Dans les coulisses de Microsoft Windows

Arnaud Maillard

Dans les coulisses de Microsoft Windows

Arnaud Maillard

La dernière date à laquelle ce document a été approuvé est **novembre 2016**. Il a à cette occasion été distribué sous le numéro de version **0.0.7.0** et le nom de code **Ariandel**. 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 condi-tions décrites par la licence

Table des matières

Introduction ..... **x**
À qui s'adresse ce livre ? x
À propos de la licence x
Assistance x
Précisions sur le glossaire xi
En ce qui concerne la lecture xi
1. Concepts et outils **1**
Systèmes d'exploitation 1
Ce qu'est un noyau 1
Ce que propose un système d'exploitation 1
Fonctions d'un système d'exploitation 2
Concepts et terminologie 2
Versions de Windows 2
Examen de la version courante du système d'exploitation 2
Fonctions, services et routines 4
Processus, threads et autres unités fonctionnelles 5
Gestionnaire des tâches 6
Registre 7
Notation hongroise 8
Généralités 8
Nommage des identifiants 9
Histoire 9
Notation hongroise *Apps* 10
Notation hongroise *Systems* 10
Raison d'être et avantages 10
Inconvénients 11
MSDN (*Microsoft Developer Network*) 12
Sites web 12
Bibliothèque MSDN 12
Forums 13
Blogs 13
Base de connaissances 13
Magazine 14
Traduction automatique 14
Anneaux de protection 15
Visualisation des temps utilisateur et noyau 15
Windows-1252 et Unicode 16
UNC (*Uniform Naming Convention*) 17
Système de fichiers 17
Services 18
Espace de noms 19
API Windows 19
Historique 19
Composants de l'API Windows 20
Autres implémentations 20
API Native 21
API noyau 21
Objets et handles 22
Au coeur de Windows 23
Build libre et build controlé 23
Vérification de la version (build) exécutée 24
2. Architecture du système **26**

Architecture générale de Microsoft Windows 26
Fichiers système fondamentaux 26
Objectifs de conception 27
Fiabilité et robustesse 27
Haute performance 28
Internationalisation 30
Extensibilité 31
Compatibilité 32
Portabilité 33
Sécurité 34
Sous-systèmes d'environnement 35
Bibliothèques de sous-système 36
Informations de démarrage de sous-système 36
Environnement MS-DOS (NTVDM) 37
Composants fondamentaux du système 37
Exécutif 37
Gestionnaire de processus 37
Gestionnaire d'objets 38
Gestionnaire de mémoire virtuelle 38
Processus système 38
Processus inactif du système 39
3. Mécanismes système 40
Windows on Windows 40
Wow64 40
Flags globaux de Windows 41
Logiciel *Global Flags* 41
Flags globaux d'image 43
Commande d'extension *!gflag* 44
Options d'exécution d'image 45
Performances 46
Moniteur de fiabilité 46
Accéder au moniteur de fiabilité 47
Index de stabilité système 47
Windows System Assessment Tool (WinSAT) 47
Emplacements du registre 48
WEI (*Windows Experience Index*) 48
Emplacements de registre pour WinSAT 49
Analyseur de performances 49
Droits utilisateurs de l'analyseur de performances 49
Composants de l'architecture Performances 50
PLA (*Performance Logs and Alerts*) 50
PDH (*Performance Data Helper*) 50
Fonctions PHD 51
Codes d'erreur 51
4. Mécanismes de gestion 54
Registre 54
Structure du Registre 54
Clés racines 54
HKEY_PERFORMANCE_DATA 54
Services 54
Objectifs de conception des services Windows 54
Services Windows et tâches planifiées 54
Comptes de services 55
Compte service réseau 55
Tâches planifiées 55

SchTasks 55
Protection et restauration du système 56
Créer un point de restauration 56
Restaurer les fichiers et paramètres système 56
5. Gestionnaire d'objets 58
Espace de noms du gestionnaire d'objets 58
Objets de base nommés 58
WinObj 58
6. Synchronisation 60
Exclusion mutuelle 60
Synchronisation au niveau IRQL bas 60
Sémaphores 60
À propos générique concernant les sémaphores 61
Structure de données 61
Interfaces Windows de sémaphore 62
Compteur et état de signal de sémaphores 62
Partage d'objets sémaphore 62
Limite 63
Synchronisation sur le plan utilisateur 63
Sections critiques 63
Généralités 63
Utilisation 64
Extension de commande !cs 64
Structures de données et mécanismes internes 65
Liste des sections critiques de processus 67
Sections critiques et événements à clé 67
Limites 68
Sections critiques et systèmes multi processeurs 68
Variables conditionnelles 69
Verrous lecture-écriture 69
Mutex 70
Mutex noyau 71
7. Processus et threads 72
Dans les coulisses des processus 72
Structure de données 72
Objet processus de l'exécutif (EPROCESS) 72
Visualisation du format de la structure EPROCESS 72
Objet processus du noyau (KPROCESS) 75
Bloc d'environnement de processus (PEB) 76
Visualisation des données du bloc PEB 76
Examen du bloc PEB 77
Attributs du bloc PEB 79
Interfaces 79
Variables noyau 80
Flux de CreateProcess 80
Hiérarchie de processus 81
Processus inactif du système 82
Processus protégés 83
Identifiants de processus 84
Dans les coulisses des threads 85
Fonctions 85
Structure de données 86
Bloc thread de l'exécutif (ETHREAD) 86
Bloc thread du noyau (KTHREAD) 87
Bloc d'environnement de thread (TEB) 90

Ordonnancement des threads 92
État d'un thread 92
Fonctions Windows d'ordonnancement 92
DLL (*Dynamic Link Library*) 93
Fonctions 93
Communication inter processus 93
Canaux de transmission 93
Tubes anonymes et tubes nommés 93
Opérations sur des tubes nommés 94
Fibres 94
Objets job 94
8. Gestion de la mémoire 96
Gestion du tas 96
Sur la notion de tas 96
Gestionnaire de tas 96
Structure du gestionnaire de tas 96
Fonctions Windows de tas 97
Synchronisation de tas 97
Verrouillage du tas 97
ASLR (*Address Space Layout Randomization*) 98
Copy-on-write 99
Réservation et engagement de pages 100
Informations concernant l'espace d'adressage 101
Protection de la mémoire 101
DEP (*Data Execution Prevention*) 101
DEP de niveau logiciel 102
Verrouillage de la mémoire 102
9. Portable Executable 103
Historique 103
Vue d'ensemble du format 103
Structure d'un fichier PE 103
Entête MS-DOS 103
Micro-programme mode réel 104
Entête PE 104
Entête optionnelle 106
Entête de sections 111
10. Sécurité 114
Composants du système de sécurité 114
Contrôle d'accès 114
SID (*Security Identifier*) 114
Niveaux d'intégrité 117
LSASS (*Local Security Authority Subsystem Service*) 118
Stratégies de restriction logicielle 119
Règles de stratégies et niveaux de sécurité 119
Exceptions 120
Conflits de règles 120
Paramètres de stratégies 120
Configurer les stratégies de restriction logicielle 121
Windows Defender 121
Paramètrer Windows Defender 122
Compatibilité avec ELAM 122
Analyser la station de travail avec Windows Defender 122
Service WinDefend 123
Automatisation de Windows Defender via PowerShell 123
ELAM (Early Launch Anti-Malware) 124

Désactiver ELAM 124
Situation d'ELAM dans la séquence de démarrage 124
Fonctionnement interne de ELAM 125
Microsoft Malware Protection Engine 125
SmartScreen 125
Régler les paramètres Windows SmartScreen 125
Authenticode 126
Protection par Authenticode 126
Autorités de confiance 127
Anatomie d'une signature numérique Authenticode 127
Limitations d'Authenticode 128
Kernel Patch Protection (KPP) 128
Code Integrity 129
Emprunt d'identité 129
BitLocker 129
A. Interfaces 130
Glossaire 136

Liste des illustrations

3.1. Boîte de dialogue Global Flags 41
3.2. Configuration des flags globaux d'image avec Gflags 42

Liste des tableaux

1.1. Systèmes de fichiers de quelques systèmes d'exploitation 18
1.2. Préfixes pour interfaces en mode noyau 22
1.3. Utilitaires de visualisation des coulisses de Windows 23
4.1. Privilèges accordés à Service Réseau 55
6.1. API Windows de sémaphore 62

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. A l'issue de la lecture de cet ouvrage, vous devriez avoir acquis une bonne connaissance des points clés qui gouvernent (architecture générale et composants fondamentaux), des mécanismes centraux qui contrôlent (dispositifs et technologies), des structures de données et des algorithmes qui régissent la structure interne de ce système d'exploitation. 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 s'adresse à un public cherchant à découvrir de fond en comble et de l'intérieur Microsoft Windows, cela dans le but et de manière à acquérir une vue (complète ou parcellaire) sur quelles technologies sont en place et quels dispositifs sont en oeuvre au sein de ces systèmes, ainsi que quels rouages (souterrains ou non) les orchestrent. Les profils concernés incluent 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.

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.

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

Afin d'aider le lecteur à mieux se repérer parmi les moults informations procurées dans ce livre, chaque chapitre a été conçu de sorte à pouvoir être consulté indépendamment. Dans le même esprit, chaque section/segment tente, dans la mesure du possible, d'être en situation de suffisance vis à vis des autres.

Chapitre 1. 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. A 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 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 à 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 important 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).

Ce que propose un système d'exploitation

Ce que propose un système d'exploitation

Un système d'exploitation (OS, *Operating System*) est une collection de programmes qui joue 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.

Concepts et terminologie

Versions de Windows

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 le 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.

■ **BuildLab** Numéro de version complet de Windows

■ **CurrentBuildNumber** Numéro de version de Windows

■ **DigitalProductId** Numéro d'identification de Windows

■ **ProductName** Nom du système d'exploitation tel qu'il a été distribué de façon commerciale

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 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.

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.

Les composants fondamentaux d'un thread incluent ceux que voici.

- Un compteur d'instructions, qui pointe vers l'instruction en cours d'exécution
- Deux piles, une pour l'exécution en mode noyau et une pour l'exécution en mode utilisateur
- Un ensemble de valeurs pour les registres, définissant une partie de l'état du processeur exécutant le thread
- Une zone de stockage privée, appelée TLS (Thread-Local Storage), qui forme un type de mémoire spécifique et locale à un thread

Tous ces éléments sont rassemblés sous le nom de *contexte d'exécution* du thread. L'espace d'adressage, et par conséquent toutes les ressources qui y sont stockées, est commun à tous les threads d'un même processus.

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 Gestionnaire des tâches affiche un résumé des applications et des processus exécutés sur le système. 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.

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.

Fichiers INI

Les systèmes de type DOS utilisaient pour l'enregistrement des paramètres globaux et locaux des fichiers textes linéaires. Avec la sortie de la première version de Windows en 1985, Microsoft introduisit un format de données spécifique, dont les noms des fichiers le respectant portent l'extension .ini.

Les fichiers ini sont des fichiers textes ; ils peuvent en conséquence être manipulés depuis n'importe quel logiciel permettant l'édition de données de cette nature. Ces fichiers sont divisés en sections, qui chacune comporte un certain nombre de paramètres de configuration et de valeurs. Partant de cette nomenclature, un fichier ini ressemble à ce qui suit :

```
[section 1]
Parameter=Value
```

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 à cet usage, le Registre.

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 marquantes, sinon la plus importante, de la notation hongroise est d'incorporer de l'information à même les identifiants logiciels (noms des éléments du programme). En pratique, cette approche s'exprime essentiellement par le biais de divers préfixes, dont l'ensemble se veut être une réponse au besoin d'un niveau de compréhension plus détaillé en ce qui les concerne les données en jeu et les procédés techniques mis en oeuvre.

