'environnement où a pris forme la session de déboggage. Exprimés sous forme de noms symboliques, ces pseudos registres sont définis en tant que variables dont les contenus ont une taille et reflètent une signification identique à celle qu'elles ont dans leur environnement natif. Par exemple, le pseudo-registre $ip existe en parallèle du registre pointeur d'instruction, tel que perçu dans l'architecture cible, à savoir le registre eip (32 bits) sur les systèmes x86, et rip (64 bits) sur les systèmes x64.

Le débogueur gère de façon automatique plusieurs pseudo-registres, certains laissés à discrétion de l'utilisateur, étiquetés de $t0 à $t19, tandis que d'autres ont de facto des valeurs significatives : pointeur d'instruction, ID de processus, région de contrôle du processeur, etc. Comme avec les registres standard, les noms de pseudo-registre doivent, pour outrepasser la recherche de symboles leur correspondant, être échappés via l'utilisation du caractère arobase (@).

## Débogage par affichage

Si ce n'est certainement pas la plus élégante (ni même souvent en réalité la plus efficace) des solutions envisageables pour répondre à la problématique, fonctions d'impression et messages de trace d'exécution forment une technique de débogage encore la plus largement répandue. Windows prévoit à cet effet un certain nombre de sous-routines, certaines appelables depuis le mode utilisateur, d'autres uniquement depuis le mode noyau.

Dans bon nombre de scénarios, l'information qui résulte de procédures visant au diagnostic est suivie sous la perspective d'un débogueur, a minima par l'intermédiaire d'outils spécialisés dans ce but, par exemple DebugView ou DebugMon, tous deux présentés dans ce livre. Sur une note plus générale, remarquez que les fonctions d'impression (de l'ordre du diagnostic ou non) ont par par nature un cout direct sur les performances. Elles ne devraient pour cette raison jamais apparaitre au sein de composants distribués à grande échelle.

Les fonctions d'impression définies au niveau de la sphère noyau en ce qui concerne le débogage sont les suivantes :

- KdPrint
- KdPrintEx
- DbgPrint
- DbgPrintEx

L'ensemble de ces expressions utilise le même format et les mêmes arguments que la fonction printf du langage C. Les deux premières sont des macros qui génèrent du code uniquement dans la version controlé (checked build) du système d'exploitation. Les deux dernières font référence à des routines présentes quelque soit la configuration. Les versions Ex sont des déclinaisons qui apportent des précisions utiles complémentaires par rapport à leur homologue de base.

## Bug checks

| | |
|---|---|
| 0x00000001 | APC_INDEX_MISMATCH |
| 0x00000002 | DEVICE_QUEUE_NOT_BUSY |
| 0x00000003 | INVALID_AFFINITY_SET |
| 0x00000004 | INVALID_DATA_ACCESS_TRAP |
| 0x00000005 | INVALID_PROCESS_ATTACH_ATTEMPT |
| 0x00000006 | INVALID_PROCESS_DETACH_ATTEMPT |
| 0x00000007 | INVALID_SOFTWARE_INTERRUPT |

| | |
|---|---|
| 0x00000008 | IRQL_NOT_DISPATCH_LEVEL |
| 0x00000009 | IRQL_NOT_GREATER_OR_EQUAL |
| 0x0000000A | IRQL_NOT_LESS_OR_EQUAL |
| 0x0000000B | NO_EXCEPTION_HANDLING_SUPPORT |
| 0x0000000C | MAXIMUM_WAIT_OBJECTS_EXCEEDED |
| 0x0000000D | MUTEX_LEVEL_NUMBER_VIOLATION |
| 0x0000000E | NO_USER_MODE_CONTEXT |
| 0x0000000F | SPIN_LOCK_ALREADY_OWNED |
| 0x00000010 | SPIN_LOCK_NOT_OWNED |
| 0x00000011 | THREAD_NOT_MUTEX_OWNER |
| 0x00000012 | TRAP_CAUSE_UNKNOWN |
| 0x00000013 | EMPTY_THREAD_REAPER_LIST |
| 0x00000014 | CREATE_DELETE_LOCK_NOT_LOCKED |
| 0x00000015 | LAST_CHANCE_CALLED_FROM_KMODE |
| 0x00000016 | CID_HANDLE_CREATION |
| 0x00000017 | CID_HANDLE_DELETION |
| 0x00000018 | REFERENCE_BY_POINTER |
| 0x00000019 | BAD_POOL_HEADER |
| 0x0000001A | MEMORY_MANAGEMENT |
| 0x0000001B | PFN_SHARE_COUNT |
| 0x0000001C | PFN_REFERENCE_COUNT |
| 0x0000001D | NO_SPIN_LOCK_AVAILABLE |
| 0x0000001E | KMODE_EXCEPTION_NOT_HANDLED |
| 0x0000001F | SHARED_RESOURCE_CONV_ERROR |
| 0x00000020 | KERNEL_APC_PENDING_DURING_EXIT |
| 0x00000021 | QUOTA_UNDERFLOW |
| 0x00000022 | FILE_SYSTEM |
| 0x00000023 | FAT_FILE_SYSTEM |
| 0x00000024 | NTFS_FILE_SYSTEM |
| 0x00000025 | NPFS_FILE_SYSTEM |
| 0x00000026 | CDFS_FILE_SYSTEM |
| 0x00000027 | RDR_FILE_SYSTEM |
| 0x00000028 | CORRUPT_ACCESS_TOKEN |
| 0x00000029 | SECURITY_SYSTEM |
| 0x0000002A | INCONSISTENT_IRP |
| 0x0000002B | PANIC_STACK_SWITCH |
| 0x0000002C | PORT_DRIVER_INTERNAL |
| 0x0000002D | SCSI_DISK_DRIVER_INTERNAL |
| 0x0000002E | DATA_BUS_ERROR |
| 0x0000002F | I NSTRUCTION_BUS_ERROR |
| 0x00000030 | SET_OF_INVALID_CONTEXT |
| 0x00000031 | PHASE0_INITIALIZATION_FAILED |
| 0x00000032 | PHASE1_INITIALIZATION_FAILED |
| 0x00000033 | UNEXPECTED_INITIALIZATION_CALL |
| 0x00000034 | CACHE_MANAGER |
| 0x00000035 | NO_MORE_IRP_STACK_LOCATIONS |
| 0x00000036 | DEVICE_REFERENCE_COUNT_NOT_ZERO |
| 0x00000037 | FLOPPY_INTERNAL_ERROR |

| | |
|---|---|
| 0x00000038 | SERIAL_DRIVER_INTERNAL |
| 0x00000039 | SYSTEM_EXIT_OWNED_MUTEX |
| 0x0000003A | SYSTEM_UNWIND_PREVIOUS_USER |
| 0x0000003B | SYSTEM_SERVICE_EXCEPTION |
| 0x0000003C | INTERRUPT_UNWIND_ATTEMPTED |
| 0x0000003D | INTERRUPT_EXCEPTION_NOT_HANDLED |
| 0x0000003E | MULTIPROCESSOR_CONFIGURATION_NOT_SUPPORTED |
| 0x0000003F | NO_MORE_SYSTEM_PTES |
| 0x00000040 | TARGET_MDL_TOO_SMALL |
| 0x00000041 | MUST_SUCCEED_POOL_EMPTY |
| 0x00000042 | ATDISK_DRIVER_INTERNAL |
| 0x00000043 | NO_SUCH_PARTITION |
| 0x00000044 | MULTIPLE_IRP_COMPLETE_REQUESTS |
| 0x00000045 | INSUFFICIENT_SYSTEM_MAP_REGS |
| 0x00000046 | DEREF_UNKNOWN_LOGON_SESSION |
| 0x00000047 | REF_UNKNOWN_LOGON_SESSION |
| 0x00000048 | CANCEL_STATE_IN_COMPLETED_IRP |
| 0x00000049 | PAGE_FAULT_WITH_INTERRUPTS_OFF |
| 0x0000004A | IRQL_GT_ZERO_AT_SYSTEM_SERVICE |
| 0x0000004B | STREAMS_INTERNAL_ERROR |
| 0x0000004C | FATAL_UNHANDLED_HARD_ERROR |
| 0x0000004D | NO_PAGES_AVAILABLE |
| 0x0000004E | PFN_LIST_CORRUPT |
| 0x0000004F | NDIS_INTERNAL_ERROR |
| 0x00000050 | PAGE_FAULT_IN_NONPAGED_AREA |
| 0x00000051 | REGISTRY_ERROR |
| 0x00000052 | MAILSLOT_FILE_SYSTEM |
| 0x00000053 | NO_BOOT_DEVICE |
| 0x00000054 | LM_SERVER_INTERNAL_ERROR |
| 0x00000055 | DATA_COHERENCY_EXCEPTION |
| 0x00000056 | INSTRUCTION_COHERENCY_EXCEPTION |
| 0x00000057 | XNS_INTERNAL_ERROR |
| 0x00000058 | FTDISK_INTERNAL_ERROR |
| 0x00000059 | PINBALL_FILE_SYSTEM |
| 0x0000005A | CRITICAL_SERVICE_FAILED |
| 0x0000005B | SET_ENV_VAR_FAILED |
| 0x0000005C | HAL_INITIALIZATION_FAILED |
| 0x0000005D | UNSUPPORTED_PROCESSOR |
| 0x0000005E | OBJECT_INITIALIZATION_FAILED |
| 0x0000005F | SECURITY_INITIALIZATION_FAILED |
| 0x00000060 | PROCESS_INITIALIZATION_FAILED |
| 0x00000061 | HAL1_INITIALIZATION_FAILED |
| 0x00000062 | OBJECT1_INITIALIZATION_FAILED |
| 0x00000063 | SECURITY1_INITIALIZATION_FAILED |
| 0x00000064 | SYMBOLIC_INITIALIZATION_FAILED |
| 0x00000065 | MEMORY1_INITIALIZATION_FAILED |
| 0x00000066 | CACHE_INITIALIZATION_FAILED |
| 0x00000067 | CONFIG_INITIALIZATION_FAILED |

