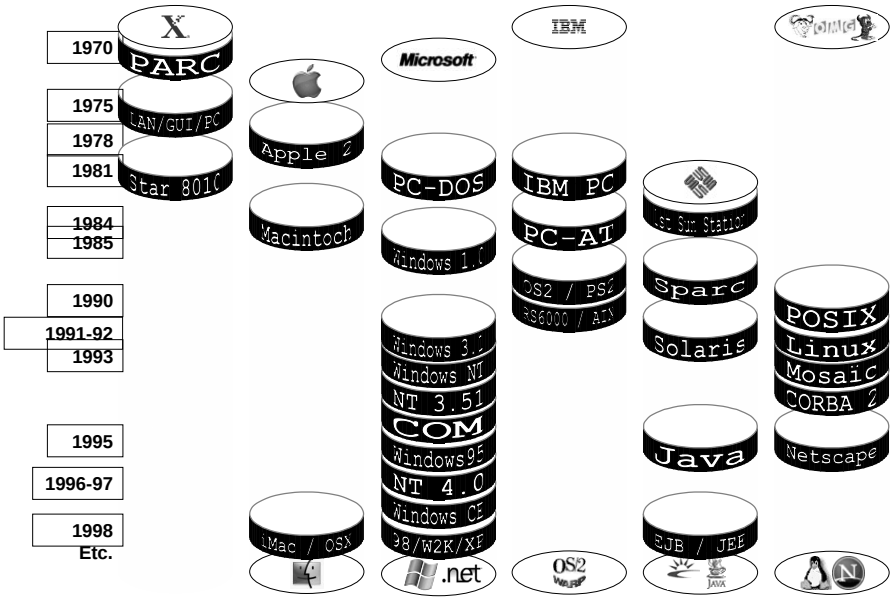
Windows XP
Fonctionnalités du noyau

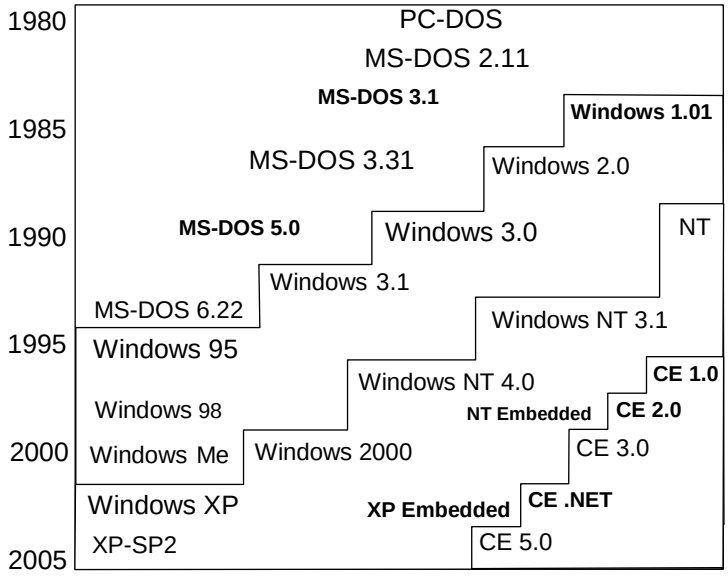
Thierry JOUBERT

Introduction

Quelques acteurs…



Les OS Microsoft

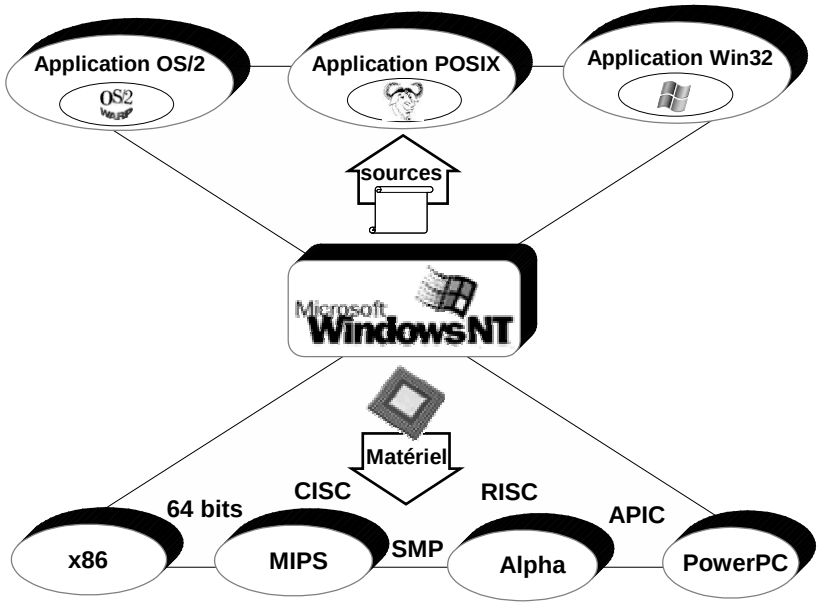


XP = Une harmonisation …

- Unification de la famille Windows pour x86
  - La lignée des 9x s'éteint enfin et avec elle :
    - Un système 16 bits
    - Une absence de sécurité
    - Une instabilité chronique
  - Une seule structure de noyau pour Windows et donc :
    - La même gestion mémoire et les mêmes API
    - Les mêmes pilotes
    - Les mêmes procédures d'installation
    - Le même outil de développement
- L'embarqué simplifié
  - XP Embedded n'est qu'un sous ensemble choisi de XP Professional
  - CE .net reste à part :
    - Temps Réel
    - Supporte d'autres processeurs (ARM, SH3, MIPS)