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]] ## 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.

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.

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

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 appelée *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 1.1. 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 1.2. Préfixes pour interfaces en mode noyau

Préfixe	Composant
Alpc	Appel de procédure asynchrone
Cc	Gestionnaire de cache
Cm	Gestionnaire de configuration
Ex	Routines de l'exécutif
FsRtl	Bibliothèque d'exécution du pilote de système de fichiers
Hal	Couche d'abstraction matérielle
Io	Gestionnaire d'E/S
Ke	Noyau
Ldr	Chargeur de modules
Lpc	Appel de procédure locale
Mm	Gestionnaire mémoire
Nt	Services système
Ob	Gestionnaire d'objets
Po	Gestionnaire d'alimentation
Pp	Gestionnaire PnP
Ps	Gestionnaire de processus
Rtl	Bibliothèque d'exécution
Se	Sécurité
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.

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.

Un *handle* est une valeur opaque qui fait office de référence à une instance d'un objet. Une large majorité des fonctions de l'API Windows retournent ce type de données, à vrai dire le seul rendu visible aux programmes en mode utilisateur.

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 1.3. 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).

Chapitre 2. 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 actuels, 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.

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.

Note

Sur le plan de la langue, 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 nombre de procédés utilisés 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).

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.

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.

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.

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.

Composants fondamentaux du système

Exécutif

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*.

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.

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.

Gestionnaire de mémoire virtuelle

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.

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.

Processus système

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.

Chapitre 3. 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 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 traité 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.

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.

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.

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
```

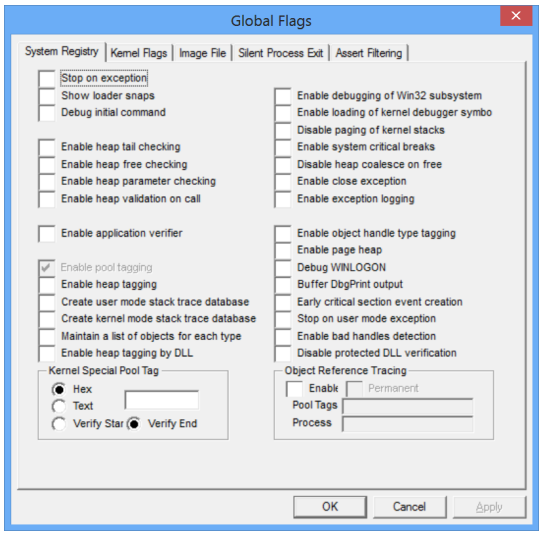
La représentation binaire (et à l'échelle d'une seule clé de registre) étant quelquefois insuffisante pour englober toutes les informations relatives à un statut, une condition ou à un chemin d'exécution alternatif, certains composants de Windows ont en partie pour tâche de gérer des protocoles additionnels 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 Figure 3.1, « Boîte de dialogue Global Flags », l'interface principale de Global Flags est divisée en onglets regroupant plusieurs options d'une même catégorie.

Figure 3.1. Boîte de dialogue Global Flags



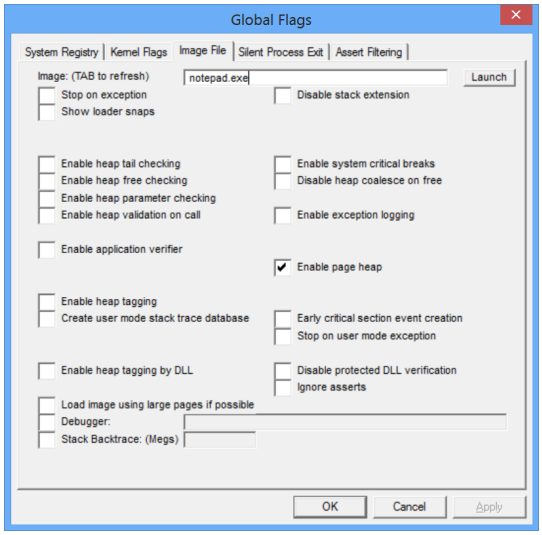
Vous pouvez basculer entre paramètres du registre (en cliquant sur System Registry) et valeur courante de la variable en mémoire système (en cliquant sur Kernel Flags). S'il est possible dans quelques cas de remanier la disposition générale des flags d'un système en cours d'exécution, la plupart des 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.

Figure 3.2. Configuration des flags globaux d'image avec Gflags



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** (*PageHeapFlags*) 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

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 détaillés 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 collectés 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
```

Analyseur de performances

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).

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

Chapitre 4. Mécanismes de gestion

Registre

Structure du Registre

Clés racines

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.

Services

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

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.

Le tableaux Tableau 4.1, « Privilèges accordés à Service Réseau » restitue les privilèges que le compte Service Réseau prend en charge.

Tableau 4.1. Privilèges accordés à Service Réseau

Privilège	Description
SeAssignPrimaryTokenPrivilege	Remplacer un jeton de niveau processus
SeIncreaseQuotaPrivilege	Ajuster les quotas de mémoire pour un processus
SeAuditPrivilege	Générer des audits de sécurité
SeChangeNotifyPrivilege	Contourner la vérification de parcours
SeNetworkLogonRight	Accéder à cet ordinateur à partir du réseau
SeInteractiveLogonRight	Ouvrir une session en tant que service
SeImpersonatePrivilege	Emprunter l'identité d'un client après l'authentification

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.

Chapitre 5. Gestionnaire d'objets

Espace de noms du gestionnaire d'objets

La liste qui suit énumère quelques uns des volumes (on parle en l'occurence de répertoire d'objets standard) définis dans l'espace de noms du gestionnaire d'objets.

- **\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 \KnownDLLs** 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.

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).

Chapitre 6. 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.

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.

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 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 6.1. API Windows de sémaphore

Fonction	Utilité
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 sémaphore.
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.

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.

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.) Comparables sur le plan conceptuel à un axe à une seule voie, les sections critiques permettent de limiter l'exécution simultanée d'un intervalle défini d'instructions exécutables à un thread à un instant donné dans une seule application.

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 liste qui suit énumère les principales fonctions de l'API Windows avec lesquelles les applications interagissent dès lors qu'elles emploient des sections critiques.

■ ***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. Pour commencer, 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.

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 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. 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.

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.

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 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.

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

La méthode la plus simple pour gérer l'accès concurrent à des ressources partagées est de recourir à des verrous. Le verrou le plus élémentaire sur le plan conceptuel est le mutex, abréviation de *mutual exclusion*, qui représente un droit d'accès exclusif à une ressource, et facilite de la sorte la synchronisation entre processus.

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.

Dans la perspective Windows, un mutex est un objet qui se définit essentiellement selon le fait d'avoir ou non 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.