| | |
|---|---|
| 0x00000068 | FILE_INITIALIZATION_FAILED |
| 0x00000069 | IO1_INITIALIZATION_FAILED |
| 0x0000006A | LPC_INITIALIZATION_FAILED |
| 0x0000006B | PROCESS1_INITIALIZATION_FAILED |
| 0x0000006C | REFMON_INITIALIZATION_FAILED |
| 0x0000006D | SESSION1_INITIALIZATION_FAILED |
| 0x0000006E | SESSION2_INITIALIZATION_FAILED |
| 0x0000006F | SESSION3_INITIALIZATION_FAILED |
| 0x00000070 | SESSION4_INITIALIZATION_FAILED |
| 0x00000071 | SESSION5_INITIALIZATION_FAILED |
| 0x00000072 | ASSIGN_DRIVE_LETTERS_FAILED |
| 0x00000073 | CONFIG_LIST_FAILED |
| 0x00000074 | BAD_SYSTEM_CONFIG_INFO |
| 0x00000075 | CANNOT_WRITE_CONFIGURATION |
| 0x00000076 | PROCESS_HAS_LOCKED_PAGES |
| 0x00000077 | KERNEL_STACK_INPAGE_ERROR |
| 0x00000078 | PHASE0_EXCEPTION |
| 0x00000079 | MISMATCHED_HAL |
| 0x0000007A | KERNEL_DATA_INPAGE_ERROR |
| 0x0000007B | INACCESSIBLE_BOOT_DEVICE |
| 0x0000007C | BUGCODE_NDIS_DRIVER |
| 0x0000007D | INSTALL_MORE_MEMORY |
| 0x0000007E | SYSTEM_THREAD_EXCEPTION_NOT_HANDLED |
| 0x0000007F | UNEXPECTED_KERNEL_MODE_TRAP |
| 0x00000080 | NMI_HARDWARE_FAILURE |
| 0x00000081 | SPIN_LOCK_INIT_FAILURE |
| 0x00000082 | DFS_FILE_SYSTEM |
| 0x00000085 | SETUP_FAILURE |
| 0x0000008B | MBR_CHECKSUM_MISMATCH |
| 0x0000008E | KERNEL_MODE_EXCEPTION_NOT_HANDLED |
| 0x0000008F | PP0_INITIALIZATION_FAILED |
| 0x00000090 | PP1_INITIALIZATION_FAILED |
| 0x00000092 | UP_DRIVER_ON_MP_SYSTEM |
| 0x00000093 | INVALID_KERNEL_HANDLE |
| 0x00000094 | KERNEL_STACK_LOCKED_AT_EXIT |
| 0x00000096 | INVALID_WORK_QUEUE_ITEM |
| 0x00000097 | BOUND_IMAGE_UNSUPPORTED |
| 0x00000098 | END_OF_NT_EVALUATION_PERIOD |
| 0x00000099 | INVALID_REGION_OR_SEGMENT |
| 0x0000009A | SYSTEM_LICENSE_VIOLATION |
| 0x0000009B | UDFS_FILE_SYSTEM |
| 0x0000009C | MACHINE_CHECK_EXCEPTION |
| 0x0000009E | USER_MODE_HEALTH_MONITOR |
| 0x0000009F | DRIVER_POWER_STATE_FAILURE |
| 0x000000A0 | INTERNAL_POWER_ERROR |
| 0x000000A1 | PCI_BUS_DRIVER_INTERNAL |
| 0x000000A2 | MEMORY_IMAGE_CORRUPT |
| 0x000000A3 | ACPI_DRIVER_INTERNAL |

| | |
|---|---|
| 0x000000A4 | CNSS_FILE_SYSTEM_FILTER |
| 0x000000A5 | ACPI_BIOS_ERROR |
| 0x000000A7 | BAD_EXHANDLE |
| 0x000000AB | SESSION_HAS_VALID_POOL_ON_EXIT |
| 0x000000AC | HAL_MEMORY_ALLOCATION |
| 0x000000AD | VIDEO_DRIVER_DEBUG_REPORT_REQUEST |
| 0x000000B1 | BGI_DETECTED_VIOLATION |
| 0x000000B4 | VIDEO_DRIVER_INIT_FAILURE |
| 0x000000B8 | ATTEMPTED_SWITCH_FROM_DPC |
| 0x000000B9 | CHIPSET_DETECTED_ERROR |
| 0x000000BA | SESSION_HAS_VALID_VIEWS_ON_EXIT |
| 0x000000BB | NETWORK_BOOT_INITIALIZATION_FAILED |
| 0x000000BC | NETWORK_BOOT_DUPLICATE_ADDRESS |
| 0x000000BD | INVALID_HIBERNATED_STATE |
| 0x000000BE | ATTEMPTED_WRITE_TO_READONLY_MEMORY |
| 0x000000BF | MUTEX_ALREADY_OWNED |
| 0x000000C1 | SPECIAL_POOL_DETECTED_MEMORY_CORRUPTION |
| 0x000000C2 | BAD_POOL_CALLER |
| 0x000000C4 | DRIVER_VERIFIER_DETECTED_VIOLATION |
| 0x000000C5 | DRIVER_CORRUPTED_EXPOOL |
| 0x000000C6 | DRIVER_CAUGHT_MODIFYING_FREED_POOL |
| 0x000000C7 | TIMER_OR_DPC_INVALID |
| 0x000000C8 | IRQL_UNEXPECTED_VALUE |
| 0x000000C9 | DRIVER_VERIFIER_IOMANAGER_VIOLATION |
| 0x000000CA | PNP_DETECTED_FATAL_ERROR |
| 0x000000CB | DRIVER_LEFT_LOCKED_PAGES_IN_PROCESS |
| 0x000000CC | PAGE_FAULT_IN_FREED_SPECIAL_POOL |
| 0x000000CD | PAGE_FAULT_BEYOND_END_OF_ALLOCATION |
| 0x000000CE | DRIVER_UNLOADED_WITHOUT_CANCELLING_PENDING_-OPERATIONS |
| 0x000000CF | TERMINAL_SERVER_DRIVER_MADE_INCORRECT_MEMORY_-REFERENCE |
| 0x000000D0 | DRIVER_CORRUPTED_MMPOOL |
| 0x000000D1 | DRIVER_IRQL_NOT_LESS_OR_EQUAL |
| 0x000000D2 | BUGCODE_ID_DRIVER |
| 0x000000D3 | DRIVER_PORTION_MUST_BE_NONPAGED |
| 0x000000D4 | SYSTEM_SCAN_AT_RAISED_IRQL_CAUGHT_IMPROPER_DRIVER_-UNLOAD |
| 0x000000D5 | DRIVER_PAGE_FAULT_IN_FREED_SPECIAL_POOL |
| 0x000000D6 | DRIVER_PAGE_FAULT_BEYOND_END_OF_ALLOCATION |
| 0x000000D7 | DRIVER_UNMAPPING_INVALID_VIEW |
| 0x000000D8 | DRIVER_USED_EXCESSIVE_PTES |
| 0x000000D9 | LOCKED_PAGES_TRACKER_CORRUPTION |
| 0x000000DA | SYSTEM_PTE_MISUSE |
| 0x000000DB | DRIVER_CORRUPTED_SYSPTES |
| 0x000000DC | DRIVER_INVALID_STACK_ACCESS |
| 0x000000DE | POOL_CORRUPTION_IN_FILE_AREA |
| 0x000000DF | I MPERSONATING_WORKER_THREAD |
| 0x000000E0 | ACPI_BIOS_FATAL_ERROR |

| | |
|---|---|
| 0x000000E1 | WORKER_THREAD_RETURNED_AT_BAD_IRQL |
| 0x000000E2 | MANUALLY_INITIATED_CRASH |
| 0x000000E3 | RESOURCE_NOT_OWNED |
| 0x000000E4 | WORKER_INVALID |
| 0x000000E6 | DRIVER_VERIFIER_DMA_VIOLATION |
| 0x000000E7 | INVALID_FLOATING_POINT_STATE |
| 0x000000E8 | INVALID_CANCEL_OF_FILE_OPEN |
| 0x000000E9 | ACTIVE_EX_WORKER_THREAD_TERMINATION |
| 0x000000EA | THREAD_STUCK_IN_DEVICE_DRIVER |
| 0x000000EB | DIRTY_MAPPED_PAGES_CONGESTION |
| 0x000000EC | SESSION_HAS_VALID_SPECIAL_POOL_ON_EXIT |
| 0x000000ED | UNMOUNTABLE_BOOT_VOLUME |
| 0x000000EF | CRITICAL_PROCESS_DIED |
| 0x000000F1 | SCSI_VERIFIER_DETECTED_VIOLATION |
| 0x000000F2 | HARDWARE_INTERRUPT_STORM |
| 0x000000F3 | DISORDERLY_SHUTDOWN |
| 0x000000F4 | CRITICAL_OBJECT_TERMINATION |
| 0x000000F5 | FLTMGR_FILE_SYSTEM |
| 0x000000F6 | PCI_VERIFIER_DETECTED_VIOLATION |
| 0x000000F7 | DRIVER_OVERRAN_STACK_BUFFER |
| 0x000000F8 | RAMDISK_BOOT_INITIALIZATION_FAILED |
| 0x000000F9 | DRIVER_RETURNED_STATUS_REPARSE_FOR_VOLUME_OPEN |
| 0x000000FA | HTTP_DRIVER_CORRUPTED |
| 0x000000FC | ATTEMPTED_EXECUTE_OF_NOEXECUTE_MEMORY |
| 0x000000FD | DIRTY_NOWRITE_PAGES_CONGESTION |
| 0x000000FE | BUGCODE_USB_DRIVER |
| 0x000000FF | RESERVE_QUEUE_OVERFLOW |
| 0x00000100 | LOADER_BLOCK_MISMATCH |
| 0x00000101 | CLOCK_WATCHDOG_TIMEOUT |
| 0x00000102 | DPC_WATCHDOG_TIMEOUT |
| 0x00000103 | MUP_FILE_SYSTEM |
| 0x00000104 | AGP_INVALID_ACCESS |
| 0x00000105 | AGP_GART_CORRUPTION |
| 0x00000106 | AGP_ILLEGALLY_REPROGRAMMED |
| 0x00000108 | THIRD_PARTY_FILE_SYSTEM_FAILURE |
| 0x00000109 | CRITICAL_STRUCTURE_CORRUPTION |
| 0x0000010A | APP_TAGGING_INITIALIZATION_FAILED |
| 0x0000010C | FSRTL_EXTRA_CREATE_PARAMETER_VIOLATION |
| 0x0000010D | WDF_VIOLATION |
| 0x0000010E | VIDEO_MEMORY_MANAGEMENT_INTERNAL |
| 0x0000010F | RESOURCE_MANAGER_EXCEPTION_NOT_HANDLED |
| 0x00000111 | RECURSIVE_NMI |
| 0x00000112 | MSRPC_STATE_VIOLATION |
| 0x00000113 | VIDEO_DXGKRNL_FATAL_ERROR |
| 0x00000114 | VIDEO_SHADOW_DRIVER_FATAL_ERROR |
| 0x00000115 | AGP_INTERNAL |
| 0x00000116 | VIDEO_TDR_ERROR |
| 0x00000117 | VIDEO_TDR_TIMEOUT_DETECTED |