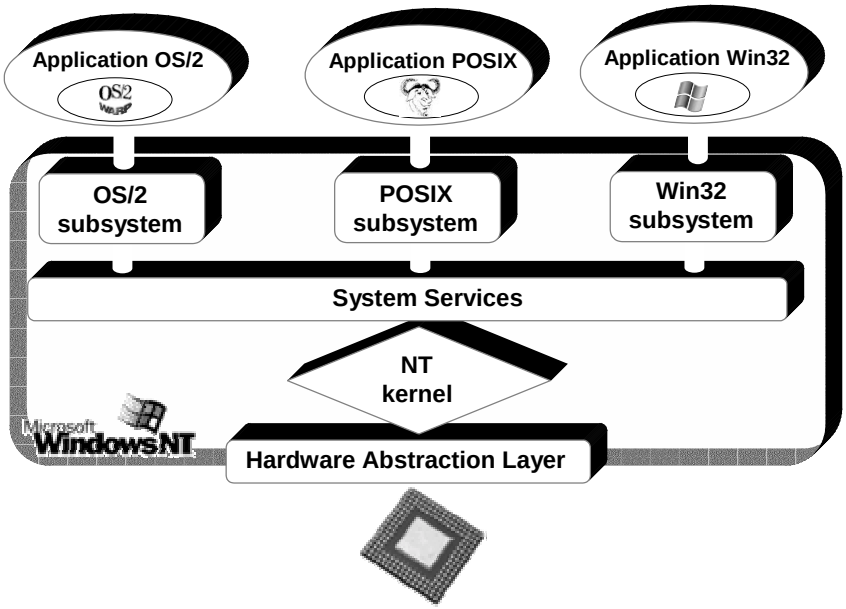
THEORIS © 2005 5

Objectifs de NT



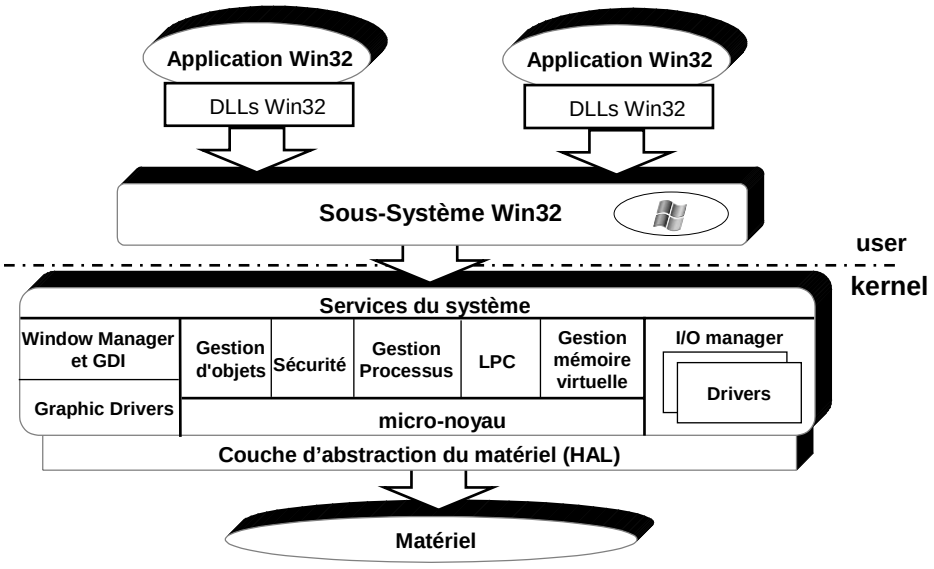
THEORIS © 2005 6

Choix d'architecture NT



THEORIS © 2005 7

Architecture XP



THEORIS © 2005 8

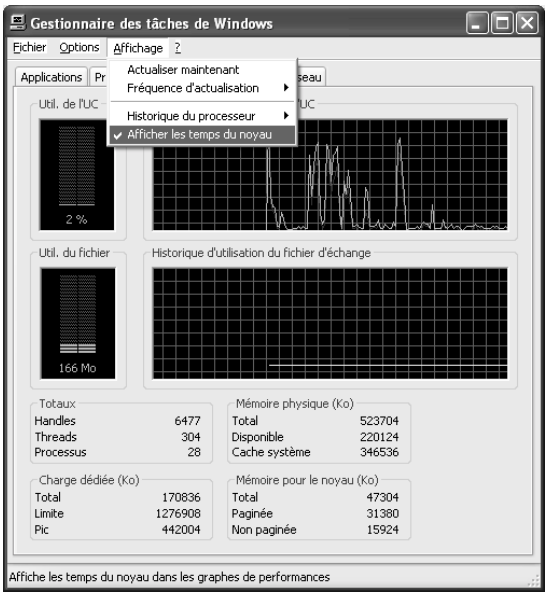
Mode “Kernel” & Mode “User”

- Deux modes gérés par Windows XP
  - Kernel (OS)
    - Mode CPU privilégié
    - Accès aux données du système et au Hardware
  - User (Applications)
    - Mode CPU non-privilégié
    - Pas d’accès au hardware et aux données système
- Les applications sont séparées de l’OS
  - Ells ne peuvent pas modifier son intégrité
- Les composants système et les pilotes peuvent atteindre les donnée du système