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
```

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.

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*** Gére 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.

Chapitre 7. 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

Structure de données

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> dt nt!_EPROCESS fffffa80453ec940
   +0x000 Pcb              : _KPROCESS
   +0x2c8 ProcessLock      : _EX_PUSH_LOCK
   +0x2d0 CreateTime       : _LARGE_INTEGER 0x01d17891`3c96f1ed
   +0x2d8 RundownProtect   : _EX_RUNDOWN_REF
   +0x2e0 UniqueProcessId  : 0x00000000`00000c84 Void
   +0x2e8 ActiveProcessLinks : _LIST_ENTRY [ 0xfffffa80`465cac28 - 0xfffffa80`47183c28 ]
   +0x2f8 Flags2           : 0x200d000
   +0x2f8 JobNotReallyActive : 0y0
   +0x2f8 AccountingFolded : 0y0
   +0x2f8 NewProcessReported : 0y0
   +0x2f8 ExitProcessReported : 0y0
   +0x2f8 ReportCommitChanges : 0y0
   +0x2f8 LastReportMemory : 0y0
   +0x2f8 NoWakeCharge     : 0y0
   +0x2f8 HandleTableRundown : 0y0
   +0x2f8 NeedsHandleRundown : 0y0
   +0x2f8 RefTraceEnabled  : 0y0
   +0x2f8 NumaAware        : 0y0
   +0x2f8 EmptyJobEvaluated : 0y0
   +0x2f8 DefaultPagePriority : 0y101
   +0x2f8 PrimaryTokenFrozen : 0y1
   +0x2f8 ProcessVerifierTarget : 0y0
   +0x2f8 StackRandomizationDisabled : 0y0
   +0x2f8 AffinityPermanent : 0y0
   +0x2f8 AffinityUpdateEnable : 0y0
   +0x2f8 PropagateNode    : 0y0
   +0x2f8 ExplicitAffinity : 0y0
   +0x2f8 ProcessExecutionState : 0y00
   +0x2f8 DisallowStrippedImages : 0y0
   +0x2f8 HighEntropyASLREnabled : 0y1
   +0x2f8 ExtensionPointDisable : 0y0
   +0x2f8 ForceRelocateImages : 0y0
   +0x2f8 ProcessStateChangeRequest : 0y00
   +0x2f8 ProcessStateChangeInProgress : 0y0
   +0x2f8 DisallowWin32kSystemCalls : 0y0
   +0x2fc Flags            : 0x144d0c01
   +0x2fc CreateReported   : 0y1
   +0x2fc NoDebugInherit   : 0y0
   +0x2fc ProcessExiting   : 0y0
   +0x2fc ProcessDelete    : 0y0
   +0x2fc Wow64SplitPages  : 0y0
   +0x2fc VmDeleted        : 0y0
   +0x2fc OutswapEnabled   : 0y0
   +0x2fc Outswapped       : 0y0
   +0x2fc ForkFailed       : 0y0
   +0x2fc Wow64VaSpace4Gb  : 0y0
   +0x2fc AddressSpaceInitialized : 0y11
   +0x2fc SetTimerResolution : 0y0
   +0x2fc BreakOnTermination : 0y0
   +0x2fc DeprioritizeViews : 0y0
   +0x2fc WriteWatch       : 0y0
   +0x2fc ProcessInSession : 0y1
   +0x2fc OverrideAddressSpace : 0y0
   +0x2fc HasAddressSpace  : 0y1
   +0x2fc LaunchPrefetched : 0y1
   +0x2fc Background       : 0y0
   +0x2fc VmTopDown        : 0y0
   +0x2fc ImageNotifyDone  : 0y1
   +0x2fc PdeUpdateNeeded  : 0y0
   +0x2fc VdmAllowed       : 0y0
   +0x2fc CrossSessionCreate : 0y0
   +0x2fc ProcessInserted  : 0y1
```

```
+0x2fc DefaultIoPriority : 0y010
+0x2fc ProcessSelfDelete : 0y0
+0x2fc SetTimerResolutionLink : 0y0
+0x300 ProcessQuotaUsage : [2] 0x1f80
+0x310 ProcessQuotaPeak : [2] 0x2130
+0x320 PeakVirtualSize  : 0xb562000
+0x328 VirtualSize      : 0xb561000
+0x330 SessionProcessLinks : _LIST_ENTRY [ 0xfffff880`046c3010 - 0xfffffa80`47183c70 ]
+0x340 ExceptionPortData : 0xfffffa80`463d5e40 Void
+0x340 ExceptionPortValue : 0xfffffa80`463d5e40
+0x340 ExceptionPortState : 0y000
+0x348 Token            : _EX_FAST_REF
+0x350 WorkingSetPage   : 0x156c40
+0x358 AddressCreationLock : _EX_PUSH_LOCK
+0x360 RotateInProgress : (null)
+0x368 ForkInProgress   : (null)
+0x370 HardwareTrigger  : 0
+0x378 CommitChargeJob  : (null)
+0x380 CloneRoot        : (null)
+0x388 NumberOfPrivatePages : 0x600
+0x390 NumberOfLockedPages : 0
+0x398 Win32Process     : 0xfffff901`01f59ca0 Void
+0x3a0 Job              : (null)
+0x3a8 SectionObject    : 0xfffff8a0`07a34120 Void
+0x3b0 SectionBaseAddress : 0x000007f6`98f00000 Void
+0x3b8 Cookie           : 0xd3a555c5
+0x3c0 WorkingSetWatch  : (null)
+0x3c8 Win32WindowStation : 0x00000000`00000038 Void
+0x3d0 InheritedFromUniqueProcessId : 0x00000000`00000708 Void
+0x3d8 LdtInformation   : (null)
+0x3e0 CreatorProcess   : 0x00000000`00000001 _EPROCESS
+0x3e0 ConsoleHostProcess : 1
+0x3e8 Peb              : 0x000007f6`98e0f000 _PEB
+0x3f0 Session          : 0xfffff880`046c3000 Void
+0x3f8 AweInfo          : (null)
+0x400 QuotaBlock       : 0xfffffa80`46017640 _EPROCESS_QUOTA_BLOCK
+0x408 ObjectTable      : 0xfffff8a0`07a59b40 _HANDLE_TABLE
+0x410 DebugPort        : (null)
+0x418 Wow64Process     : (null)
+0x420 DeviceMap        : 0xfffff8a0`01389760 Void
+0x428 EtwDataSource    : (null)
+0x430 PageDirectoryPte : 0
+0x438 ImageFileName    : [15]  "notepad.exe"
+0x447 PriorityClass    : 0x2 ''
+0x448 SecurityPort     : (null)
+0x450 SeAuditProcessCreationInfo : _SE_AUDIT_PROCESS_CREATION_INFO
+0x458 JobLinks         : _LIST_ENTRY [ 0x00000000`00000000 - 0x00000000`00000000 ]
+0x468 HighestUserAddress : 0x000007ff`ffff0000 Void
+0x470 ThreadListHead   : _LIST_ENTRY [ 0xfffffa80`46fbaf00 - 0xfffffa80`46fbaf00 ]
+0x480 ActiveThreads    : 1
+0x484 ImagePathHash    : 0xeb1b961a
+0x488 DefaultHardErrorProcessing : 1
+0x48c LastThreadExitStatus : 0n0
+0x490 PrefetchTrace    : _EX_FAST_REF
+0x498 LockedPagesList  : (null)
+0x4a0 ReadOperationCount : _LARGE_INTEGER 0x0
+0x4a8 WriteOperationCount : _LARGE_INTEGER 0x0
+0x4b0 OtherOperationCount : _LARGE_INTEGER 0x0
+0x4b8 ReadTransferCount : _LARGE_INTEGER 0x0
+0x4c0 WriteTransferCount : _LARGE_INTEGER 0x0
+0x4c8 OtherTransferCount : _LARGE_INTEGER 0x0
+0x4d0 CommitChargeLimit : 0
+0x4d8 CommitCharge     : 0x687
+0x4e0 CommitChargePeak : 0x687
+0x4e8 Vm               : _MMSUPPORT
+0x578 MmProcessLinks   : _LIST_ENTRY [ 0xfffffa80`465caeb8 - 0xfffffa80`47183eb8 ]
+0x588 ModifiedPageCount : 0x39b
+0x58c ExitStatus       : 0n259
```

```
   +0x590 VadRoot          : _MM_AVL_TABLE
   +0x5c0 VadPhysicalPages : 0
   +0x5c8 VadPhysicalPagesLimit : 0
   +0x5d0 AlpcContext      : _ALPC_PROCESS_CONTEXT
   +0x5f0 TimerResolutionLink : _LIST_ENTRY [ 0x00000000`00000000 - 0x00000000`00000000 ]
   +0x600 TimerResolutionStackRecord : (null)
   +0x608 RequestedTimerResolution : 0
   +0x60c SmallestTimerResolution : 0
   +0x610 ExitTime         : _LARGE_INTEGER 0x0
   +0x618 InvertedFunctionTable : (null)
   +0x620 InvertedFunctionTableLock : _EX_PUSH_LOCK
   +0x628 ActiveThreadsHighWatermark : 1
   +0x62c LargePrivateVadCount : 0
   +0x630 ThreadListLock   : _EX_PUSH_LOCK
   +0x638 WnfContext       : (null)
   +0x640 SectionMappingSize : 0x40000
   +0x648 SignatureLevel   : 0 ''
   +0x649 SectionSignatureLevel : 0 ''
   +0x64a SpareByte20      : [2]  ""
   +0x64c KeepAliveCounter : 0
   +0x650 DiskCounters     : 0xfffffa80`453ecfa0 _PROCESS_DISK_COUNTERS
   +0x658 LastFreezeInterruptTime : 0
```

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
```

Bloc d'environnement de processus (PEB)

Visualisation des données du bloc PEB

Pour voir concrètement quels attributs sont enregistrés à l'échelle de structures PEB, utilisez la commande dt des débogueurs Windows standards.

```
lkd> dt nt!_PEB
   +0x000 InheritedAddressSpace : UChar
   +0x001 ReadImageFileExecOptions : UChar
   +0x002 BeingDebugged    : UChar
   +0x003 BitField         : UChar
   +0x003 ImageUsesLargePages : Pos 0, 1 Bit
   +0x003 IsProtectedProcess : Pos 1, 1 Bit
   +0x003 IsImageDynamicallyRelocated : Pos 2, 1 Bit
   +0x003 SkipPatchingUser32Forwarders : Pos 3, 1 Bit
   +0x003 IsPackagedProcess : Pos 4, 1 Bit
   +0x003 IsAppContainer   : Pos 5, 1 Bit
   +0x003 IsProtectedProcessLight : Pos 6, 1 Bit
   +0x003 SpareBits        : Pos 7, 1 Bit
   +0x004 Padding0         : [4] UChar
   +0x008 Mutant           : Ptr64 Void
   +0x010 ImageBaseAddress : Ptr64 Void
   +0x018 Ldr              : Ptr64 _PEB_LDR_DATA
   +0x020 ProcessParameters : Ptr64 _RTL_USER_PROCESS_PARAMETERS
   +0x028 SubSystemData    : Ptr64 Void
   +0x030 ProcessHeap      : Ptr64 Void
   +0x038 FastPebLock      : Ptr64 _RTL_CRITICAL_SECTION
   +0x040 AtlThunkSListPtr : Ptr64 Void
   +0x048 IFEOKey          : Ptr64 Void
   +0x050 CrossProcessFlags : Uint4B
   +0x050 ProcessInJob     : Pos 0, 1 Bit
   +0x050 ProcessInitializing : Pos 1, 1 Bit
   +0x050 ProcessUsingVEH  : Pos 2, 1 Bit
   +0x050 ProcessUsingVCH  : Pos 3, 1 Bit
   +0x050 ProcessUsingFTH  : Pos 4, 1 Bit
   +0x050 ReservedBits0    : Pos 5, 27 Bits
   +0x054 Padding1         : [4] UChar
   +0x058 KernelCallbackTable : Ptr64 Void
   +0x058 UserSharedInfoPtr : Ptr64 Void
   +0x060 SystemReserved   : [1] Uint4B
   +0x064 AtlThunkSListPtr32 : Uint4B
   +0x068 ApiSetMap        : Ptr64 Void
   +0x070 TlsExpansionCounter : Uint4B
   +0x074 Padding2         : [4] UChar
   +0x078 TlsBitmap        : Ptr64 Void
   +0x080 TlsBitmapBits    : [2] Uint4B
   +0x088 ReadOnlySharedMemoryBase : Ptr64 Void
   +0x090 SparePvoid0      : Ptr64 Void
   +0x098 ReadOnlyStaticServerData : Ptr64 Ptr64 Void
   +0x0a0 AnsiCodePageData : Ptr64 Void
   +0x0a8 OemCodePageData  : Ptr64 Void
   +0x0b0 UnicodeCaseTableData : Ptr64 Void
   +0x0b8 NumberOfProcessors : Uint4B
   +0x0bc NtGlobalFlag     : Uint4B
   +0x0c0 CriticalSectionTimeout : _LARGE_INTEGER
   +0x0c8 HeapSegmentReserve : Uint8B
   +0x0d0 HeapSegmentCommit : Uint8B
   +0x0d8 HeapDeCommitTotalFreeThreshold : Uint8B
   +0x0e0 HeapDeCommitFreeBlockThreshold : Uint8B
   +0x0e8 NumberOfHeaps    : Uint4B
```