| | |
|---|---|
| 0x00000119 | VIDEO_SCHEDULER_INTERNAL_ERROR |
| 0x0000011A | EM_INITIALIZATION_FAILURE |
| 0x0000011B | DRIVER_RETURNED_HOLDING_CANCEL_LOCK |
| 0x0000011C | ATTEMPTED_WRITE_TO_CM_PROTECTED_STORAGE |
| 0x0000011D | EVENT_TRACING_FATAL_ERROR |
| 0x0000011E | TOO_MANY_RECURSIVE_FAULTS |
| 0x0000011F | INVALID_DRIVER_HANDLE |
| 0x00000120 | BITLOCKER_FATAL_ERROR |
| 0x00000121 | DRIVER_VIOLATION |
| 0x00000122 | WHEA_INTERNAL_ERROR |
| 0x00000123 | CRYPTO_SELF_TEST_FAILURE |
| 0x00000124 | WHEA_UNCORRECTABLE_ERROR |
| 0x00000125 | NMR_INVALID_STATE |
| 0x00000126 | NETIO_INVALID_POOL_CALLER |
| 0x00000127 | PAGE_NOT_ZERO |
| 0x00000128 | WORKER_THREAD_RETURNED_WITH_BAD_IO_PRIORITY |
| 0x00000129 | WORKER_THREAD_RETURNED_WITH_BAD_PAGING_IO_PRIORITY |
| 0x0000012A | MUI_NO_VALID_SYSTEM_LANGUAGE |
| 0x0000012B | FAULTY_HARDWARE_CORRUPTED_PAGE |
| 0x0000012C | EXFAT_FILE_SYSTEM |
| 0x0000012D | VOLSNAP_OVERLAPPED_TABLE_ACCESS |
| 0x0000012E | INVALID_MDL_RANGE |
| 0x0000012F | VHD_BOOT_INITIALIZATION_FAILED |
| 0x00000130 | DYNAMIC_ADD_PROCESSOR_MISMATCH |
| 0x00000131 | INVALID_EXTENDED_PROCESSOR_STATE |
| 0x00000132 | RESOURCE_OWNER_POINTER_INVALID |
| 0x00000133 | DPC_WATCHDOG_VIOLATION |
| 0x00000134 | DRIVE_EXTENDER |
| 0x00000135 | REGISTRY_FILTER_DRIVER_EXCEPTION |
| 0x00000136 | VHD_BOOT_HOST_VOLUME_NOT_ENOUGH_SPACE |
| 0x00000137 | WIN32K_HANDLE_MANAGER |
| 0x00000138 | GPIO_CONTROLLER_DRIVER_ERROR |
| 0x00000139 | KERNEL_SECURITY_CHECK_FAILURE |
| 0x0000013A | KERNEL_MODE_HEAP_CORRUPTION |
| 0x0000013B | PASSIVE_INTERRUPT_ERROR |
| 0x0000013C | INVALID_IO_BOOST_STATE |
| 0x0000013D | CRITICAL_INITIALIZATION_FAILURE |
| 0x00000140 | STORAGE_DEVICE_ABNORMALITY_DETECTED |
| 0x00000141 | VIDEO_ENGINE_TIMEOUT_DETECTED |
| 0x00000142 | VIDEO_TDR_APPLICATION_BLOCKED |
| 0x00000143 | PROCESSOR_DRIVER_INTERNAL |
| 0x00000144 | BUGCODE_USB3_DRIVER |
| 0x00000145 | SECURE_BOOT_VIOLATION |
| 0x00000147 | ABNORMAL_RESET_DETECTED |
| 0x00000149 | REFS_FILE_SYSTEM |
| 0x0000014A | KERNEL_WMI_INTERNAL |
| 0x0000014B | SOC_SUBSYSTEM_FAILURE |
| 0x0000014C | FATAL_ABNORMAL_RESET_ERROR |

| | |
|---|---|
| 0x0000014D | EXCEPTION_SCOPE_INVALID |
| 0x0000014E | SOC_CRITICAL_DEVICE_REMOVED |
| 0x0000014F | PDC_WATCHDOG_TIMEOUT |
| 0x00000150 | TCPIP_AOAC_NIC_ACTIVE_REFERENCE_LEAK |
| 0x00000151 | UNSUPPORTED_INSTRUCTION_MODE |
| 0x00000152 | INVALID_PUSH_LOCK_FLAGS |
| 0x00000153 | KERNEL_LOCK_ENTRY_LEAKED_ON_THREAD_TERMINATION |
| 0x00000154 | UNEXPECTED_STORE_EXCEPTION |
| 0x00000155 | OS_DATA_TAMPERING |
| 0x00000156 | WINSOCK_DETECTED_HUNG_CLOSESOCKET_LIVEDUMP |
| 0x00000157 | KERNEL_THREAD_PRIORITY_FLOOR_VIOLATION |
| 0x00000158 | ILLEGAL_IOMMU_PAGE_FAULT |
| 0x00000159 | HAL_ILLEGAL_IOMMU_PAGE_FAULT |
| 0x0000015A | SDBUS_INTERNAL_ERROR |
| 0x0000015B | WORKER_THREAD_RETURNED_WITH_SYSTEM_PAGE_PRIORITY_-ACTIVE |
| 0x0000015C | PDC_WATCHDOG_TIMEOUT_LIVEDUMP |
| 0x0000015F | CONNECTED_STANDBY_WATCHDOG_TIMEOUT_LIVEDUMP |
| 0x00000160 | WIN32K_ATOMIC_CHECK_FAILURE |
| 0x00000161 | LIVE_SYSTEM_DUMP |
| 0x00000162 | KERNEL_AUTO_BOOST_INVALID_LOCK_RELEASE |
| 0x00000163 | WORKER_THREAD_TEST_CONDITION |
| 0x00000164 | WIN32K_CRITICAL_FAILURE |
| 0x0000016C | INVALID_RUNDOWN_PROTECTION_FLAGS |
| 0x0000016D | INVALID_SLOT_ALLOCATOR_FLAGS |
| 0x0000016E | ERESOURCE_INVALID_RELEASE |
| 0x00000175 | PREVIOUS_FATAL_ABNORMAL_RESET_ERROR |
| 0x00000178 | ELAM_DRIVER_DETECTED_FATAL_ERROR |
| 0x00000187 | VIDEO_DWMINIT_TIMEOUT_FALLBACK_BDD |
| 0x00000188 | CLUSTER_CSVFS_LIVEDUMP |
| 0x00000189 | BAD_OBJECT_HEADER |
| 0x0000018B | SECURE_KERNEL_ERROR |
| 0x00000190 | WIN32K_CRITICAL_FAILURE_LIVEDUMP |
| 0x00000191 | PF_DETECTED_CORRUPTION |
| 0x00000192 | KERNEL_AUTO_BOOST_LOCK_ACQUISITION_WITH_RAISED_IRQL |
| 0x00000193 | VIDEO_DXGKRNL_LIVEDUMP |
| 0x00000195 | SMB_SERVER_LIVEDUMP |
| 0x00000196 | LOADER_ROLLBACK_DETECTED |
| 0x00000197 | WIN32K_SECURITY_FAILURE |
| 0x00000198 | UFX_LIVEDUMP |
| 0x00000199 | KERNEL_STORAGE_SLOT_IN_USE |
| 0x0000019A | WORKER_THREAD_RETURNED_WHILE_ATTACHED_TO_SILO |
| 0x0000019B | TTM_FATAL_ERROR |
| 0x0000019C | WIN32K_POWER_WATCHDOG_TIMEOUT |
| 0x0000019D | CLUSTER_SVHDX_LIVEDUMP |
| 0x000001C4 | DRIVER_VERIFIER_DETECTED_VIOLATION_LIVEDUMP |
| 0x000001C5 | IO_THREADPOOL_DEADLOCK_LIVEDUMP |
| 0x00000BFE | BC_BLUETOOTH_VERIFIER_FAULT |

| | |
|---|---|
| 0x00000BFF | BC_BTHMINI_VERIFIER_FAULT |
| 0x00020001 | HYPERVISOR_ERROR |
| 0x1000007E | SYSTEM_THREAD_EXCEPTION_NOT_HANDLED_M |
| 0x1000007F | UNEXPECTED_KERNEL_MODE_TRAP_M |
| 0x1000008E | KERNEL_MODE_EXCEPTION_NOT_HANDLED_M |
| 0x100000EA | THREAD_STUCK_IN_DEVICE_DRIVER_M |
| 0x4000008A | THREAD_TERMINATE_HELD_MUTEX |
| 0xC0000218 | STATUS_CANNOT_LOAD_REGISTRY_FILE |
| 0xC000021A | STATUS_SYSTEM_PROCESS_TERMINATED |
| 0xC0000221 | STATUS_IMAGE_CHECKSUM_MISMATCH |
| 0xDEADDEAD | MANUALLY_INITIATED_CRASH |

Windows, compte tenu de la diversité des problèmes susceptibles de le heurter, définit un grand nombre de codes d'erreur. Plusieurs catégories se distinguent en la matière :

- Utilisation incorrecte des différents pools de mémoire système (erreurs d'allocation, IRQL trop élevé pour l'opération demandée, etc.). Entrent dans cette catégorie les codes BAD_POOL_CALLER et DRIVER_CORRUPTED_EXPOOL.
- Corruption des structures de données essentielles à la gestion de la mémoire, par exemple la base de données PFN. Entrent dans cette catégorie les codes MEMORY_MANAGEMENT et PFN_LIST_CORRUPT.
- Paramétrage incohérent ou non valide au regard de la stratégie d'alimentation du système. Entrent dans cette catégorie les codes DRIVER_POWER_STATE_FAILURE, INTERNAL_POWER_ERROR et MANUALLY_INITIATED_POWER_BUTTON_HOLD.
- Matériel ou pilote défectueux. Entrent dans cette catégorie les codes KERNEL_STACK_INPAGE_ERROR, KERNEL_DATA_INPAGE_ERROR et WHEA_UNCORRECTABLE_ERROR.
- Insuffisance des filtres prévus en ce qui la concerne la gestion des exceptions. Entrent dans cette catégorie les codes KMODE_EXCEPTION_NOT_HANDLED, SYSTEM_THREAD_EXCEPTION_NOT_HANDLED et UNEXPECTED_KERNEL_MODE_TRAP.
- Accès à une zone de mémoire non allouée ou interdite dont il résulte une violation d'accès. Entrent dans cette catégorie les codes PAGE_FAULT_IN_NONPAGED_AREA et KERNEL_MODE_EXCEPTION_NOT_HANDLED (STATUS_ ACCESS_VIOLATION).
- Irrégularités dans la gestion de la mémoire vidéo. Entrent dans cette catégorie les codes THREAD_STUCK_IN_DEVICE_DRIVER, VIDEO_MEMORY_MANAGEMENT_INTERNAL et VIDEO_TDR_FAILURE.
- Corruption du système de fichiers offrant sa structure à un volume de stockage. Entrent dans cette catégorie les codes FAT_FILE_SYSTEM et NTFS_FILE_SYSTEM.
- Arrêt inattendu d'un processus ou d'un thread crucial pour le fonctionnement du système. Entrent dans cette catégorie le code CRITICAL_OBJECT_TERMINATION.

## Identification et résolution des incidents

### Options de récupération

Il est possible de contrôler la récupération du système à l'aide des volets Défaillance du système et Écriture des informations de débogage de la boite de dialogue Démarrage et récupération. Les administrateurs font appel font appel auxdites méthodes de sorte à maitriser précisément les actions à prendre lorsque le sytème subit une erreur fatale. Voici, pour commencer, quelles options renferme le volet Défaillance du système :

- *Écrire un événement dans le journal système* Consigne l'erreur dans le journal Système, offrant de la sorte la possibilité à des administrateurs d'enquêter sur les causes du problème (via par exemple l'Observateur d'événements).
- *Redémarrer automatiquement* Sélectionnez cette option pour que le système tente de redémarrer après une erreur fatale.

Servez-vous du menu Écriture des informations de débogage pour choisir le type d'informations de débogage que vous souhaitez que Windows enregistre dans un fichier de vidage. Les options disponibles sont les suivantes :

- *(aucun)*
- *Image mémoire partielle*
- *Image mémoire du noyau*
- *Image mémoire complète*

Par défaut, les emplacements des fichiers générés lors d'un effondrement sont %SystemRoot%\Minidump pour les images mémoire partielles et %SystemRoot%\Memory.dmp pour toutes les autres.

La configuration des informations de débogage est stockée dans le registre Windows sous la clé HKLM\System \CurrentControlSet\Control\CrashControl.

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\CrashControl
    AutoReboot    REG_DWORD    0x1
    CrashDumpEnabled    REG_DWORD    0x7
    DumpFile    REG_EXPAND_SZ    %SystemRoot%\MEMORY.DMP
    LogEvent    REG_DWORD    0x1
    MinidumpDir    REG_EXPAND_SZ    %SystemRoot%\Minidump
    MinidumpsCount    REG_DWORD    0x5
    Overwrite    REG_DWORD    0x1
    DumpFilters    REG_MULTI_SZ    dumpfve.sys
```

## Scénarios

### Dernier événement

Si tous les messages affichés par les environnements de débogage sont à ne pas prendre a la légère, l'un est particulièrement utile pour savoir où et pourquoi le débogueur s'est arrêté. Les cycles de vie amorcés par les composants de système informatiques, comme ceux de sessions de débogage, sont faits d'événement, desquels pour ne pas perdre certaines précieuses données il faut s'informer. La commande .lastevent (Display Last Event) affiche des informations sur le dernier événement de débogage (événement ou exception) qui se soit produit. Le listing x montre un exemple de sortie générée par la commande .lastevent dans deux cas : [....]() Notez que pénétrer dans le débogueur est une action elle-même génératrice d'événements, à savoir le code d'exception 0x80000003 (first chance).

# Glossaire

ABI (Application Binary Interface)
: format binaire respecté par les points d'entrée d'un code compilé.

Entrée de contrôle d'accès (ACE)
: Assignation d'autorisations à une entité de sécurité.

ACL (Access Control List)
: Terme couramment utilisé pour faire référence à une liste de contrôle d'accès discrétionnaire, laquelle constitue un mécanisme de restriction d'autorisation qui identifie les utilisateurs et groupes à qui l'on a accordé ou refusé des droits d'accès sur un objet.

AES (Advanced Encryption Standard)
: L'un des algorithmes les plus utilisés en chiffrement à clé secrète (symétrique). AES a une taille de bloc fixe de 128 bits, et une taille de clé de 128, 192 ou 256 bits.

ANSI (American National Standards Institute)
: Organisme privé à but non lucratif supervisant le développement de normes, en particulier, mais pas exclusivement, dans le domaine de l'informatique. L'ANSI est le représentant des États-Unis à l'ISO.

Interface de programmation d'application (API, Application Programming Interface)
: Ensemble normalisé de fonctions et de structures de données constituant l'interface entre plusieurs composants logiciels quand ils doivent collaborer.

API Windows
: Interfaces de programmation 32/64 bits intégrées aux systèmes de la gamme Microsoft Windows. Couvrant un large éventail de fonctionnalités, l'API Windows constitue la manière privilégiée pour du code mode utilisateur de solliciter le système en vue d'une tâche prédéfinie : créer un processus (CreateProcess), ouvrir un fichier (OpenFile), émettre une requête d'E/S (DeviceIoControl), etc.

APC (Asynchronous Procedure Call)
: Fonctionnalité qui octroie aux programmes utilisateur et aux composants système la possibilité d'exécuter du code dans le contexte d'un thread choisi.

APM (*Advanced Power Management*)
: Gestion avancée de l'alimentation. API qui permet au système d'exploitation de collaborer avec le BIOS en vue de gérer la consommation électrique de la machine. À l'heure actuelle, la norme utilisée pour la gestion de l'énergie est ACPI, qui a remplacé APM. Le support de APM n'est plus assuré à partir de Windows Vista (2007).

ASCII (American Standard Code for Information Interchange)
: Norme employée pour le codage des caractères alphanumériques.

AWE (Address Windowing Extensions)
: Mécanisme par lequel Windows permet à une application d'aller au-delà de la limite d'espace d'adressage virtuel de processus sur une plateforme 32 bits.

Accès séquentiel
: Modèle de lecture ou d'écriture de données effectué en suivant un ordre préétabli.

Accélération de priorité
: Mécanisme par lequel Windows augmente temporairement la priorité d'ordonnancement d'un thread, améliorant de la sorte sa réactivité.

Administrateur
: Personne en charge de la maintenance d'un système informatique (même réduit à un seul poste). Les tâches qui relèvent usuellement de l'administrateur incluent l'installation et le suivi (mise à jour) des logiciels, la création de nouveaux utilisateurs, la mise en oeuvre de la sécurité (configuration des autorisations, sauvegardes, etc.).

Adresse mémoire
: Nombre spécifiant un emplacement à l'intérieur d'un dispositif de mémorisation au niveau duquel sont stockées des données. Les adresses mémoire sont souvent classées en adresses virtuelles ou physiques, selon qu'elles désignent la mémoire physique ou la mémoire virtuelle.

Adresse virtuelle
: Valeur utilisée par un processus de sorte à accéder à la mémoire, traduite à partir de structures de contrôle internes en emplacement physique réel.

Affinité processeur
: Ensemble des processeurs sur lesquels un thread ou un processus est autorisé à s'exécuter.

Aide contextuelle
: Fonction d'aide sélective d'un logiciel d'application qui permet d'obtenir des informations relatives à une action que l'utilisateur souhaite entreprendre.

Algorithme
: Ensemble d'opérations ou d'instructions permettant d'effectuer des calculs ou de résoudre des problèmes.

Anneau
: Niveau de privilège défini dans quelques architectures de processeur, dont x86 et x64, pour protéger le code et les données du système contre les dommages, intentionnels ou non, émanant de code de moindre privilège.

Antémémoire
: Mémoire ultra rapide destinée à accélérer l'accès aux données les plus fréquemment utilisées.

Application de premier plan
: Processus qui possède le thread propriétaire de la fenêtre ayant le focus.

Appel système
: Mécanisme fondamental permettant à un programme s'exécutant en espace utilisateur de solliciter un service offert par le noyau, tel que l'accès au système de fichiers, la gestion des processus ou la mémoire virtuelle.

Application native
: Application consommatrice uniquement des API de services système fournies par Ntdll (hors, donc, le cadre du sous-système Windows). Exemples d'application native : Gestionnaire de session (Smss), autochk.

Arbre
: Représentation d'un ensemble d'objets sous forme hiérarchique. Un arbre informatique a une racine, des branches représentant la hiérarchie, et des feuilles qui sont les objets.

Arborescence
: Organisation hiérarchique des répertoires et des fichiers, comparable à la structure d'un arbre : à partir d'un répertoire racine se déploient divers sous-répertoires, eux-mêmes pouvant se subdiviser et/ou accueillir des fichiers.

Argument
: Valeur transmise à une fonction ou une application, éventuellement renvoyée par ces dernières au moment de l'exécution.

Arrière-plan
: Se dit d'un processus ou d'une tache (thread) dès lors que l'entité n'est pas sous le contrôle direct d'un utilisateur et n'interagit pas avec lui. C'est typiquement le cas des *démons* Unix ou des *services* Windows.

Atomique
: Se dit d'une opération dès lors que celle-ci a la particularité d'être insécable. Une large majorité des architectures de processeur actuelles disposent d'une instruction spécialisée qui teste, puis assigne, de façon atomique, le contenu d'un emplacement mémoire, ouvrant de la sorte la voie à la conception à de verrous qui protègent les ressources des accès concurrents.

Authenticode
: Schéma de signature numérique basé sur procédés cryptographiques standards (PKCS #7, X.509) utilisé pour déterminer l'origine et l'intégrité des binaires logiciels.

Authentification
: Consiste, pour un système informatique, à vérifier l'identité d'une personne ou d'un ordinateur afin d'autoriser l'accès de cette entité à des ressources.

BIOS (Basic Input Output System)
: Ensemble de fonctions spécialisées, contenu dans la mémoire morte de l'ordinateur, lui permettant d'effectuer des opérations élémentaires lors de sa mise sous tension, par exemple la lecture d'un secteur sur un disque. Par extension, le terme est souvent utilisé pour décrire l'ensemble du micrologiciel (*firmware*) équipant la carte mère.

Bitmap
: Mécanisme de représentation compact qu'emploie Windows en vue de pister l'état d'un ensemble d'éléments à l'aide d'une séquence de bits, chaque bit indiquant si l'élément correspondant est libre ou occupé. Sont de la sorte surveillés les pages de la mémoire virtuelle, les clusters d'un disque, etc.

Balance set manager
: Thread système chargé de contrôler et éventuellement initier divers événements liés à la gestion de la mémoire.

Barre d'état
: Partie située en bas de l'écran dans certains programmes Windows qui affiche des informations sur l'état du programme ou les données traitées à ce moment.

Barre d'outil
: Barre contenant des boutons permettant d'accéder rapidement à certaines fonctions d'un logiciel.

Barre des tâches
: Barre qui apparaît en bas de l'écran sous Windows. Cette barre comporte le bouton Démarrer, l'horloge, les icônes de tous les programmes résidents en mémoire à ce moment et des raccourcis pour accéder rapidement à certains fonctions du système d'exploitation.

Barrière
: Primitive de synchronisation qui fixe le réveil d'un nombre préétabli de threads sur un point de rencontre donné.

Basculement de contexte
: Dans le cadre de l'ordonnancement des threads, procédure qui consiste à mémoriser l'état volatile de la machine associé au thread en cours d'exécution et à restaurer (charger) l'état volatile d'un autre thread.

Base de données du dispatcher
: Ensemble de structures de contrôle que gère Windows en vue d'assurer le suivi des threads en attente d'exécution et de l'état des processeurs du système, lui permettant en définitive de prendre des décisions en ce qui concerne l'ordonnancement des threads.

Base de données PFN
: Ensemble de structures de données par lesquelles Windows piste l'état des différentes pages en mémoire physique. Sont définis huit états : active (appelé aussi valide), en transition, en attente (standby), modifiée, modifiée pas d'écriture, libre, mise à zéro et erronée.

Base de données SAM
: Base de données contenant les utilisateurs et groupes définis, avec leurs mots de passe et autres attributs.

Base de registre
: Banque d'informations qui contient les informations de configuration des composants matériels et logiciels.

Big endian
: Convention sur l'ordre de stockage en mémoire des octets d'un mot entier. La convention big endian veut que les octets de poids fort soient stockés en premier.

Bit
: Unité la plus simple dans un système de numération, ne pouvant prendre que deux valeurs : zéro (0) ou un (1). Selon le contexte, un bit peut aussi représenter une alternative logique (vrai ou faux), une réponse dichotomique (oui ou non), ou une parité (pair ou impair).

Bit de poids faible
: Par convention, le bit le plus à droite d'une séquence de n bits.

Bit de poids fort
: Par convention, le bit le plus à gauche d'une séquence de n bits.

Bloc
: Groupe de données enregistré ou transmis globalement pour des motifs techniques indépendamment de leur contenu.

Blocage
: Situation dans laquelle deux ou plusieurs processus sont dans l'impossibilité de poursuivre leur exécution car chacun d'eux attend que des ressources soient libérées par l'autre.

Bogue
: Défaut de conception ou de réalisation se manifestant par des anomalies de fonctionnement. Anglais : bug.

Bugcheck
: Evénement qui empêche le système d'exploitation de continuer à fonctionner, le forçant dès lors à s'interrompre brusquement.

Bus
: Ensemble physique de canaux de transmission capables de délivrer simultanément la même information à plusieurs équipements, par exemple les différents éléments d'un ordinateur (processeurs, mémoire vive, cartes de contrôle, périphériques, etc.).

Build contrôlé
: Déclinaison spéciale de Windows, orientée débogage, visant à aider les concepteurs de logiciel à mieux diagnostiquer et solutionner les problèmes survenant dans les applications, voire dans le système d'exploitation lui-même. Anglais : Checked build.

Build libre