```
   +0x0ec MaximumNumberOfHeaps : Uint4B
   +0x0f0 ProcessHeaps     : Ptr64 Ptr64 Void
   +0x0f8 GdiSharedHandleTable : Ptr64 Void
   +0x100 ProcessStarterHelper : Ptr64 Void
   +0x108 GdiDCAttributeList : Uint4B
   +0x10c Padding3         : [4] UChar
   +0x110 LoaderLock       : Ptr64 _RTL_CRITICAL_SECTION
   +0x118 OSMajorVersion   : Uint4B
   +0x11c OSMinorVersion   : Uint4B
   +0x120 OSBuildNumber    : Uint2B
   +0x122 OSCSDVersion     : Uint2B
   +0x124 OSPlatformId     : Uint4B
   +0x128 ImageSubsystem   : Uint4B
   +0x12c ImageSubsystemMajorVersion : Uint4B
   +0x130 ImageSubsystemMinorVersion : Uint4B
   +0x134 Padding4         : [4] UChar
   +0x138 ActiveProcessAffinityMask : Uint8B
   +0x140 GdiHandleBuffer  : [60] Uint4B
   +0x230 PostProcessInitRoutine : Ptr64     void
   +0x238 TlsExpansionBitmap : Ptr64 Void
   +0x240 TlsExpansionBitmapBits : [32] Uint4B
   +0x2c0 SessionId        : Uint4B
   +0x2c4 Padding5         : [4] UChar
   +0x2c8 AppCompatFlags   : _ULARGE_INTEGER
   +0x2d0 AppCompatFlagsUser : _ULARGE_INTEGER
   +0x2d8 pShimData        : Ptr64 Void
   +0x2e0 AppCompatInfo    : Ptr64 Void
   +0x2e8 CSDVersion       : _UNICODE_STRING
   +0x2f8 ActivationContextData : Ptr64 _ACTIVATION_CONTEXT_DATA
   +0x300 ProcessAssemblyStorageMap : Ptr64 _ASSEMBLY_STORAGE_MAP
   +0x308 SystemDefaultActivationContextData : Ptr64 _ACTIVATION_CONTEXT_DATA
   +0x310 SystemAssemblyStorageMap : Ptr64 _ASSEMBLY_STORAGE_MAP
   +0x318 MinimumStackCommit : Uint8B
   +0x320 FlsCallback      : Ptr64 _FLS_CALLBACK_INFO
   +0x328 FlsListHead      : _LIST_ENTRY
   +0x338 FlsBitmap        : Ptr64 Void
   +0x340 FlsBitmapBits    : [4] Uint4B
   +0x350 FlsHighIndex     : Uint4B
   +0x358 WerRegistrationData : Ptr64 Void
   +0x360 WerShipAssertPtr : Ptr64 Void
   +0x368 pUnused          : Ptr64 Void
   +0x370 pImageHeaderHash : Ptr64 Void
   +0x378 TracingFlags     : Uint4B
   +0x378 HeapTracingEnabled : Pos 0, 1 Bit
   +0x378 CritSecTracingEnabled : Pos 1, 1 Bit
   +0x378 LibLoaderTracingEnabled : Pos 2, 1 Bit
   +0x378 SpareTracingBits : Pos 3, 29 Bits
   +0x37c Padding6         : [4] UChar
   +0x380 CsrServerReadOnlySharedMemoryBase : Uint8B
   +0x388 TppWorkerpListLock : Uint8B
   +0x390 TppWorkerpList   : _LIST_ENTRY
   +0x3a0 WaitOnAddressHashTable : [128] Ptr64 Void
```

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.

- **Page de code ANSI** *AnsiCodePageData* Page de code utilisée utilisée pour les applications graphiques.
- **Flag de débogage** *BeingDebugged* Indique si le processus a été démarré sous le contrôle d'un débogueur.
- **Délai de section critique** *CriticalSectionTimeout* Limite le temps que peut passer le processus à attendre une section critique
- **Adresse de base de l'image** *ImageBaseAddress* Adresse à laquelle l'image exécutable sous-jacente au processus est chargée en mémoire.
- **Masque d'affinité de processus image** *ImageProcessAffinityMask*
- **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.
- **Liste des modules** *Ldr* Listes des modules chargées dans l'espace d'adressage mode utilisateur du processus.
- **Nombre de tas** *NumberOfHeaps* Nombre de tas que le processus a en sa possession.
- **Page de code OEM** *OemCodePageData* Page de code utilisée dans la console.
- **Version majeure du du système d'exploitation** *OSMajorVersion*
- **Version mineure du du système d'exploitation** *OSMinorVersion*
- **Informations de tas** Tas par défaut (*ProcessHeap*) et tas supplémentaires (*ProcessHeaps*) du processus.
- **Nombre de tas** *MaximumNumberOfHeaps* Nombre maximal de tas que le processus peut réclamer.
- **Masque d'affinité de processus image** ActiveProcessAffinityMask
- **Table des handles GDI partagés** GdiSharedHandleTable.
- **Synchronisation de niveau image** Section critique utilisée par le chargeur d'image lors de la création de processus (*LoaderLock*).
- **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.

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 liste qui suit énumère quelques-unes des variables globales système ayant un rôle prépondérant parmi les stratégies de gestion des processus.

- *PsActiveProcessHead* Tête de liste des processus actifs.

- *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.

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. Au sein du sous-système Windows, préparation de la configuration du processus et du thread nouvellement créés.

6. Démarrage de l'exécution du thread initial.

7. Fin de l'initialisation de l'espace d'adressage (par exemple, chargement des DLL requises) et commencement de l'exécution du programme.

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 débogguer 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.)

■ READ_CONTROL Empêche que la liste de contrôle d'accès (ACL) du processus protégé soit lue.
- PROCESS_ALL_ACCESS Empêche l'accès complet 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 à partir d'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

■ PROCESS_VM_READ Empêche l'accès à la mémoire d'un processus protégé.

■ PROCESS_VM_WRITE

■ 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 (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.

Identifiants de processus

Identifiants 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.

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.

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 fin d'un thread

■ ***Get/SetThreadContext*** Retourne ou modifie les valeurs des registres processeur d'un thread

Structure de données

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)

- **ID client *Cid*** Réunie l'ID du thread et du processus auquel il appartient.

- **Heures de début *CreateTime***

- **Heures de fin *ExitTime***

- **Adresse de départ *StartAddress***

- **Thread système *SystemThread*** Indique si le thread fait partie de ceux exécutés en mode noyau, et considérés à ce titre comme partie intégrante du système d'exploitation.

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
   +0x000 Header           : _DISPATCHER_HEADER
   +0x018 SListFaultAddress : Ptr64 Void
   +0x020 QuantumTarget    : Uint8B
   +0x028 InitialStack     : Ptr64 Void
   +0x030 StackLimit       : Ptr64 Void
   +0x038 StackBase        : Ptr64 Void
   +0x040 ThreadLock       : Uint8B
   +0x048 CycleTime        : Uint8B
   +0x050 CurrentRunTime   : Uint4B
   +0x054 ExpectedRunTime  : Uint4B
   +0x058 KernelStack      : Ptr64 Void
   +0x060 StateSaveArea    : Ptr64 _XSAVE_FORMAT
   +0x068 SchedulingGroup  : Ptr64 _KSCHEDULING_GROUP
   +0x070 WaitRegister     : _KWAIT_STATUS_REGISTER
   +0x071 Running          : UChar
   +0x072 Alerted          : [2] UChar
   +0x074 KernelStackResident : Pos 0, 1 Bit
   +0x074 ReadyTransition  : Pos 1, 1 Bit
   +0x074 ProcessReadyQueue : Pos 2, 1 Bit
   +0x074 WaitNext         : Pos 3, 1 Bit
   +0x074 SystemAffinityActive : Pos 4, 1 Bit
   +0x074 Alertable        : Pos 5, 1 Bit
   +0x074 CodePatchInProgress : Pos 6, 1 Bit
   +0x074 UserStackWalkActive : Pos 7, 1 Bit
   +0x074 ApcInterruptRequest : Pos 8, 1 Bit
   +0x074 QuantumEndMigrate : Pos 9, 1 Bit
   +0x074 UmsDirectedSwitchEnable : Pos 10, 1 Bit
   +0x074 TimerActive      : Pos 11, 1 Bit
   +0x074 SystemThread     : Pos 12, 1 Bit
   +0x074 ProcessDetachActive : Pos 13, 1 Bit
   +0x074 CalloutActive    : Pos 14, 1 Bit
   +0x074 ScbReadyQueue    : Pos 15, 1 Bit
   +0x074 ApcQueueable     : Pos 16, 1 Bit
   +0x074 ReservedStackInUse : Pos 17, 1 Bit
   +0x074 UmsPerformingSyscall : Pos 18, 1 Bit
   +0x074 Reserved         : Pos 19, 13 Bits
   +0x074 MiscFlags        : Int4B
   +0x078 AutoAlignment    : Pos 0, 1 Bit
   +0x078 DisableBoost     : Pos 1, 1 Bit
   +0x078 UserAffinitySet  : Pos 2, 1 Bit
   +0x078 AlertedByThreadId : Pos 3, 1 Bit
   +0x078 QuantumDonation  : Pos 4, 1 Bit
   +0x078 EnableStackSwap  : Pos 5, 1 Bit
   +0x078 GuiThread        : Pos 6, 1 Bit
   +0x078 DisableQuantum   : Pos 7, 1 Bit
   +0x078 ChargeOnlyGroup  : Pos 8, 1 Bit
   +0x078 DeferPreemption  : Pos 9, 1 Bit
   +0x078 QueueDeferPreemption : Pos 10, 1 Bit
   +0x078 ForceDeferSchedule : Pos 11, 1 Bit
   +0x078 ExplicitIdealProcessor : Pos 12, 1 Bit
   +0x078 FreezeCount      : Pos 13, 1 Bit
   +0x078 EtwStackTraceApcInserted : Pos 14, 8 Bits
   +0x078 ReservedFlags    : Pos 22, 10 Bits
   +0x078 ThreadFlags      : Int4B
   +0x07c Spare0           : Uint4B
   +0x080 SystemCallNumber : Uint4B
   +0x084 Spare1           : Uint4B
   +0x088 FirstArgument    : Ptr64 Void
   +0x090 TrapFrame        : Ptr64 _KTRAP_FRAME
   +0x098 ApcState         : _KAPC_STATE
   +0x098 ApcStateFill     : [43] UChar
   +0x0c3 Priority         : Char
   +0x0c4 UserIdealProcessor : Uint4B
   +0x0c8 WaitStatus       : Int8B
   +0x0d0 WaitBlockList    : Ptr64 _KWAIT_BLOCK
   +0x0d8 WaitListEntry    : _LIST_ENTRY
```

```
+0x0d8 SwapListEntry    : _SINGLE_LIST_ENTRY
+0x0e8 Queue            : Ptr64 _KQUEUE
+0x0f0 Teb              : Ptr64 Void
+0x0f8 RelativeTimerBias : Uint8B
+0x100 Timer            : _KTIMER
+0x140 WaitBlock        : [4] _KWAIT_BLOCK
+0x140 WaitBlockFill4   : [20] UChar
+0x154 ContextSwitches  : Uint4B
+0x140 WaitBlockFill5   : [68] UChar
+0x184 State            : UChar
+0x185 NpxState         : Char
+0x186 WaitIrql         : UChar
+0x187 WaitMode         : Char
+0x140 WaitBlockFill6   : [116] UChar
+0x1b4 WaitTime         : Uint4B
+0x140 WaitBlockFill7   : [164] UChar
+0x1e4 KernelApcDisable : Int2B
+0x1e6 SpecialApcDisable : Int2B
+0x1e4 CombinedApcDisable : Uint4B
+0x140 WaitBlockFill8   : [40] UChar
+0x168 ThreadCounters   : Ptr64 _KTHREAD_COUNTERS
+0x140 WaitBlockFill9   : [88] UChar
+0x198 XStateSave       : Ptr64 _XSTATE_SAVE
+0x140 WaitBlockFill10  : [136] UChar
+0x1c8 Win32Thread      : Ptr64 Void
+0x140 WaitBlockFill11  : [176] UChar
+0x1f0 Ucb              : Ptr64 _UMS_CONTROL_BLOCK
+0x1f8 Uch              : Ptr64 _KUMS_CONTEXT_HEADER
+0x200 TebMappedLowVa   : Ptr64 Void
+0x208 QueueListEntry   : _LIST_ENTRY
+0x218 NextProcessor    : Uint4B
+0x21c DeferredProcessor : Uint4B
+0x220 Process          : Ptr64 _KPROCESS
+0x228 UserAffinity     : _GROUP_AFFINITY
+0x228 UserAffinityFill : [10] UChar
+0x232 PreviousMode     : Char
+0x233 BasePriority     : Char
+0x234 PriorityDecrement : Char
+0x234 ForegroundBoost  : Pos 0, 4 Bits
+0x234 UnusualBoost     : Pos 4, 4 Bits
+0x235 Preempted        : UChar
+0x236 AdjustReason     : UChar
+0x237 AdjustIncrement  : Char
+0x238 Affinity         : _GROUP_AFFINITY
+0x238 AffinityFill     : [10] UChar
+0x242 ApcStateIndex    : UChar
+0x243 WaitBlockCount   : UChar
+0x244 IdealProcessor   : Uint4B
+0x248 ApcStatePointer  : [2] Ptr64 _KAPC_STATE
+0x258 SavedApcState    : _KAPC_STATE
+0x258 SavedApcStateFill : [43] UChar
+0x283 WaitReason       : UChar
+0x284 SuspendCount     : Char
+0x285 Saturation       : Char
+0x286 SListFaultCount  : Uint2B
+0x288 SchedulerApc     : _KAPC
+0x288 SchedulerApcFill0 : [1] UChar
+0x289 ResourceIndex    : UChar
+0x288 SchedulerApcFill1 : [3] UChar
+0x28b QuantumReset     : UChar
+0x288 SchedulerApcFill2 : [4] UChar
+0x28c KernelTime       : Uint4B
+0x288 SchedulerApcFill3 : [64] UChar
+0x2c8 WaitPrcb         : Ptr64 _KPRCB
+0x288 SchedulerApcFill4 : [72] UChar
+0x2d0 LegoData         : Ptr64 Void
+0x288 SchedulerApcFill5 : [83] UChar
+0x2db CallbackNestingLevel : UChar
```

```
   +0x2dc UserTime         : Uint4B
   +0x2e0 SuspendEvent     : _KEVENT
   +0x2f8 ThreadListEntry  : _LIST_ENTRY
   +0x308 MutantListHead   : _LIST_ENTRY
   +0x318 ReadOperationCount : Int8B
   +0x320 WriteOperationCount : Int8B
   +0x328 OtherOperationCount : Int8B
   +0x330 ReadTransferCount : Int8B
   +0x338 WriteTransferCount : Int8B
   +0x340 OtherTransferCount : Int8B
```

Bloc d'environnement de thread (TEB)

```
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
```

Ordonnancement des threads

État d'un thread

Le cycle de vie de toute unité de code est au sein du modèle d'ordonnancement de Windows marqué par une succession de phases, divers états d'exécution. Voici les états possibles pour un thread :

- **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.