: Version commercialisée normale de Windows, générée avec activation de toutes les optimisations de compilation et avec suppression d'une bonne partie des aides utiles pour l'analyse en profondeur du système.

CISC (Complex Instruction Set Computer)
: Se dit d'une architecture processeur possédant un jeu d'instructions comprenant un très grand

nombre de commandes mixées à des modes d'adressages complexes.

CPU (Central Processing Unit, unité centrale de traitement)
: Composant électronique central de tout ordinateur, qui traite successivement (processeurs simple coeur) ou en parallèle (processeurs multi-coeurs) les instructions machine émanant des programmes. Plus la puissance du processeur est élevée, plus l'ordinateur peut exécuter rapidement les commandes. Cette puissance est indiquée par la fréquence d'horloge, qui se mesure en Hertz. Les CPU modernes possèdent plusieurs coeurs.

CSV (Comma-separated values)
: Format informatique ouvert représentant des données tabulaires sous forme de valeurs séparés par un caractère de séparation (virgule, point-virgule sous certaines localisations - dont le français, guillemet, etc.). D'abord essentiellement utilisé autour de logiciels tableur (Microsoft Excel, par exemple), le format a par la suite gagné en popularité auprès de logiciels de toute sorte. Si CSV n'a jamais vraiment fait l'objet d'une spécification formelle, la RFC 4180 en décrit la forme la plus courante.

Cache
: Mémoire ultra rapide qui contient temporairement des copies de données provenant d'une autre source de données, par exemple la RAM, afin de diminuer les temps d'accès.

Carte mère
: Dispositif électronique complexe, essentiellement composé de circuits imprimés et de ports de connexion par le biais desquels interconnecter les composants primaires d'un système informatique, y compris le processeur, la mémoire vive, les disques durs, etc.

Casse
: Forme majuscule (haut de casse) ou minuscule (bas de casse) des lettres.

Checked build
: Voir Build contrôlé.

Chargeur (loader)
: Partie du système d'exploitation ayant pour fonction de reconnaître et de charger en mémoire les fichiers exécutables, bibliothèques, etc.

Chemin d'accès
: Séquence de noms de lecteurs, de répertoires et de sous-dossiers qui conduit à un fichier ou un dossier spécifiques. Voir aussi *chemin absolu* et *chemin relatif*.

Chemin absolu
: Chemin d'accès complet à un objet, par exemple un fichier. Les noms d'accès absolus commencent au niveau le plus élevé de l'espace de noms concerné, dit répertoire racine.

Cliché (snapshot)
: copie d'un ensemble de données.

Client
: Tout programme (ou station de travail sur lequel il s'exécute) qui accède à un serveur et communique avec ce dernier.

Cluster
: Unité d'espace disque offrant la possibilité de stocker des fichiers dans un système de fichiers. Physiquement, un cluster correspond à un ou plusieurs secteurs d'un périphérique de stockage.

Clustering
: Méthode par laquelle le gestionnaire de mémoire résout un défaut de pages en chargeant en mémoire plusieurs pages voisines de la page explicitement référencée.

Coeur (core)
: Sous-ensemble d'un microprocesseur moderne, constitué d'une UAL (Unité arithmétique et logique) indépendante.

Cohérence de cache
: Protocole de communication employée dans les architectures multi-processeurs en vue d'assurer une vue cohérence de la mémoire. Il s'agit en particulier de répercuter les écritures faites par chaque processeur aux autres, cela en modifiant ou en invalidant les lignes de cache communes.

Collision
: Situation dans laquelle deux données font émerger un résultat de hachage identique.

Collision de défauts de page
: Survient quand un thread génère un défaut de page sur une page qui est en cours de rechargement.

Compatibilité
: Degré de capacité d'un composant ou d'un système à échanger des informations avec d'autres composants ou systèmes.

Compatible PC
: Fait référence à la conformité aux spécifications logicielles et matérielles issues des premières incarnations de l'informatique personnelle, lesquelles sont au cour des années devenues des normes de fait dans l'industrie informatique pour les ordinateurs basés sur les microprocesseurs de la famille x86 inventée par Intel. Les systèmes d'exploitation MS-DOS, Windows, OS/2 et GNU/Linux ont notamment été créés pour les compatibles PC. Mac OS X a été également porté sur cette architecture depuis le passage d'Apple à des processeurs Intel en 2006.

Compte utilisateur
: Ensemble d'informations visant à identifier un utilisateur, lui assurant de la sorte un accès personnalisé au système d'exploitation. Ces informations incluent notamment son nom, son mot de passe, les groupes auxquels il appartient ainsi que les droits et autorisations qui lui sont accordés pour l'utilisation des ressources de l'ordinateur.

Compte de service
: Compte d'utilisateur créé explicitement afin de fournir un contexte de sécurité aux différents services.

Concurrence
: Utilisation partagée et simultanée des ressources entre plusieurs utilisateurs ou processus.

Contention
: Phénomène par lequel, du fait de blocage, les temps de réponse moyens d'un système se rallongent considérablement.

Contexte d'exécution
: Environnement d'exécution d'un processus ou d'un thread (registres, zones mémoires…)

Contremarque de temps
: Donnée qui lie une représentation d'une donnée à un temps particulier.

Contrôle d'accès
: Mécanisme permettant de prévenir de l'utilisation non autorisée d'une ressource.

Contrôle d'accès discrétionnaire
: Moyen de limiter l'accès à des ressources en fonction de l'identité des utilisateurs ou des groupes auxquels elles appartiennent.

Copier/coller
: Système permettant de créer des copies d'une information au sein d'une application ou entre deux applications. Le fait de copier un élément (texte, image, etc.) le place temporairement dans une zone appelée le Presse-papiers, à partir de laquelle il sera collé (comprendre reproduit) dans un autre document.

Coprocesseur
: Circuit électronique conçu de sorte à ajouter une fonction au processeur central ou l'épauler pour un type de calcul précis. On compte des coprocesseurs arithmétiques (pour le calcul en virgule flottante), graphiques (pour accélérer du rendu 2D ou 3D), spécialisés dans le chiffrement, le traitement du son, l'affichage des animations, etc.

Corbeille
: Emplacement du système de fichiers dans lequel sont stockés les fichiers supprimés (à condition qu'ils n'aient pas été supprimés de façon permanente).

Couche d'abstraction matérielle
: Fournit l'interface de bas niveau vers la plateforme matérielle sur laquelle Windows s'exécute, isolant de la sorte le noyau des caractéristiques internes de l'ordinateur (interfaces d'E/S, contrôleurs d'interruption, mécanismes de communication multi processeur et autres).

Cycle d'horloge
: Unité élémentaire de temps d'un ordinateur, définie par sa fréquence d'horloge. Chaque action d'un microprocesseur requiert au moins un cycle pour s'exécuter.

DACL (Discretionary Access Control List)
: Liste de contrôle utilisée pour les informations de protection discrétionnaire.

DLL (Dynamic-Link Library)
: Ensemble de sous-routines, liées entre elles en tant qu'image binaire, que les applications utilisatrices peuvent charger dynamiquement.

DMA (Direct Memory Access)
: Procédé employé de telle sorte à minimiser l'intervention du microprocesseur lors du transfert d'informations entre un périphérique (port de communication, disque dur, etc.) et la mémoire principale.

DNS (Domain Name System)
: Norme Internet pour l'affectation d'adresses IP à des noms de domaine. Une comparaison courante pour expliquer ce qu'est DNS consiste à dire qu'il se comporte comme un annuaire

téléphonique, traduisant des noms d'hôte facilement compréhensibles (ntoskrnl.org) en adresses IP (217.160.223.229).

DRAM (Dynamic RAM)
: Un des deux principaux types de mémoire. Dans une SRAM, l'information est mémorisée comme la présence ou l'absence d'un courant électrique. Les mémoires SRAM sont généralement assez rapides mais de faible capacité. Elles sont souvent utilisées pour construire des mémoires caches.

Débogueur
: Logiciel spécialisé visant à détecter et diagnostiquer les anomalies qui entravent le bon fonctionnement d'un programme.

Défaut de page
: Événement qui se produit quand un thread référence un emplacement mémoire invalide, soit parce que la page concernée doit être récupérée à partir du disque (défaut de page matériel), soit parce qu'elle réside ailleurs qu'attendu dans la mémoire (défaut de page logiciel).

Défragmentation
: Processus de réorganisation des informations enregistrées par un volume, cela afin de réduire (au mieux éliminer) le phénomène de fragmentation du système de fichier.

Descripteur d'adresse virtuelle (VAD, Virtual Address Descriptor)
: Structures de données sur lesquelles s'appuie le gestionnaire de mémoire pour pister les adresses virtuelles qui ont été réservées dans l'espace d'adressage d'un processus, permettant en définitive de différer la création des tables de page correspondantes.

Descripteur de sécurité
: Structure de données qui renferme des informations de sécurité associées à une ressource sécurisable.

DirectX
: Interface de programmation regroupant des bibliothèques de fonctions destinées au développement d'application ludiques ou multimédia sous Windows.

Dispatcher
: Ensemble des routines établies de sorte à gérer l'ordonnancement des threads.

Dispatcher d'interruptions
: Sous-ensemble du gestionnaire d'interceptions chargé de répondre aux interruptions.

Disque GPT
: Disque dont le schéma de partitionnement est un enregistrement d'amorçage principal (MBR).

Disque GPT
: Disque dont le schéma de partitionnement est une table de partition GUID (GPT).

Disque de base
: Disque qui s'appuie sur le partitionnement style MBR ou GPT.

Domaine
: Ensemble d'ordinateurs et de groupes de sécurité associés partageant des informations d'annuaire, le tout étant géré comme une entité unique.

Droits d'accès accordés
: Ensemble des permissions dont bénéficie un thread sur un objet après validation par le sous-système de sécurité, autorisant un éventail plus ou moins large d'opérations : lecture/écriture, suppression, modification des propriétés, et autres droits génériques et/ou spécifiques à la nature de l'objet concerné (fichiers, processus, etc.).

Droits d'accès désirés
: Ensemble des permissions que requiert un thread vis-à-vis d'un objet, déterminant ainsi les opérations envisagées par son intermédiaire (lecture, écriture, exécution, suppression, synchronisation, etc.).

Dump
: Copie de la mémoire vive et des registres d'un processeur, donnant lieu à un instantané de l'état du système.

EMS (Emergency Management Services)
: Mode de configuration du système où Windows rapporte des informations d'amorçage et accepte des commandes d'administration système via un port série.

Écrivain paresseux (*lazy writer*)
: Ensemble de threads système apparentés au gestionnaire de cache chargés de vidanger le contenu du cache sur le disque, sous forme d'activités conduites en tâche de fond.

État de signal
: Etat d'un objet de synchronisation.

Événement
: Objet doté d'un état persistant (signalé ou non signalé) utilisable pour de la notification ou de la

synchronisation ; désigne aussi une occurrence qui au niveau du système déclenche une action.

Emprunt d'identité
: Mécanisme permettant à un thread d'adopter temporairement l'identité d'un autre utilisateur que celui pour le compte duquel il s'exécute.

Enclave sécurisée
: Composante matérielle intégrée à certains appareils (processeurs, smartphones, ordinateurs), conçue pour offrir un environnement isolé et sécurisé servant d'abri à toutes sortes d'informations sensibles, y compris clés de chiffrement, traces d'identité (données biométriques) et autres secrets.

Enregistrement d'amorçage principal
: L'un des deux schémas de partitionnement d'un disque. Voir schéma de partitionnement.

Entête dispatcher
: Encapsule les capacités de synchronisation dont sont pourvus certains objets du noyau, tels mutex et sémaphores, mais aussi processus, threads, jobs et bien d'autres.

Environnement de bureau
: Ensemble de programmes permettant de manipuler l'ordinateur à travers une interface graphique. Le terme « environnement de bureau » provient de la métaphore du bureau à laquelle fait écho ce type de logiciels.

Espace d'adressage
: Ensemble des adresses mémoire virtuelles utilisables par un processus.

Espace de session
: Sous-ensemble de l'espace système servant à mapper les données spécifiques aux sessions utilisateur.

Exécutable
: Programme dont la prise en charge s'effectue sans plus d'intermédiaires que le système d'exploitation. En tant qu'adjectif, caractérise un fichier susceptible d'être exécuté.

Exécuter
: Donner le départ à un programme.

Exception
: Situation anormale qui se produit pendant l'exécution d'un programme.

Exclusion mutuelle
: Processus par lequel un thread ou processus et un seul est autorisé à accéder à une ressource à un moment donné.

Exécutif
: Sur-ensemble du noyau Windows intégrant les services de base qu'offre le système d'exploitation, tels que le gestionnaire de processus et de threads, le gestionnaire de mémoire virtuelle, le moniteur de références de sécurité, le système d'E/S et le gestionnaire de cache.

FAT (File Allocation Table)
: Table d'allocation des fichiers. Élément essentiel du système de fichiers des ordinateurs exécutant DOS et Windows (de la version 3 à 98), qui décrit l'occupation d'un disque et l'endroit où résident les fichiers.

FIFO (First In First Out)
: Méthode de gestion de files (ou d'autres représentations utilisées en structure de données) où les premiers éléments ajoutés à la file seront les premiers à être récupérés.

Fenêtre
: Portion de l'écran consacrée à l'affichage de tout ou partie d'un logiciel.

Fichier
: Ensemble d'informations stockée dans un support de stockage, depuis lequel il est accessible via un nom. Selon le système de fichier employé, les fichiers peuvent être stockés de différentes manières, avec différents noms de fichiers et différentes façons d'exprimer l'endroit où réside le fichier parmi l'arborescence des répertoires.

Fichier d'amorçage
: Fichier système (nommé $Boot) qui contient le code de démarrage de Windows.

Fichier d'échange
: Fichier grâce auquel Windows simule de la mémoire vive sur le disque dur.

Flot
: Suite d'octets dans un fichier.

Format de système de fichiers
: Définit la manière dont les données des fichiers sont enregistrées sur le support et conditionne les fonctionnalités du système de fichiers (niveau d'intégration de la sécurité, par exemple), de

mêmes que leurs limites (tailles des fichiers et périphériques gérés).

Formatage
: Processus de création d'un système de fichiers sur un volume. Ce n'est qu'à la condition qu'un volume soit formaté qu'il est possible d'y enregistrer des fichiers et des répertoires.

Free build
: Voir Build libre.

Fréquence
: Nombre de signaux par seconde, exprimée le plus souvent en mégahertz et utilisée pour mesurer la vitesse des composants d'un ordinateur (le processeur ou le bus, par exemple).

GDI
: Ensemble de primitives utilisées dans Windows pour la représentation d'objets graphiques ainsi que pour leur transmission aux périphériques de sortie, typiquement un écran ou une imprimante.

GNU/Linux
: Nom générique donné à un système d'exploitation utilisant les utilitaires GNU et le noyau Linux .

GUI (*Graphical User Interface*)
: Interface dont la navigation s'effectue à l'aide d'éléments graphiques (fenêtres, icônes, menus, et ainsi de suite). Ce type d'interface homme-machine s'oppose à la notion de *ligne de commande* où la majorité de l'interaction entre l'utilisateur et l'ordinateur se fait au clavier, sans visualisation élaborée, par l'intermédiaire d'un terminal.

GUID (Globally Unique Identifier)
: Type spécial d'identifiant utilisé parmi un système logiciel à titre de numéro de référence unique dans le contexte dans lequel il est utilisé. Le terme est utilisé à la fois dans le monde Microsoft et dans le monde Unix.

Gestionnaire de processus
: Composant de l'exécutif en charge de superviser le cycle de vie des processus et des threads, depuis leur création jusqu'à leur démantèlement.

Gestionnaire de mémoire
: Composant de l'exécutif qui implémente la mémoire virtuelle paginée à la demande, en donnant à chaque processus l'illusion de disposer d'un très large espace d'adressage, et d'en être de surcroît le seul détenteur.

Gestionnaire de configuration
: Composant de l'exécutif responsable du de la mise en oeuvre et du suivi du Registre Windows.

Gestionnaire d'alimentation
: Composant de l'exécutif qui coordonne les événements électriques de l'ordinateur et généré des des notifications d'E/S de gestion électrique aux pilotes de périphériques concernés.

Gestionnaire d'objets
: Composant de l'exécutif chargé de créer, protéger, suivre et supprimer les objets.

Gestionnaire d'interceptions
: Sous-ensemble du noyau qui fait office de poste d'aiguillage en centralisant les exceptions et interruptions détectés par le processeur et en passant le relais à du code chargé de traiter la condition.

Handle
: Identificateur d'objet. Un processus reçoit un handle vers un objet lorsqu'il le crée ou l'ouvre.

Hertz (Hz)
: Unité de mesure des fréquences. Une fréquence de 1 hertz correspond à un cycle d'horloge par seconde.

Hachage
: Valeur statistiquement unique générée à partir d'un ensemble de données (tel un fichier) à l'aide d'algorithmes cryptographiques. Des données différentes produisent des hachages distincts, permettant ainsi de détecter la corruption ou la falsification. Le système de signatures des pilotes intégré à Windows, notamment, utilise des hachages.

Horloge
: Organe émettant des impulsions permettant à l'ordinateur de se synchroniser.

Hôte
: Désigne potentiellement tout ordinateur qui prend le rôle de source de l'information.

Hyperthread
: Technologie matérielle qui offre un schéma d'optimisation du traitement des threads équivalent à un mode multiprocesseur sur un seul processeur physique.

IA-32
: Architecture Intel 32 bits. Application du 32 bits sur l'architecture x86.

IA-64
: Architecture Intel 64 bits ou Itanium. Correspond à l'implémentation du traitement 64 bits sur l'architecture x86.

ID de processus
: Numéro unique affecté à chaque processus (appelé aussi, en interne, ID client).

ID de thread
: Numéro unique affecté à chaque thread (appelé aussi, en interne, ID client).

INI
: Extension des fichiers de configuration employés par de nombreuses applications Windows.

IPC (Inter-Process Communication)
: Ensemble de méthodes offrant à plusieurs processus la possibilité de communiquer entre eux et/ou de synchroniser leurs actions.

IRP (I/O Request Packet)
: Structure de données par laquelle le noyau contrôle le traitement d'une opération à chaque étape de sa prise en charge. Une large majorité des requêtes d'E/S sont représentées de la sorte.

IRQL
: Mécanisme par l'intermédiaire duquel Windows accorde une importance préférentielle aux interruptions émanant de différentes sources (processeur, clavier, souris, etc.).

ISA (Industry Standard Architecture)
: Ancien format de bus 8 et 16 bits qui a été largement utilisé.

Icône
: Illustration, généralement de petite taille, qui symbolise un programme, un élément ou une commande.

Ingénierie inverse
: Analyse d'un système en vue d'en comprendre les mécanismes internes. Anglais : Reverse engineering.

Interopérabilité
: Ensemble de moyens logiciels et matériels permettant d'assurer une coopération harmonieuse entre plusieurs systèmes hétérogènes.

Interruption
: Signal provenant d'un matériel ou d'un logiciel qui indique qu'un événement nécessite une intervention ou un traitement.

Interruption inter processeur
: Interruption que le noyau émet pour le compte d'un processeur afin de demander à un autre processeur d'accomplir une action, par exemple actualiser son cache TLB.

Interblocage (*deadlock*)
: Situation telle que deux processus (ou plus) se trouvent chacun en attente d'une ressource non partageable déjà allouée à l'autre.

Intel 64
: Technologie d'extension mémoire Intel EM64T (Extended Memory 64 Technology).

Interface utilisateur
: Ensemble de principes, méthodes et moyens par lesquels les programmes interagissent avec l'utilisateur.

Interface
: Dispositif servant de jonction entre deux matériels ou logiciels leur permettant d'échanger des informations par l'adoption de règles communes.

Jeton d'accès
: Structure contenant les éléments et informations d'identification de sécurité d'un processus ou d'un thread : SID, liste des groupes dont l'utilisateur est membre, liste des privilèges qui sont activés et désactivés. Chaque processus a un jeton d'accès principal qu'il hérite, par défaut, du processus créateur.

Joliet
: Système de fichiers pour supports CD-ROM et dérivés, créé par Microsoft pour ses systèmes d'exploitation Windows sur la base du format ISO-9660 et généralisé par la suite sur la plupart des autres systèmes d'exploitation.

Journal
: Fichier dans lequel des évènements conformes à une catégorie spécifique sont consignés au fur et à mesure qu'ils ont lieu.

Journalisation
: Désigne l'enregistrement séquentiel dans un fichier ou une base de données des événements affectant un processus ou système en particulier (application, activité réseau, etc.). Le journal désigne alors le fichier contenant ces enregistrements.

Lecture anticipée intelligente
: Technique permettant de prédire les données qu'un thread est susceptible de lire

prochainement, cela sur la base des données qu'il est en train de lire.

LDT (Local Descriptor Table)
: Structure qui contient des descripteurs de segments mémoire spécifiques à un processus, utilisée principalement dans les systèmes basés sur l'architecture x86 pour gérer l'accès aux segments mémoire en mode protégé.

LPC (Local Procedure Call)
: Mécanisme de communication inter processus.

LSASS (Local Security Authority Subsystem)
: Processus système, exécuté en mode utilisateur, qui est chargé d'authentifier les comptes accédant à un ordinateur.

LSN (Logical Sequence Number)
: Les LSN permettent à NTFS d'identifier les enregistrements écrits dans le fichier journal.

Librairie
: Anglicisme qui désigne couramment les bibliothèques de fonctions (DLL) employées lors de l'exécution d'un programme.

Linux
: Noyau de système d'exploitation compatible Unix développé initialement par Linus Torvalds. Ce projet implique aujourd'hui des milliers de contributeurs de par le monde.

Liste look-aside
: Mécanisme d'allocation de mémoire allouée à partir de blocs de taille fixe. Les listes look-aside peuvent être paginables ou non.

Little endian
: Convention sur l'ordre de stockage en mémoire des octets d'un mot entier. La convention little endian veut que les octets de poids faible soient stockés en premier.

Memory Descriptor List (MDL)
: Structure système décrivant les pages d'un tampon (des adresses virtuelles) par un ensemble d'adresses physiques.

MBR (*Master Boot Record*)
: Nom donné au premier secteur adressable d'un disque dur (cylindre 0, tête 0 et secteur 1, ou secteur 0 en adressage logique). Le secteur de démarrage maître contient le code machine exécutable utilisé pour charger le système d'exploitation, ou éventuellement un autre chargeur de démarrage, et la table des partitions de ce disque dur.

MFT (Master File Table)
: Structure organisationnelle dévolue au stockage des informations concernant fichiers et dossiers dans un volume NTFS.

MMCSS (Multimedia Class Scheduler Service)
: Service mode utilisateur oeuvrant de concert avec le noyau de sorte à renforcer la priorité d'ordonnancement de threads s'exécutant pour le compte de tâches multimédias hautement prioritaires.

MMU (Memory Management Unit)
: Unité matérielle de gestion mémoire, laquelle prend en charge le processus de traduction des adresses virtuelles en adresses physiques.