- **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.

- **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.

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.

DLL (*Dynamic Link Library*)

Fonctions

La liste qui suit énumère les principales fonctions Windows 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.

■ **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**

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

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.

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.

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** Termine tous les processus d'un job.
- **SetInformationJobObject** Définit des limites.
- **QueryInformationJobObject** Donne des informations sur le job.

Chapitre 8. Gestion de la mémoire

Gestion du tas

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.

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.

Fonctions Windows de tas

La bibliothèque principale du sous-système Windows (Kernel32) communique et sollicite les services du gestionnaire de tas par le biais des fonctions que voici :

■ *HeapCreate* ou *HeapDestroy* Crée ou supprime, respectivement, un tas.

■ *HeapAlloc* Alloue un bloc du tas.

■ *HeapFree* Libère Libère un bloc précédemment alloué via *HeapAlloc*.

■ *HeapRealloc* Modifie la taille d'une allocation existante.

■ *HeapLock* ou *HeapLock* Contrôle l'exclusion mutuelle aux opérations de tas.

■ *HeapWalk* Enumère les entrées et régions d'un tas.

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.

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 doté 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.

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 la concernant. 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).

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.

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.

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.

DEP (*Data Execution Prevention*)

La prévention d'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 sa forme la plus aboutie, DEP fonctionne conjointement à la protection « pas d'exécution » définie au niveau du matériel. Si elle ne l'est pas, Windows se replie alors sur une voie entièrement logicielle. Pour plus de détails sur ce dernier point, consultez la section « DEP de niveau logiciel ».

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 pages verrouillées n'étant pas sous l'influence du schéma classique qui les auraient un moment donné peut-être conduits jusqu'au fichier de pagination (forçant le système à éventuellement les recharger depuis ce support), l'accès à une page verrouillée en mémoire physique ne suscite aucun défaut de page. Les pages qu'un processus a verrouillées le restent jusqu'à ce que le processus les déverrouille ou se termine.

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.

Chapitre 9. 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 mentionnés 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.

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. La valeur à laquelle cette variable est affectée correspond à la chaine ASCII MZ (hexadecimal 4D 5A), laquelle se réfère aux initiales de Mark Zbikowski, qui fut la personne aux commandes du format des fichiers exécutables sous DOS.

- **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. 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.

```
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".

Entête PE

- **Plateforme d'accueil** *Machine* Configuration matérielle pour laquelle le fichier est destiné, par exemple machine à base de processeurs compatibles 386. Le programme ne peut être utilisé que sur un système compatible avec l'architecture décrite. Sont concevables à cet égard les attributs que voici.

  - 0x0000 - IMAGE_FILE_MACHINE_UNKNOWN

  - 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)

- **Nombre de sections** *NumberOfSections* Nombre de sections (et donc d'entêtes de sections) incorporées au fichier.
- **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*
- **Cachet horodaté** *TimeDateStamp* Enregistre l'heure et la date à laquelle le fichier a été généré.
- **Caractéristiques** (*Characteristics*) Propriétés générales du fichier.
  - 0x0001 - IMAGE_FILE_RELOCS_STRIPPED - Les informations de relogement sont retirées du fichier. C'est généralement le cas en ce qui concerne les images exécutables, tandis que les fichiers objets en sont dotés.
  - 0x0002 - IMAGE_FILE_EXECUTABLE_IMAGE - 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.
  - 0x0004 - IMAGE_FILE_LINE_NUMS_STRIPPED - Les numéros de ligne dans les informations de débogage COFF ont été supprimés.
  - 0x0008 - IMAGE_FILE_LOCAL_SYMS_STRIPPED - Les symboles locaux sont retirés du fichier.
  - 0x0010 - IMAGE_FILE_AGGRESIVE_WS_TRIM
  - 0x0020 - IMAGE_FILE_LARGE_ADDRESS_AWARE - Le processus peut croitre au-delà de 2 Go d'espace d'adressage.
  - 0x0080 - IMAGE_FILE_BYTES_REVERSED_LO - Little endian.
  - 0x0100 - IMAGE_FILE_32BIT_MACHINE
  - 0x0200 - IMAGE_FILE_DEBUG_STRIPPED - Les informations de débogage ont été supprimées.
  - 0x0400 - IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP - Si l'image réside sur un média amovible, copier et exécuter depuis le fichier d'échange.
  - 0x0800 - IMAGE_FILE_NET_RUN_FROM_SWAP - Si l'image réside sur un partage réseau, copier et exécuter depuis le fichier d'échange.
  - 0x1000 - IMAGE_FILE_SYSTEM - Fichier système, par opposition à un programme utilisateur.
  - 0x2000 - IMAGE_FILE_DLL - Le fichier est une DLL.
  - 0x4000 - IMAGE_FILE_UP_SYSTEM_ONLY - L'image ne peut être exécutée que sur un seul processeur.
  - 0x8000 - IMAGE_FILE_BYTES_REVERSED_HI - 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.

- **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.

- *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
  - 0x0800 - IMAGE_DLLCHARACTERISTICS_NO_BIND - Ne pas lier cette image.
  - 0x2000 - IMAGE_DLLCHARACTERISTICS_WDM_DRIVER - Pilote de type WDM.
  - 0x8000 - IMAGE_DLLCHARACTERISTICS_TERMINAL_SERVER_AWARE
- **Alignement du fichier** *FileAlignment* Taille minimale affectée à une zone de données avant son chargement en mémoire. Cette valeur est couramment définie à 512 octets, soit 0x200.
- **Adresse de base de l'image** *ImageBase* Adresse préférentielle utilisée lors de l'enregistrement (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 - cela correspondant à 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.
- **Etat du fichier** (*Magic*)
  - 0x10b - IMAGE_NT_OPTIONAL_HDR32_MAGIC - Exécutable 32 bits
  - 0x20b - IMAGE_NT_OPTIONAL_HDR64_MAGIC - Exécutable 64 bits
  - 0x107 - IMAGE_ROM_OPTIONAL_HDR_MAGIC - Image ROM
- **Alignement d'une section** *SectionAlignment* Taille minimale qu'une section peut occuper en mémoire. Cette valeur est couramment définie à 4096 octets, soit 0x1000.
- **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 de l'exécutable, par exemple le système graphique Windows ou POSIX. Les valeurs suivantes sont définies.
  - 0x00 - IMAGE_SUBSYSTEM_UNKNOWN - Sous-système non répertorié.
  - 0x01 - IMAGE_SUBSYSTEM_NATIVE - Pilotes de périphérique et processus n'utilisant que les API de services système fournies par Ntdll.
  - 0x02 - IMAGE_SUBSYSTEM_WINDOWS_GUI
  - 0x03 - IMAGE_SUBSYSTEM_WINDOWS_CUI
  - 0x05 - IMAGE_SUBSYSTEM_OS2_CUI
  - 0x07 - IMAGE_SUBSYSTEM_POSIX_CUI
  - 0x08 - IMAGE_SUBSYSTEM_NATIVE_WINDOWS
  - 0x09 - IMAGE_SUBSYSTEM_WINDOWS_CE_GUI

- 0x0a - IMAGE_SUBSYSTEM_EFI_APPLICATION
- 0x0b - IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER
- 0x0c - IMAGE_SUBSYSTEM_EFI_RUNTIME_DRIVER
- 0x0d - IMAGE_SUBSYSTEM_EFI_ROM
- 0x0e - IMAGE_SUBSYSTEM_XBOX
- 0x10 - IMAGE_SUBSYSTEM_WINDOWS_BOOT_APPLICATION -

- **Informations de version du linker** Numéros de version qui correspondent, respectivement, à la version majeure (*MajorLinkerVersion*) et mineure (*MinorLinkerVersion*) du logiciel utilisé lors de la phase d'éditions des liens.
- **Taille du code** (*SizeOfCode*) Taille totale occupée par toutes les sections hébergeant du code machine executable. En principe, la valeur de ce champ correspond 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* - Usuellement 16.
- **Adresse du point d'entrée** (*AddressOfEntryPoint*) Information de contexte utilisée par l'environnement d'execution de sorte à déterminer où le programme commence après son chargement en mémoire. Ce champ pointe généralement sur la première instruction exécutable, située quelque part dans une section de code.
- **DllCharacteristics** Caractéristiques supplémentaires dans le cas où le fichier est une DLL.
  - 0x0040 - IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE
  - 0x0080 - IMAGE_DLLCHARACTERISTICS_FORCE_INTEGRITY
  - 0x0100 - IMAGE_DLLCHARACTERISTICS_NX_COMPAT
  - 0x0200 - IMAGE_DLLCHARACTERISTICS_NO_ISOLATION
  - 0x0400 - IMAGE_DLLCHARACTERISTICS_NO_SEH
  - 0x0800 - IMAGE_DLLCHARACTERISTICS_NO_BIND
  - 0x2000 - IMAGE_DLLCHARACTERISTICS_WDM_DRIVER
  - 0x8000 - IMAGE_DLLCHARACTERISTICS_TERMINAL_SERVER_AWARE
- **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. La liste qui suit montre comment s'enchaine les informations au sein de la table des répertoires.
  - 0x00 Table des exports
  - 0x08 Table des imports

- 0x10 Table des ressources
- 0x18 Table des exceptions
- 0x20 Table des certificats
- 0x28 Table des relocalisations
- 0x30 Informations de débogage
- 0x38 Données spécifiques au droits de copies ou à l'architecture
- 0x40 Pointeurs globaux
- 0x48 Table de stockage des threads locaux
- 0x50 Load configuration table
- 0x58 Table des imports liés
- 0x60 Table des adresses des imports
- 0x68 Descripteur des imports en différé
- 0x70 En-tête runtime COM+

- **Informations de piles** Quantités de mémoire (nombre d'octets) définies pour la réservation (*SizeOfStackReserve*) et l'engagement effectif (*SizeOfStackCommit*) de la pile.

Portable Executable

Pour afficher des détailées concernant un module, commencez par en repérer un qui soit chargé en mémoire, ce par le biais de la commande informations La commande lm.

TRAVAUX PRATIQUES Visualisation des informations d'image d'un module

```
lkd> !lmi fffff800`bc489000
Loaded Module Info: [fffff800`bc489000]
         Module: ntkrnlmp
   Base Address: fffff800`bc489000
     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
```

```
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 fffff800`bc489000
   +0x000 e_magic          : 0x5a4d
   +0x002 e_cblp           : 0x90
   +0x004 e_cp             : 3
   +0x006 e_crlc           : 0
   +0x008 e_cparhdr        : 4
   +0x00a e_minalloc       : 0
   +0x00c e_maxalloc       : 0xffff
   +0x00e e_ss             : 0
   +0x010 e_sp             : 0xb8
   +0x012 e_csum           : 0
   +0x014 e_ip             : 0
   +0x016 e_cs             : 0
   +0x018 e_lfarlc         : 0x40
   +0x01a e_ovno           : 0
   +0x01c e_res            : [4] 0
   +0x024 e_oemid          : 0
   +0x026 e_oeminfo        : 0
   +0x028 e_res2           : [10] 0
   +0x03c e_lfanew         : 0n248
```

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....$"
```

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.