MSDN (Microsoft Developer Network)
: Programme d'assistance Microsoft destiné aux concepteurs de logiciels. Pour plus d'informations, voir msdn.microsoft.com.

Maître-esclave
: Modèle sur lequel se fonde une distinction entre un composant maitre (par exemple un périphérique, un processus ou un serveur) et un ou plusieurs autres composants qui en sont les esclaves. Le maître donne des ordres à l'esclave qui les exécute.

Mégahertz
: La fréquence d'horloge des microprocesseurs est exprimée en mégahertz, c'est-à-dire en millions de cycles par seconde.

Mémoire morte
: Mémoire contenant des instructions ou des données susceptibles d'être lues, mais pas modifiées.

Mémoire partagée
: Mémoire visible à plus d'un processus.

Mémoire virtuelle
: Mécanisme par lequel le système d'exploitation emploie une partie du disque dur pour étendre la mémoire centrale. Il est ainsi possible d'exécuter un plus grand nombre d'applications et/ou de manipuler des fichiers d'une taille supérieure à celle de la mémoire centrale.

Microprogramme (*firmware*)
: Programme intermédiaire entre le logiciel et le matériel souvent contenu dans une ROM ou une EPROM.

Minuterie (*timer*)
: Mécanisme informant un thread qu'un laps de temps s'est écoulé.

Mode noyau
: Mode d'execution privilégié du processeur dans lequel toute la mémoire et toutes les instructions, y compris celles se rapportant à la configuration de la machine, sont accessibles. Le noyau, l'exécutif Windows ainsi qu'une large majorité des pilotes de périphérique s'exécutent dans ce mode. Voir aussi mode utilisateur.

Mode protégé
: Phase du processus d'amorçage au cours de laquelle le processeur ouvre la possibilité d'un espace d'adressage 32 bits.

Mode sans échec
: Configuration d'amorçage composée du minimum essentiel de pilotes et de services.

Mode utilisateur
: Mode d'exécution du processeur qui, par opposition au mode noyau, ne donne accès qu'à un ensemble limité d'interfaces et ne permet pas d'accéder à l'intégralité des données du système. Voir aussi mode noyau.

Moniteur de références de sécurité (SRM, Security Reference Monitor)
: Composant de l'exécutif en charge du maintien et de la stricte application des stratégies de sécurité sur l'ordinateur local. Il sécurise les ressources du système d'exploitation, en assurant la protection et l'audit des objets.

Montage
: Procédure par laquelle la structure des répertoires d'une partition est rendue visible dans l'espace de nommage d'un système de fichiers.

Mot
: Valeur numérique stockée sur plusieurs octets contigus.

Multicoeur
: Architecture dans laquelle plusieurs processeurs coexistent sur le même circuit intégré.

Multitâche
: Capacité d'un système d'exploitation à partager un même emplacement d'exécution (un même processeur) entre de multiples threads. Il existe deux sortes de multitâche : coopératif et préemptif.

Mutant
: Nom interne du mutex.

Mutex
: Mécanisme de synchronisation ayant pour fonction de protéger une section critique (comprendre un intervalle défini d'instructions) utilisée par plusieurs threads (ou processus), et à laquelle un seul thread à la fois peut accéder.

NTFS
: Système de fichiers avancé conçu pour une utilisation spécifique au sein des systèmes d'exploitation Microsoft Windows. NTFS prend en charge la récupération des systèmes de fichiers, les supports de stockage de très grande taille, les noms de fichiers et de dossiers longs, ainsi que la compression des fichiers et des dossiers.

NUMA (Non Uniform Memory Access)
: Architecture où les accès mémoire sont dirigés, selon l'adresse, vers différents composants mémoire aux coûts d'accès inégaux.

Nanoseconde
: Unité de mesure de temps équivalant à un milliardième de seconde.

NetBEUI (*NetBios Extended User Interface*)
: Extension du protocole NetBIOS fournissant des services de transport de données.

Netbios
: Protocole pour réseaux locaux conçu par IBM puis repris par Microsoft sous le nom NetBEUI.

Nom de volume
: Nom facultatif assignable à une partition en vue de l'identifier plus facilement.

Noyau
: Supervise la manière dont le système d'exploitation s'alimente auprès des ressources incorporées à l'ordinateur (essentiellement le ou les processeurs et la mémoire). Le noyau fournit l'ordonnancement et la ventilation des threads, le traitement des interceptions (trappes, interruptions et exceptions), et la synchronisation multi processeur.

OEM (Original Equipment Manufacturer)
: Désigne un produit vendu sous un autre nom que celui du manufacturier.

Objet de synchronisation
: Objet dont la structure interne le rend utilisable en tant que vecteur pour coordonner l'exécution de plusieurs threads.

Objet dispatcher
: Objet noyau incorporant des capacités de synchronisation qui permettent d'affecter l'ordonnancement de thread. Les objets dispatcher incluent les sémaphores, les mutex, les événements, les minuteries et d'autres objets que les threads peuvent attendre.

Objet job
: Objet Windows qui permet de gérer et manipuler comme un tout un ensemble de processus.

Objet mappage de fichier
: Primitive sur laquelle s'appuie le gestionnaire de mémoire de sorte à implémenter la mémoire partagée (appelé en interne objet section). Voir aussi objet section.

Objet répertoire
: Structure de données sur laquelle s'appuie le gestionnaire d'objets pour implémenter l'espace de noms hiérarchique, dans lequel noyau et composants de l'exécutif entreposent leurs objets.

Octet
: Ensemble ordonné de 8 bits, soit 256 valeurs différentes, comprises entres 0 et 255 (décimal) ou 0 et FF (hexadécimal), pouvant être utilisée pour représenter de nombreux types d'informations, tels qu'une lettre de l'alphabet, un nombre décimal ou tout autre caractère. Anglais : Byte.

PE (Portable Executable)
: Format standard des fichiers exécutables, du code objet et des bibliothèques logicielles (DLL) employés dans les versions 32 et 64 bits du système d'exploitation Windows.

PING (*Packet Internet Groper*)
: Composante du protocole de connexion TCP/IP dont la principale mission consiste à vérifier qu'un paquet de données est en mesure d'être distribué sans erreurs à une adresse donnée. La commande ping, intégré en standard dans une grande majorité de systèmes d'exploitation, permet par extension de s'assurer de l'accessibilité d'une machine à travers un réseau IP.

POSIX
: Ensemble de normes techniques chargées de définir les interfaces de programmation des logiciels destinés à fonctionner sur les variantes de systèmes d'exploitation de type UNIX.

Page
: Dans un système de stockage virtuel, bloc de longueur fixe d'adresses virtuelles contiguës copiées depuis la mémoire vers le disque, et vice versa, à la suite d'une opération de pagination.

Panneau de configuration
: Utilitaire Windows permettant à l'utilisateur de contrôler divers aspects du système d'exploitation ou du matériel, tels que la date et l'heure, les caractéristiques du clavier et les paramètres de réseau.

Pare feu
: Programme de protection de l'ordinateur, empêchant les intrusions en provenance du réseau ou d'Internet.

Partition mémoire
: Unité logique (abstraction) établie en vue d'isoler, de gérer et de surveiller l'utilisation de la mémoire physique par un processus ou un groupe de processus, principalement au sein d'environnements complexes tels que machines virtualisées ou centres de données.

Permission
: Autorisation de réaliser une opération sur un objet spécifique, par exemple fichier ou imprimante, sur un ordinateur.

Parité
: Caractère pair ou impair d'une valeur numérique et permettant la détection d'erreurs.

Partition
: Zone discrète définie au sein d'un support de stockage.

Partitionnement
: Processus de création d'une structure logique au sein d'un support de stockage (disque dur, SSD, etc.). Une telle opération consiste généralement en la création d'un ou de plusieurs volumes.

Permutation (swapping)
: Technique qui consiste à remiser sur disque des parties de la mémoire momentanément inutilisées. Les données concernées seront rappelées à la demande, si nécessaire.

Pilote de périphérique
: Module mode noyau chargeable qui fait l'interface entre le système d'E/S et le matériel concerné (ou éventuellement un autre pilote). Les pilotes de périphériques sous Windows ne manipulent pas directement les périphériques matériels, mais s'en remettent pour ce faire à la couche d'abstraction prévue à cet effet.

PICO
: Interface spéciale qui assure l'exécution de processus non-Windows natifs dans un environnement Windows.

Pilote
: Logiciel de contrôle d'un composant du système, lequel peut être de nature physique ou logique.

Pilote d'amorçage
: Pilote dont la présence est indispensable pour amorcer le système.

Pilote WDM
: Pilote de périphérique qui se constitue et se définit par son adhésion au modèle de pilote Windows (WDM, Windows Driver Model). Il existe trois types de pilotes WDM : pilotes de fonction, pilotes de bus et pilotes de filtre.

Pilote en mode noyau
: Pilote qui s'exécute dans l'espace mémoire du noyau.

Pilote en mode utilisateur
: Pilote qui s'exécute dans l'espace mémoire utilisateur. Les pilotes sous ce mode font l'objet des mêmes restrictions que les applications.

Pipeline
: Dans un processeur, séquence d'unités exécutant différentes étapes du traitement d'une instruction.

Plan de cache partagé
: Structure qui décrit l'état d'un fichier mis en cache.

Plan de cache partagé
: Structure qui contient l'emplacement des dernières lectures se rapportant à un fichier, informations que le gestionnaire de cache emploie pour effectuer de la lecture anticipée.

Plug and Play
: Fonction de Windows permettant la reconnaissance et l'installation automatique des périphériques connectés à l'ordinateur.

Pool non paginé
: Région de mémoire virtuelle de l'espace système composée de plages d'adresses virtuelles qui résident en permanence en mémoire physique, auxquelles il est en l'occurence possible d'accéder sans qu'il émane de défaut de page. Voir aussi *pool paginé*.

Pool paginé
: Région de mémoire virtuelle de l'espace système composée de plages d'adresses virtuelles dont tout ou partie est susceptible d'être remisée vers le fichier de pagination, puis rechargée en mémoire physique en fonction des circonstances. Voir aussi *pool non paginé*.

Pool spécial
: Région de mémoire noyau dont le Vérificateur de pilote supervise l'usage en vue de détecter d'éventuels débordements de tampon.

Port d'achèvement d'E/S (completion port)
: Mécanisme permettant de délivrer des notifications de fin d'E/S aux threads d'une application.

Portabilité
: Aptitude d'un programme à être utilisé sur des systèmes informatiques de types différents et/ou ayant des conditions d'utilisation différentes.

Processus
: Espace d'adressage virtuel et données de contrôle nécessaires pour l'exécution d'un ensemble de threads. De manière plus large, instance d'un programme en cours d'exécution sur l'ordinateur. Voir aussi programme.

Processus inactif du système
: Processus spécial qui contient un thread par processeur pour comptabiliser le temps d'inactivité de la machine.

Processus d'ouverture de session
: Processus mode utilisateur, exécutant Winlogon.exe, chargé de capturer le nom et le mot de passe de l'utilisateur, de les relayer (pour vérification) à l'autorité de sécurité locale (LSA) et de créer le processus initial de la session utilisateur.,

Processus de support système
: Processus utilisateur, tels le processus d'ouverture de session (Winlogon.exe) et le gestionnaire de session (Smss.exe), qui ne sont pas des services Windows - et demeurent par conséquent hors de portée du controleur de services.

Programme
: Suite statique d'instructions. Voir aussi processus.

Presse-papiers
: Espace en mémoire utilisé afin de stocker provisoirement des données de toutes sortes, incluant images, textes, etc.

Préemption
: Capacité d'un système d'exploitation à exécuter ou stopper une tâche au profit d'une autre.

Privilège