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.

- **Nom** - *Name* - Le nom comporte 8 caractères. 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

- **Taille virtuelle** - *VirtualSize* - Taille de la section une fois celle-ci chargée en mémoire.

- **Adresse virtuelle** - *VirtualAddress* - Adresse 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.

- 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.

Chapitre 10. Sécurité

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.

- **Logon Process**, Processus d'ouverture de Session. Processus mode utilisateur (Winlogon.exe) chargé de gérer les sessions interactives.

- **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.

Contrôle d'accès

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.

Les principales interfaces que fournit Windows en matière de SID incluent AllocateAndInitializeSid, CreateWellKnownSi, GetLengthSid, LookupAccountSid, ConvertSidToStringSid.

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.

Niveaux d'intégrité

Pour utiles et dignes d'intérêt qu'elles puissent être, les politiques de contrôle d'accès en œuvre dans les systèmes d'exploitation ne répondent que partiellement aux problématiques auxquelles elles doivent faire, et d'autant plus à l'échelle des usages modernes, passent à côté de certains aspects pourtant cruciaux. Ainsi, définir qui peut faire quoi avec une ressource (soit tout le propos du contrôle d'accès) n'a en définitive que peu à voir avec la stabilité de l'écosystème - si cela influe effectivement, c'est de manière tout à fait parallèle. En basant les moyens de limiter l'accès aux objets uniquement sur l'identité d'un sujet et les habilitations subséquentes, il est dans ce contexte difficile de hiérarchiser les actions effectuées depuis un même compte d'utilisateur. Pensez, par exemple, à la distinction entre un service exécuté sous un compte x, et une application (potentiellement malveillante) que l'utilisateur x à téléchargé en naviguant sur Internet. 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. 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.

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.

Windows définit principalement quatre niveaux d'intégrité : bas, moyen, haut et système. Les applications lancées par un utilisateur standard s'exécutent avec un niveau d'intégrité moyen. Les utilisateurs disposant de droits étendus, par exemple les membres d'un groupe d'administration, ont le niveau haut. Les services du système reçoivent le niveau système. Les objets dépourvus d'étiquette d'intégrité prennent implicitement le niveau moyen.

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.

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

Windows Defender permet de lancer une analyse manuelle du système pour essayer de détecter des applications malveillantes.

1. Démarrez Windows Defender.

2. Sous l'onglet Accueil, choisissez une option d'analyse, puis cliquez sur Analyser maintenant.

Il existe plusieurs façons d'analyser uniquement certains fichiers, lecteurs ou processus avec Windows Defender :

- Une analyse rapide examine uniquement les zones du PC les plus susceptibles d'être infectées par des logiciels malveillants, ainsi que les applications en cours d'exécution.

- Une analyse Complète passe en revue tous les fichiers de la machine.

- Une analyse Personnalisée examine uniquement les fichiers et les emplacements sélectionnés par l'utilisateur.

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
 80510100000000000100000003000000140000000300000064000000000000000640000000000000064000000
    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.

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.

Microsoft Malware Protection Engine

Les technologies de protection rendues visibles par Windows, y compris l'analyse à base de signatures, l'émulation de code et la détection comportementale, sont regroupées au sein d'un programme unique nommé Microsoft Malware Protection Engine.

SmartScreen

Parmi les multiples couches de défense intégrées aux fonctionnalités réseau de Windows, le filtre SmartScreen fournit un système d'alerte anticipée contre les sites Web potentiellement dangereux et les applications malveillantes. Introduite initialement sous Internet Explorer 8, une telle protection équipe à l'heure actuelle un certain nombre de produits 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.

Reposant essentiellement sur le principe de la réputation des individus (classification des emails), des URL et des applications, SmartScreen peut s'avérer notamment efficace dans la protection contre les attaques par ingénierie sociale, en particulier lorsque des logiciels malveillants tentent de se faire passer pour des programmes légitimes. La liste noire des sites et des applications connus pour leur toxicité est mise à jour très régulièrement par Microsoft.

SmartScreen tire parti de la réputation d'une URL - via les informations recueillies en la matière par Authenticode, ce qui signifie que les serveurs hôtes de programmes sont évalués afin de déterminer s'ils servent de moyen de diffusion à des contenus non sûr. Cette analyse, faite de concert avec les autres technologies anti malware intégrées à Windows, dont Defender, est basée sur la télémétrie (remontée des statistiques de téléchargements) et la signature numérique.

Régler les paramètres Windows 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.

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).

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 (KPP)

Kernel Patch Protection (KPP), connu aussi sous le nom de 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 a 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, ndis.sys et consorts)** 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.

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.

Emprunt d'identité

Figure médiatrice entre le modèle de sécurité de Windows et les applications, l'emprunt d'identité est un mécanisme par lequel un thread peut disposer d'un jeton d'accès différent de celui du processus dans lequel il s'exécute. Une telle technique donne de la sorte la possibilité à un processus d'agir pour le compte d'un autre, et plus important encore, de le faire sous un profil de sécurité lui permettant d'accéder à des ressources qui lui seraient interdites en temps normal.

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.

Annexe A. Interfaces

■ **CloseHandle**

∩ kernel32

⧵ BOOL WINAPI

⎧ ① IN HANDLE hObject

→ NtClose

■ **DISPATCHER_HEADER**

≡ Entête dispatcheur ; fournit à divers objets (processus, threads, sémaphore, etc.-) des capacités de synchronisation

⧵ struct DISPATCHER_HEADER, *PDISPATCHER_HEADER

∩ KTHREAD Header, KPROCESS Header, KSEMAPHORE Header…

■ **DesiredAccess**

⧵ IN ACCESS_MASK

∩ NtAccessCheck, NtAccessCheckAndAuditAlarm, NtAccessCheckByType, NtAccessCheckByTypeAndAuditAlarm, NtAccessCheckByTypeResultList, NtAccessCheckByTypeResultListAndAuditAlarm, NtAccessCheckByTypeResultListAndAuditAlarmByHandle, NtCreateDirectoryObject, NtCreateEvent, NtCreateEventPair, NtCreateFile, NtCreateIoCompletion, NtCreateJobObject, NtCreateKey, NtCreateMailslotFile, NtCreateMutant, NtCreateNamedPipeFile, NtCreateProcess, NtCreateSection, NtCreateSemaphore, NtCreateSymbolicLinkObject, NtCreateThread, NtCreateTimer, NtCreateToken, NtDuplicateObject, NtDuplicateToken, NtOpenEvent, NtOpenEventPair, NtOpenDirectoryObject, NtOpenFile, NtOpenIoCompletion, NtOpenJobObject, NtOpenKey, NtOpenMutant, NtOpenObjectAuditAlarm, NtOpenProcessToken, NtOpenProcessTokenEx, NtOpenSection, NtOpenSemaphore, NtOpenSymbolicLinkObject, NtOpenThread, NtOpenTimer, NtPrivilegeObjectAuditAlarm

■ **DirectoryHandle**

⧵ OUT PHANDLE

∩ NtCreateDirectoryObject, NtOpenDirectoryObject

■ **EPROCESS**

≡ Bloc processus de l'exécutif

⧵ struct EPROCESS, *PEPROCESS

∩ PsGetCurrentProcess, PsGetProcessCreateTimeQuadPart, PsGetProcessDebugPort Process (1), PsGetProcessJob Process (1), PsGetProcessExitStatus Process (1), PsGetProcessId Process (1), PsGetProcessInheritedFromUniqueProcessId Process (1), PsGetProcessPeb Process (1), PsGetProcessPriorityClass Process (1), PsGetProcessSectionBaseAddress Process (1), PsGetProcessSessionId(Ex) Process (1), PsGetProcessWin32Process Process(1), PsGetProcessWin32WindowStation Process (1), PsIsProcessBeingDebugged Process (1), PspCreateThread ProcessPointer…

■ **EPROCESS ActiveThreads**

≡ Nombre de threads en fonctionnement à l'intérieur d'un processus.

⧵ ULONG ActiveThreads

← NtQueryInformationThread (ThreadInformationClass = ThreadAmILastThread), PspCreateThread, PspExitThread.

■ **EPROCESS BreakOnTermination**

⧵ ULONG BreakOnTermination: 1;

← NtQueryInformationProcess, NtTerminateProcess, PspExitThread…

■ **EPROCESS Cookie**

⧵ ULONG Cookie

← NtQueryInformationProcess…

■ **EPROCESS ExitStatus**

⧵ NTSTATUS ExitStatus

← NtQueryInformationProcess, PsGetProcessExitStatus, PspCreateProcess, PspExitThread…

■ **EPROCESS ExitTime**

⧵ LARGE_INTEGER ExitTime

← PsGetProcessExitTime

■ **EPROCESS ForkInProgress**

⧵ PETHREAD ForkInProgress

■ **EPROCESS InheritedFromUniqueProcessId**

← PsGetProcessInheritedFromUniqueProcessId, PspCreateProcess…

■ **EPROCESS OutswapEnabled**

⧵ ULONG OutswapEnabled: 1

← PS_PROCESS_FLAGS_OUTSWAP_ENABLED

■ **EPROCESS PriorityClass**

≡ Classe de priorité du processus

⧵ UCHAR PriorityClass

← PsGetProcessPriorityClass, PsSetProcessPriorityClass

■ **EPROCESS SetTimerResolution**

⧵ ULONG SetTimerResolution

1 ←

■ **EPROCESS UniqueProcessId**

← PsGetProcessId, PsGetCurrentProcessId…

■ **EPROCESS VdmAllowed**

≡ Autorisation vis à vis de la machine DOS virtuelle

⧵ ULONG VdmAllowed: 1

← (PROCESSINFOCLASS ProcessWx86Information)

■ **ETHREAD**

≡ Bloc thread de l'exécutif

⧵ struct ETHREAD, *PETHREAD

∩ PsGetCurrentThread, PsGetThreadFreezeCount, PsGetThreadId Thread…

■ **ETHREAD Cid**

⧵ CLIENT_ID Cid

≡ ID client de processus et de thread

■ **ETHREAD Cid**
← PsGetCurrentThreadProcessId, PsGetThreadId, PsGetThreadProcessId…

■ **ETHREAD StartAddress**
⟍ PVOID StartAddress
← PspCreateThread

■ **ExSemaphoreObjectType**
⟍ POBJECT_TYPE
← DISPATCHER_HEADER Type, KeInitializeSemaphore…

■ **ExpGetCurrentUserUILanguage**
⟍ NTSTATUS
⎧ ① IN PWSTR MuiName

■ **GetCurrentProcessId**
≡ Retourne l'ID du processus appelant
⟍ DWORD WINAPI GetCurrentProcessId()

■ **GetExitCodeProcess**
⟍ BOOL WINAPI
→ NtQueryInformationProcess
{params} ① hProcess ② lpExitCode

■ **GetExitCodeThread**
⟍ BOOL GetExitCodeThread
→ NtQueryInformationThread

■ **GetProcessId**
∩ kernel32
⟍ DWORD WINAPI
{params} ① Process
→ NtQueryInformationProcess

■ **GetProcessId Process**
⟍ IN HANDLE
→ NtQueryInformationProcess Process

■ **GetThreadId**
∩ kernel32
⟍ DWORD WINAPI
⎧ ① IN HANDLE Thread
→ NtQueryInformationThread

■ **GetThreadIOPendingFlag**
∩ kernel32
⟍ DWORD WINAPI
⎧ ① IN HANDLE Thread
② ←
IN
OUT
PBOOL
lpIOIsPending

■ **IsThreadAFiber**
∩ kernel32

■ **IsThreadAFiber**
⟍ BOOL WINAPI

■ **KAFFINITY**
⟍ typedef ULONG_PTR KAFFINITY
∩ PROCESS_BASIC_INFORMATION AffinityMask, THREAD_-BASIC_INFORMATION AffinityMask, KeSetAffinityThread Affinity, KeSetSystemAffinityThread Affinity…

■ **KEY_SET_INFORMATION_CLASS**
⟍ enum KEY_SET_INFORMATION_CLASS
∩ NtSetInformationKey

■ **KMUTANT**
⟍ struct KMUTANT, *PKMUTANT, *PRKMUTANT, KMUTEX, *PKMUTEX, *PRKMUTEX
∩ KeReleaseMutant Mutant

■ **KPRCB**
⟍ struct KPRCB, *PKPRCB
∩ KPCR Prcb

■ **KSEMAPHORE**
⟍ struct KSEMAPHORE, *PKSEMAPHORE, *PRKSEMAPHORE
∩ KeInitializeSemaphore Semaphore, KeReadStateSemaphore Semaphore, KeReleaseSemaphore Semaphore, ObInsertObject Object…

■ **KTIMER**
⟍ struct KTIMER, *PKTIMER, *PRKTIMER
∩ ETIMER KeTimer, KTHREAD Timer, KeCancelTimer Timer, KeClearTimer Timer, KeInitializeTimer(Ex) Timer, KeQueryTimerDueTime Timer, KeReadStateTimer Timer, KeSetTimer(Ex) Timer.

■ **KTHREAD**
≡ Bloc thread du noyau
⟍ struct KTHREAD, *PKTHREAD
∩ KAPC, KeAlertResumeThread, KeAlertThread, KeBoostPriorityThread, KeForceResumeThread, KeGetCurrentThread, KeInitializeThread, KeInitThread, KeQueryAutoAlignmentThread, KeQueryBasePriorityThread, KeQueryPriorityThread, KeQueryRuntimeThread, KeReadyThread, KeReadStateThread, KeResumeThread, KeSetAffinityThread, KeStartThread, KeSuspendThread, KeUninitThread, ObDereferenceObject…

■ **KTHREAD AutoAlignment**
⟍ ULONG AutoAlignment: 1
← KeQueryAutoAlignmentThread…

■ **KTHREAD FreezeCount**
⟍ CHAR FreezeCount
≡ Nombre de fois qu'un thread a été suspendu
← KeFreezeAllThreads, PsGetThreadFreezeCount.

■ **KTHREAD GuiThread**
≡ Indique quand un thread est apparenté à une interface graphique

■ **KTHREAD GuiThread**
⟍ ULONG GuiThread: 1

■ **KTHREAD MutantListHead**
⟍ LIST_ENTRY MutantListHead
≡ Liste des objets mutant qu'un thread possède

■ **KeProcessorArchitecture**
⟍ USHORT KeProcessorArchitecture
≡ Architecture matérielle sous-jacente aux processeurs de la machine
← KiInitializeKernel, ExpGetSystemEmulationProcessorInformation, ExpGetSystemProcessorInformation…

■ **NtAdjustGroupsToken**
⟍ NTSYSAPI NTSTATUS NTAPI
← AdjustTokenGroups

■ **NtAdjustPrivilegesToken**
⟍ NTSYSAPI NTSTATUS NTAPI
← AdjustTokenPrivileges

■ **NtAlertResumeThread**
⟍ NTSYSAPI NTSTATUS NTAPI
⎧ ThreadHandle, PreviousSuspendCount

■ **NtAlertThread**
⟍ NTSYSAPI NTSTATUS NTAPI
⎧ ① IN HANDLE ThreadHandle

■ **NtAllocateLocallyUniqueId**
⟍ NTSYSAPI NTSTATUS NTAPI
← AllocateLocallyUniqueId

■ **NtCancelTimer**
⟍ NTSYSAPI NTSTATUS NTAPI
⎧ ① IN HANDLE TimerHandle
② OUT OPT PBOOLEAN CurrentState
← CancelWaitableTimer

■ **NtClose**
⟍ NTSYSAPI NTSTATUS NTAPI
⎧ Handle
← CloseHandle

■ **NtCreateMutant**
⟍ NTSYSAPI NTSTATUS NTAPI
← Kernel32!CreateMutex…

■ **NtCreateProcess**
⟍ NTSYSAPI NTSTATUS NTAPI
← CreateProcess, CreateProcessAsUser

■ **NtCreateSemaphore**
⟍ NTSYSAPI NTSTATUS NTAPI

■ **NtCreateSemaphore**
← CreateSemaphore

■ **NtCreateThread**
⟍ NTSTATUS NtCreateThread
← Kernel32!CreateRemoteThread, Kernel32!CreateThread…

■ **NtCreateToken**
⟍ NTSTATUS NtCreateToken

■ **NtDuplicateToken**
⟍ NTSYSAPI NTSTATUS NTAPI
⎧ ExistingTokenHandle, DesiredAccess, ObjectAttributes, EffectiveOnly, TokenType, NewTokenHandle
← DuplicateToken, DuplicateTokenEx…

■ **NtEnableLastKnownGood**
⟍ NTSYSAPI NTSTATUS NTAPI

■ **NtEnumerateKey**
⟍ NTSYSAPI NTSTATUS NTAPI
⎧ ① IN HANDLE KeyHandle
② IN ULONG Index
③ IN KEY_INFORMATION_CLASS KeyInformationClass
④ OUT PVOID KeyInformation
⑤ IN ULONG KeyInformationLength
⑥ OUT PULONG ResultLength
← RegEnumKey, RegEnumKeyEx.

■ **NtFlushWriteBuffer**
⟍ NTSYSAPI NTSTATUS NTAPI

■ **NtGetContextThread**
⟍ NTSYSAPI NTSTATUS NTAPI
⎧ ① IN HANDLE ThreadHandle
② OUT PCONTEXT Context
← GetThreadContext…

■ **NtLockVirtualMemory**
∩ ntoskrnl
⟍ NTSYSAPI NTSTATUS NTAPI
⎧ ① IN HANDLE ProcessHandle
② IN OUT PVOID *BaseAddress
③ IN OUT PULONG LockSize
④ IN ULONG LockType
← VirtualLock

■ **NtMakeTemporaryObject**
⟍ NTSYSAPI NTSTATUS NTAPI
⎧ ① IN HANDLE Handle

■ **NtOpenProcess**
⟍ NTSYSAPI NTSTATUS NTAPI
⎧ ① OUT PHANDLE ProcessHandle
② IN ACCESS_MASK DesiredAccess
③ IN POBJECT_ATTRIBUTES ObjectAttributes

■ **NtOpenProcess**
④ IN OPT PCLIENT_ID ClientId
← OpenProcess

■ **NtOpenProcessToken**
↘ NTSTATUS
⎧ ① IN HANDLE ThreadHandle
② IN ACCESS_MASK DesiredAccess
③ IN BOOLEAN OpenAsSelf
④ OUT PHANDLE TokenHandle
← OpenProcessToken

■ **NtOpenSection**
↘ NTSTATUS NtOpenSection
← Kernel32!OpenFileMapping

■ **NtOpenSemaphore**
↘ NTSYSAPI NTSTATUS NTAPI
← OpenSemaphore

■ **NtOpenSection DesiredAccess**
∩ NtOpenSection <2>
↘ IN ACCESS_MASK DesiredAccess
← MapViewOfFile dwDesiredAccess

■ **NtOpenThread**
↘ NTSYSAPI NTSTATUS NTAPI
← OpenThread

■ **NtProtectVirtualMemory**
↘ NTSYSAPI NTSTATUS NTAPI
⎧ ① IN HANDLE ProcessHandle
② IN OUT PVOID *BaseAddress
③ IN OUT PULONG ProtectSize
④ IN ULONG NewProtect
⑤ OUT PULONG OldProtect
← VirtualProtect, VirtualProtectEx…

■ **NtQueryDefaultLocale**
↘ NTSYSAPI NTSTATUS NTAPI
⎧ ① IN BOOLEAN ThreadOrSystem
② OUT PLCID Locale

■ **NtQueryInformationThread**
↘ NTSYSAPI NTSTATUS NTAPI
⎧ ① IN HANDLE ThreadHandle
② IN THREADINFOCLASS ThreadInformationClass
③ OUT PVOID ThreadInformation
④ IN ULONG ThreadInformationLength
⑤ OUT OPT PULONG ReturnLength
← GetThreadIOPendingFlag, GetExitCodeThread, GetThreadId, GetThreadPriority, GetThreadPriorityBoost, GetThreadTimes, GetThreadSelectorEntry, GetProcessIdOfThread…

■ **NtQueryKey**
↘ NTSYSAPI NTSTATUS NTAPI

■ **NtQueryKey**
⎧ ① IN HANDLE KeyHandle
② IN KEY_INFORMATION_CLASS KeyInformationClass
③ OUT PVOID KeyInformation
④ IN ULONG KeyInformationLength
⑤ OUT PULONG ResultLength
← RegQueryInfoKey…

■ **NtQuerySemaphore**
↘ NTSYSAPI NTSTATUS NTAPI
⎧ SemaphoreHandle, SemaphoreInformationClass, SemaphoreInformation, SemaphoreInformationLength, ResultLength

■ **NtQueryVirtualMemory**
↘ NTSYSAPI NTSTATUS NTAPI
⎧ ① IN HANDLE ProcessHandle
② IN PVOID BaseAddress
③ IN MEMORY_INFORMATION_CLASS MemoryInformationClass
④ OUT PVOID MemoryInformation
⑤ IN ULONG MemoryInformationLength
⑥ OUT OPT PULONG ReturnLength
← VirtualQuery

■ **NtQueueApcThread**
↘ NTSYSAPI NTSTATUS NTAPI
← QueueUserAPC

■ **NtReleaseSemaphore**
↘ NTSYSAPI NTSTATUS NTAPI
← ReleaseSemaphore

■ **NtResumeThread**
↘ NTSTATUS NtResumeThread
← Kernel32!ResumeThread
→ PsResumeThread

■ **NtSetContextThread**
↘ NTSTATUS NtSetContextThread
← Kernel32!SetThreadContext…

■ **NtSetInformationKey**
↘ NTSTATUS NtSetInformationKey

■ **NtSetInformationThread**
↘ NTSTATUS NtSetInformationThread
← SetThreadAffinityMask, SetThreadIdealProcessor, SetThreadPriority, SetThreadPriorityBoost…

■ **NtSetInformationToken**
↘ NTSYSAPI NTSTATUS NTAPI
⎧ ① IN HANDLE TokenHandle
② IN TOKEN_INFORMATION_CLASS TokenInformationClass
③ IN PVOID TokenInformation
④ IN ULONG TokenInformationLength

■ **NtSetInformationToken**
← SetTokenInformation

■ **NtSetTimerResolution**
✎ NTSTATUS NtSetTimerResolution

■ **NtShutdownSystem**
✎ NTSTATUS NtShutdownSystem
← user32!ExitWindows(Ex), advapi32! InitiateSystemShutdown(Ex)…

■ **NtSuspendProcess**
✎ NTSTATUS NtSuspendProcess
→ PsSuspendProcess

■ **NtSuspendThread**
✎ NTSYSAPI NTSTATUS NTAPI
⎧ ① IN HANDLE ThreadHandle
②←
OUT
OPT
PULONG
PreviousSuspendCount
SuspendThread