: Type de droit utilisateur qui accorde des autorisations pour l'exécution de tâches clés du système d'exploitation, telles que démarrer une application sous le contrôle d'un débogueur ou mettre l'ordinateur hors tension.

Processeur idéal
: Processeur privilégié pour l'exécution d'un thread.

Programme
: Suite statistique d'instructions.

Quantum
: Intervalle de temps pendant lequel un thread est autorisé à occuper les ressources machine (processeur, mémoire, etc.) avant que Windows ne l'interrompe pour vérifier s'il existe un autre thread en attente d'exécution. Voir aussi unité de quantum.

RAM (Random Access Memory)
: Mémoire à accès aléatoire. Mémoire permettant au processeur d'accéder à n'importe quelle donnée en connaissant son adresse. Voir DRAM et SRAM.

RPC (Remote Procedure Call)
: Système de communication normalisé entre processus client et processus serveur sur un réseau.

RDC (Remote Differential Compression)
: Protocole client serveur de synchronisation utilisé en vue de mettre à jour des fichiers sur les réseaux longue distance à bande passante limitée en répliquant uniquement les modifications nécessaires.

ReadyBoost
: Fonctionnalité de mise en cache visant à réduire les temps d'accès qu'impliquent certaines opérations de lecture, cela en utilisant toute forme de mémoire flash (clé USB, carte SD, etc.) en guise de cache dédié.

Référentiel
: Ensemble structuré d'informations constituant un cadre commun à plusieurs applications.

Redémarrage
: Action par laquelle l'ordinateur est éteint temporairement pour être aussitôt démarré à nouveau.

Registre
: Unité de mémoire intégrée au processeur. Les registres sont utilisés comme source ou destination pour la plupart des opérations effectuées par un processeur.

Répertoire racine
: Répertoire dont découle tous les autres au sein d'un système de fichiers. Partant du répertoire racine, il est possible d'exprimer sans équivoque la position de n'importe quel fichier dans l'arborescence des répertoires, en nommant successivement tous les dossiers intermédiaires imbriqués. Exemple : \Windows \system32\notepad.exe, où le dossier Windows est un sous-ensemble du répertoire racine, le dossier system32 un sous-dossier du dossier Windows et le fichier notepad.exe un fichier stocké dans le dossier system32.

Routine de support pilote
: Routine qu'un pilote appelle de sorte à effectuer ses requêtes d'E/S.

Ruche de secours
: Duplicata de la ruche SYSTEM, stocké dans \Winnt \System32\Config sous le nom System.alt.

SACL (System access control list)
: Contrôle la génération de messages d'audit pour les tentatives d'accès à un objet sécurisable.

SID (*Security Identifier*)
: Moyen permettant d'identifier sans ambiguïté les entités (ressources, utilisateurs, groupes, etc.) d'un système.

SMB (Server Message Block)
: Protocole réseau de niveau application qui permet un accès partagé à des fichiers, imprimantes, ports série et communications diverses entre les nœuds d'un réseau.

SRAM (Static RAM)
: Un des deux principaux types de mémoire. Dans une DRAM, l'information est mémorisée comme la présence ou l'absence de charge dans un minuscule condensateur. Les mémoires DRAM sont plus lentes que les SRAM mais ont une plus grande capacité.

Schéma de partitionnement
: Méthode d'organisation des volumes sur un disque. Il existe deux schémas de partitionnement : enregistrement d'amorçage principal (MBR) et table de partition GUID (GPT), l'un comme l'autre régissant le nombre de volumes

qu'un disque peut posséder et la taille maximale de chaque volume.

Secteur
: Unité minimale employée dans le cadre du partitionnement d'un volume de stockage. Un secteur d'un disque dur de PC compatible IBM comporte généralement 512 octets. La taille du secteur est définie selon le périphérique (disque dur, CD-ROM, etc).

Section critique
: Partie d'un processus où plusieurs threads se chevauchent et qui contient au moins une ressource partagée à laquelle les différents threads peuvent accéder. Un seul thread est autorisé à la section critique à un moment donné. Dans la perspective Windows, primitive de synchronisation employée de sorte à coordonner les accès émanant de multiples threads à une ressource.

Sémaphore
: Objet de synchronisation protégeant l'accès à une ressource matérielle ou logicielle en ne permettant qu'à un nombre fixe de threads concurrents d'y accéder.

Service
: Logiciel généralement chargé lors du démarrage du système qui n'est pas lié à un utilisateur interactif. Les services, analogues des démons UNIX, implémentent souvent le second sous-ensemble d'un mode de transaction client/serveur.

Seuil de pages modifiées
: Nombre de pages que le cache système garde en mémoire avant de solliciter la fonction d'écrivain paresseux (lazy writer) du gestionnaire de cache en vue qu'elle remise (écrive) les pages sur le disque.

Shell
: Interpréteur de commandes sous les système Unix et dérivés.

Socket
: Dispositif de communication bidirectionnel utilisable pour communiquer avec un autre processus sur la même machine ou avec un processus s'exécutant sur d'autres machines.

Stratégie de groupe
: Ensemble de règles et de stratégies qui contrôlent l'environnement de travail des utilisateurs et des ordinateurs dans un environnement Active Directory.

Sous-systèmes d'environnement
: Processus en charge de restituer auprès des applications utilisateur un environnement d'exécution dédié, incluant une ligne de conduite visuelle, une combinaison de caractéristiques programmatiques, et, in fine, une version spécialisée des services fondamentaux du système. Exemples de sous-système : Windows (interface graphique et interface caractère), Posix, OS/2, XBOX, et d'autres.

Synchronisation
: Capacité qu'a un thread à adapter son exécution parallèlement à des circonstances externes hors de son contrôle (manque de mémoire, expiration d'un délai) ou dans l'attente d'un autre objet du système d'exploitation (processus, thread, fichier, mutex, sémaphore, etc.).

Kit de développement logiciel (SDK, Software Development Kit)
: Ensemble d'outils (programmes, bibliothèques, fichiers de configuration, etc.) facilitant le développement d'un logiciel pour une plateforme donnée.

Spinlock
: Mécanisme d'exclusion mutuelle multi processeur basé sur l'attente active.

Superscalaire
: Se dit d'un processeur capable d'exécuter plusieurs instructions simultanément, chacune dans un pipeline différent.

Symbole
: Valeur nommée qui désigne une variable ou une fonction, contenues dans les sections d'un fichier objet ou exécutable.

Système d'E/S
: Composant de l'exécutif responsable de l'acheminement des requêtes d'E/S aux pilotes ou périphériques d'E/S.

Système de fichiers
: Structure logique chargée d'inventorier, d'organiser et de localiser les données hébergées sur un support de stockage quelconque, par exemple disques dur, clés USB, CD, etc.

TLB (Translation Lookaside buffer)
: Cache processeur sur lequel s'appuie l'unité de gestion mémoire (MMU) dans le but d'accélérer le processus de traduction des adresses virtuelles-vers-physiques.

TPM (Trusted Platform Module)
: Standard cryptographique sur lequel s'appuie l'implémentation au niveau matériel d'un système informatique dit de confiance.

Table de partition GUID
: L'un des deux schémas de partitionnement d'un disque. Voir schéma de partitionnement.

Table de ventilation des services système (SSDT, System Service Dispatch Table)
: Table sur laquelle Windows s'appuie de sorte à diriger les appels à des services système vers un traitement approprié.

Table des handles
: Table qui renferme des pointeurs vers tous les objets pour lesquels le processus a ouvert un handle.

Table des handles noyau
: Table dont les handles sont accessibles uniquement depuis le mode noyau.

Tas (heap)
: Mémoire allouée à l'exécution dans la mémoire virtuelle d'une application. Les objets volumineux sont généralement alloués à partir du tas, et les objets de petite taille enregistrés sur la pile.

Thread
: Entité exécutable ordonnancée par le dispatcheur du noyau, souvent associée à un processus qui encapsule un espace d'adressage virtuel.

Thread safe
: Propriété se rapportant au fait qu'une séquence d'instructions puisse ou non fonctionner correctement du moment qu'elle est exécutée simultanément par plusieurs threads.

Thread système
: Type de threads exécutés uniquement en mode noyau, la plupart du temps dans le contexte du processus System.

Thread principal
: Constitue l'assise d'éventuels nouveaux threads.

Tolérance aux pannes
: Capacité d'un système à trouver une compensation du moment que survient une défaillance.

Traitement structuré des exceptions (SEH, Structured Exception Handling)
: Mécanisme par lequel Windows permet aux applications de gérer certaines situations de code exceptionnelles, et de la sorte garder le contrôle en cas d'exception.

Trame d'interception (trap frame)
: Structure de données contenant l'état d'exécution d'un thread interrompu suite à l'apparition d'une interruption ou d'une exception, état sauvegardé dans l'optique de reprendre son exécution par la suite. La trame est généralement un sous-ensemble du contexte du thread.

Tube
: Mécanisme de redirection des entrées-sorties permettant de relier la sortie d'un programme à l'entrée d'un autre, de sorte à créer des files de traitement.

UAC (User Account Control)
: Mécanisme de protection des données qui contribue à empêcher les modifications non autorisées sur l'ordinateur.

UNC (Uniform Naming Convention)
: Convention de dénomination spécifiant une syntaxe commune pour décrire l'emplacement d'une ressource réseau telle qu'un répertoire, une imprimante ou un fichier partagé.

UUID (Universally Unique IDentifier)
: Spécification permettant à des systèmes informatisés d'identifier de façon unique un objet quelconque.

Unicode
: Standard d'encodage de caractères représentant la quasi-totalité des systèmes d'écriture de langue (alphabets, idéogrammes, etc.). Le répertoire de caractères Unicode comporte plusieurs formulaires de présentation, notamment UTF-8, UTF-16 et UTF-32. La plupart des interfaces Windows utilisent le formulaire UTF-16.

Unité de quantum
: Valeur représentant la durée pendant laquelle un thread est en droit d'occuper un processeur avant que son quantum expire. Cette valeur est un entier, et non pas une durée. Voir aussi quantum.

Verrou
: Primitive logicielle de synchronisation qui garantit l'exclusivité au contexte d'exécution qui le détient à un instant donné.

Violation d'accès
: Défaut d'accès à la mémoire qui se produit généralement suite à la tentative d'un thread de lire ou d'écrire dans une région qui n'a pas été

allouée, ou auquel il n'a pas accès. Les erreurs de ce type indiquent presque toujours des erreurs de programmation graves.

Windows Installer
: Service chargé de l'installation, de la mise à jour et de la suppression de logiciel propre aux systèmes Windows. Les fonctionnalités offertes par Windows Installer s'articulent autour de deux axes qui se complètent : un service d'installation coté client (Msiexec.exe) et un fichier de package (fichier .msi ou .msp).

Working set
: Sous ensemble de pages virtuelles résidant en mémoire physique. La zone ainsi formée est accessible sans générer de défaut de page.

Working Set Manager
: Sous-ensemble du gestionnaire de mémoire en charge des stratégies générales de gestion de gestion mémoire, y compris le vieillissement (aging) et l'écriture des pages modifiées.

Winsock
: API pour le fonctionnement des applications Internet dans un environnement Windows.

Windows Installer
: Service d'installation et configuration de logiciels intégré à Windows. Les fichiers d'installation pour Windows Installer portent l'extension .msi (pour une application complète) ou .msp (modification postérieure d'une application installée par un fichier .msi).

X509
: Format utilisé pour les certificats cryptographique.

x86
: Regroupe les microprocesseurs compatibles avec le jeu d'instructions de l'Intel 8086, incluant de la sorte indifféremment des processeurs 16 bits, 32 bits et 64 bits.