■ **NtTerminateThread**
✎ NTSYSAPI NTSTATUS NTAPI
⎧ ① IN HANDLE ThreadHandle
② IN NTSTATUS ExitStatus
← ExitThread, TerminateThread…
→ PspTerminateThreadByPointer…

■ **NtTestAlert**
✎ NTSYSAPI NTSTATUS NTAPI
→ KeTestAlertThread

■ **ObOpenObjectByName**
✎ NTSTATUS ObOpenObjectByName()
← NtOpenSection

■ **ObOpenObjectByName ObjectAttributes**
∩ ObOpenObjectByName <1>
✎ IN POBJECT_ATTRIBUTES ObjectAttributes

■ **ObjectAttributes**
✎ IN POBJECT_ATTRIBUTES
∩ NtCreateDirectoryObject, NtCreateEvent, NtCreateEventPair, NtCreateFile, NtCreateIoCompletion, NtCreateJobObject, NtCreateKey, NtCreateMailslotFile, NtCreateMutant, NtCreateNamedPipeFile, NtCreatePort, NtCreateProcess, NtCreateSection, NtCreateSemaphore, NtCreateSymbolicLinkObject, NtCreateThread, NtCreateTimer, NtCreateToken, NtCreateWaitablePort, NtDuplicateToken, NtLoadKey, NtNotifyChangeMultipleKeys, NtOpenDirectoryObject, NtOpenEvent, NtOpenEventPair, NtOpenFile, NtOpenIoCompletion, NtOpenJobObject, NtOpenKey, NtOpenMutant, NtOpenProcess, NtOpenSection, NtOpenSemaphore, NtOpenSymbolicLinkObject, NtOpenThread, NtQueryAttributesFile, NtQueryOpenSubKeys, NtReplaceKey

■ **OpenFileMapping**
✎ HANDLE OpenFileMapping()
→ NtOpenSection

■ **OpenFileMapping bInheritHandle**
✎ IN BOOL bInheritHandle
∩ OpenFileMapping
rdx→

■ **OpenProcess**
∩ kernel32
✎ HANDLE WINAPI
⎧ ① IN DWORD dwDesiredAccess
② IN BOOL bInheritHandle
③ IN DWORD dwProcessId
→ NtOpenProcess

■ **PROCESSINFOCLASS**
✎ enum PROCESSINFOCLASS
∩ NtQueryInformationProcess ProcessInformationClass, NtSetInformationProcess ProcessInformation, PspQueryQuotaLimits ProcessInformationClass, PspQueryWorkingSetWatch ProcessInformationClass, PspSetQuotaLimits ProcessInformationClass

■ **PsActiveProcessHead**
✎ LIST_ENTRY PsActiveProcessHead

■ **PsGetThreadFreezeCount**
✎ ULONG PsGetThreadFreezeCount

■ **PsGetThreadFreezeCount Thread**
∩ PsGetThreadFreezeCount <1>
✎ IN PETHREAD Thread

■ **PsThreadType**
✎ PBOBJECT_TYPE PsThreadType

■ **SECTION_INFORMATION_CLASS**
✎ enum SECTION_INFORMATION_CLASS

■ **SECTION_INFORMATION_CLASS SectionBasicInformation**
✎ SectionBasicInformation = 0

■ **SECTION_INFORMATION_CLASS SectionImageInformation**
✎ SectionImageInformation = 1

■ **SHUTDOWN_ACTION**
✎ enum SHUTDOWN_ACTION
∩ NtShutdownSystem Action…

■ **SHUTDOWN_ACTION ShutdownNoReboot**
✎ ShutdownNoReboot = 0

■ **SHUTDOWN_ACTION ShutdownReboot**
⟍ ShutdownReboot = 1

■ **SHUTDOWN_ACTION ShutdownPowerOff**
⟍ ShutdownPowerOff = 2

■ **TEB CurrentLocale**
≡ Identificateur de locale
← GetThreadLocale, SetThreadLocale…

■ **TEB HardErrorMode**
⟍ ULONG HardErrorMode
← GetThreadErrorMode, SetThreadErrorMode…

■ **TEB LastErrorValue**
⟍ ULONG LastErrorValue
← Kernel32!GetLastError, Kernel32!SetLastError.

■ **THREADINFOCLASS**
⟍ enum THREADINFOCLASS
∩ NtQueryInformationThread, NtQueryInformationThread…

■ **THREADINFOCLASS ThreadAmILastThread**
⟍ ThreadAmILastThread = 12

■ **THREADINFOCLASS ThreadBasicInformation**
⟍ ThreadAmILastThread = 0

■ **TOKEN_TYPE**
⟍ enum TOKEN_TYPE
∩ NtCreateToken…

■ **TOKEN_TYPE TokenImpersonation**
⟍ TokenImpersonation = 1

■ **TOKEN_TYPE TokenPrimary**
⟍ TokenPrimary = 0

■ **WaitForMultipleObjectsEx bAlertable**
∩ WaitForMultipleObjectsEx rsp
⟍ BOOL bAlertable
→ NtWaitForMultipleObjects r9

■ **NtTerminateJobObject**
≡ Met fin à un job et aux processus et aux threads qu'il contient
← TerminateJobObject

■ **PreviousSuspendCount**
∩ NtAlertResumeThread, NtResumeThread, NtSuspendThread
⟍ OUT PULONG OPTIONAL

■ **PsDefaultSystemLocaleId**
⟍ LCID PsDefaultSystemLocaleId

■ **PsGetProcessId**
⟍ HANDLE PsGetProcessId()
→ EPROCESS UniqueProcessId

■ **PsGetProcessImageFileName**
⟍ UCHAR *PsGetProcessImageFileName
→ EPROCESS ImageFileName

■ **PsIsProcessBeingDebugged**
⟍ BOOLEAN PsIsProcessBeingDebugged
→ EPROCESS DebugPort

■ **SemaphoreHandle**
⟍ IN HANDLE
∩ NtQuerySemaphore, NtReleaseSemaphore

■ **SemaphoreHandle**
⟍ OUT PHANDLE
∩ NtCreateSemaphore, NtOpenSemaphore

■ **TokenType**
⟍ IN TOKEN_TYPE
∩ NtCreateToken, NtDuplicateToken

■ **VirtualQuery**
∩ kernel32
⟍ SIZE_T WINAPI
⎧ ① IN OPT LPCVOID lpAddress
② OUT PMEMORY_BASIC_INFORMATION lpBuffer
③ IN SIZE_T dwLength

■ **VirtualLock**
∩ kernel32
⟍ BOOL WINAPI
⎧ ① IN LPVOID lpAddress
② IN SIZE_T dwSize
→ NtLockVirtualMemory

■ **VirtualUnlock**
∩ kernel32
⟍ BOOL WINAPI
⎧ ① IN LPVOID lpAddress
② IN SIZE_T dwSize
→ NtUnlockVirtualMemory

■ **VirtualProtect**
∩ kernel32
⟍ BOOL WINAPI
⎧ ① IN LPVOID lpAddress
② IN SIZE_T dwSize
③ IN DWORD flNewProtect
④ OUT PDWORD lpflOldProtect
→ NtProtectVirtualMemory

Glossaire

ABI (Application Binary Interface)
: format binaire respecté par les points d'entrée d'un code compilé.

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.

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.

API
: Ensemble de fonctions et de structures de données constituant l'interface entre plusieurs composants logiciels quand ils doivent collaborer.

API Windows
: Ensemble de fonctions, de procédés et de moyens par lesquels les applications Windows interagissent avec les fonctionnalités des systèmes d'exploitation de la famille Microsoft Windows.

Affinité de processeur
: Ensemble des processeurs sur lesquels un thread a le droit d'être exécuté.

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.

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.

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 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.

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.

Bus
Ensemble de liaisons électroniques permettant la circulation des données entre le processeur, la mémoire vive et les cartes d'extension.

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.

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.

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.

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.

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.

Contexte d'exécution
Environnement d'exécution d'un processus ou d'un thread (registres, zones mémoires…)

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, et d'autres.

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).

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 informatique via lequel 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
Programme d'aide à l'élimination des bogues.

Descripteur de sécurité
Structure de données qui spécifie qui peut faire quoi sur un objet.

Droits d'accès désirés
Accès désirés par un thread ouvrant un objet.

Emprunt d'identité
Mécanisme par lequel un thread peut avoir un jeton d'accès différent de celui du processus dans lequel dans lequel il s'exécute.

Exclusion mutuelle
Processus par lequel un thread ou processus et un seul est autorisé à accéder à une ressource à un moment donné.

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.

Flot
Suite d'octets dans un fichier.

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 .

Hôte
Désigne potentiellement tout ordinateur qui prend le rôle de source de l'information.

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).

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.

Interblocage
Phénomène se produisant lorsque deux processus s'empêchent mutuellement d'accéder à une ressource partagée. Anglais : deadlock.

Intel 64
Technologie d'extension mémoire Intel EM64T (Extended Memory 64 Technology).

Interface
Dispositif servant de jonction entre deux matériels ou logiciels leur permettant d'échanger des informations par l'adoption de règles communes.

Lecture anticipée intelligente
Technique permettant de prédire les données qu'un thread est susceptible de lire prochainement, ce en se basent sur les données qu'il est en train de lire.

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.

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.

MSDN (Microsoft Developer Network)
Programme d'assistance Microsoft destiné aux concepteurs de logiciels. Pour plus d'informations, voir msdn.microsoft.com.

Mémoire partagée
Mémoire visible à plus d'un processus.

Mutant
Nom interne du mutex.

Mutex
Mécanisme de synchronisation ayant fonction de protéger une section critique (comprendre un intervalle défini d'instructions) utilisée par plusieurs threads (ou processus) et à laquelle un seul thread, au plus, doit avoir accès à la fois.

NTFS
Technique de stockage et de gestion des fichiers sur un disque dur. NTFS est le système utilisé par défaut sous Windows.

Nanoseconde
Un milliardième de seconde.

Netbios
Protocole pour réseaux locaux conçu par IBM puis repris par Microsoft sous le nom NetBEUI.

OEM
Désigne un produit vendu sous un autre nom que celui du manufacturier.

Octet
Unité d'information composée de 8 bits, soit 256 valeurs différentes. Les valeurs possibles sont comprises entre 0 et 255 (décimal) ou 0 et FF (hexadécimal). 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.

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.

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.

Plug & Play
Fonction de Windows permettant la reconnaissance et l'installation automatique des périphériques connectés à l'ordinateur.

Portabilité
Aptitude d'un programme à être utilisé sur des systèmes informatiques de types différents et/ou ayant des conditions d'utilisation différentes.

Presse-papiers
Espace en mémoire utilisé afin de stocker provisoirement des données de toutes sortes, incluant images, textes, etc.

Programme
Suite statistique d'instructions.

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.

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.

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.

Section crique
Primitive de synchronisation utilisée de sorte à coordonner les accès des threads aux ressources privées d'un processus.

Shell
Interpréteur de commandes sous les système Unix et dérivés.

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.

Thread
Entité exécutable ordonnancée par le dispatcheur du noyau, souvent associée à un processus qui encapsule un espace d'adressage virtuel.

Thread principal
constitue l'assise d'éventuels nouveaux threads.

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.

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.

Unicode
Standard d'encodage de caractères représentant la quasi-totalité des langues écrites du monde. Le répertoire de caractères Unicode a plusieurs

formulaires de présentation, notamment UTF-8, UTF-16 et UTF-32. La plupart des interfaces Windows utilisent le formulaire UTF-16.

X509
: Format utiliśe 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.