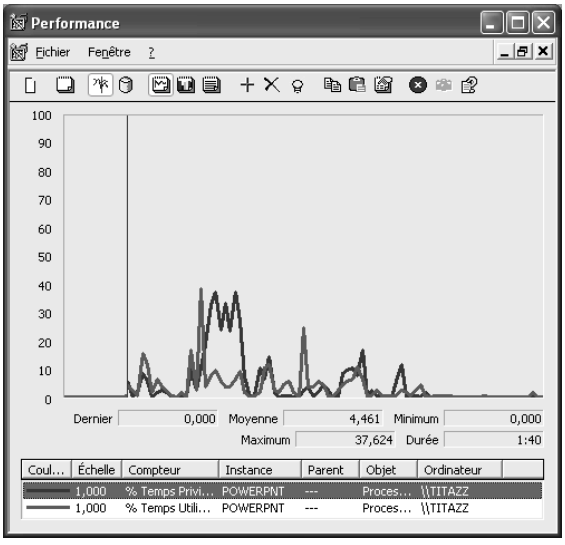
Quelques outils…

Outil	*EXE*	*source*	*Fonction*
Performance Monitor	**perfmon**	**natif**	**Etat système, compteurs de perf., journaux**
Task Manager	**taskman**	**natif**	**Information générales sur le système**
Process Viewer	**pview**	**Resource Kit**	**Détails des processus et des threads**
Process Explorer	**procexp**	**Sysinternals.com**	**Taskman évolué**
Registry Monitor	**regmon**	**Sysinternals.com**	**Activité du registre**
File Monitor	**filemon**	**Sysinternals.com**	**Activité fichiers**

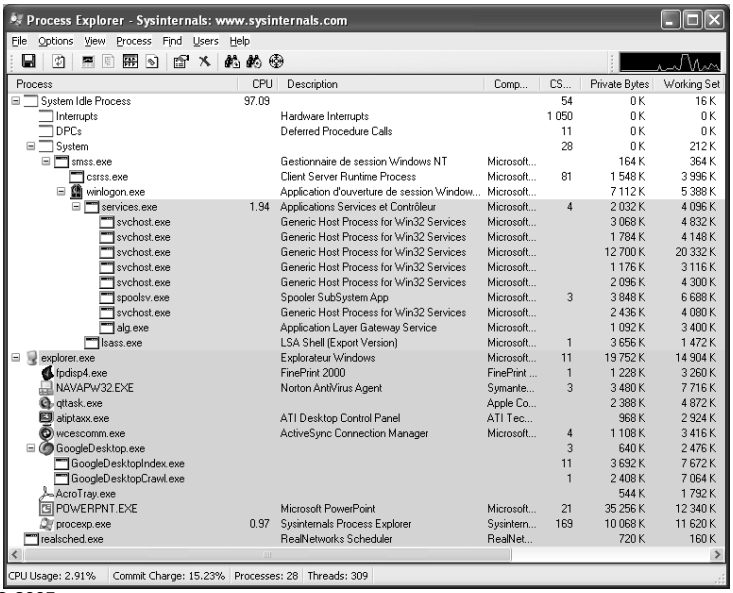
Task Manager



Performance Monitor

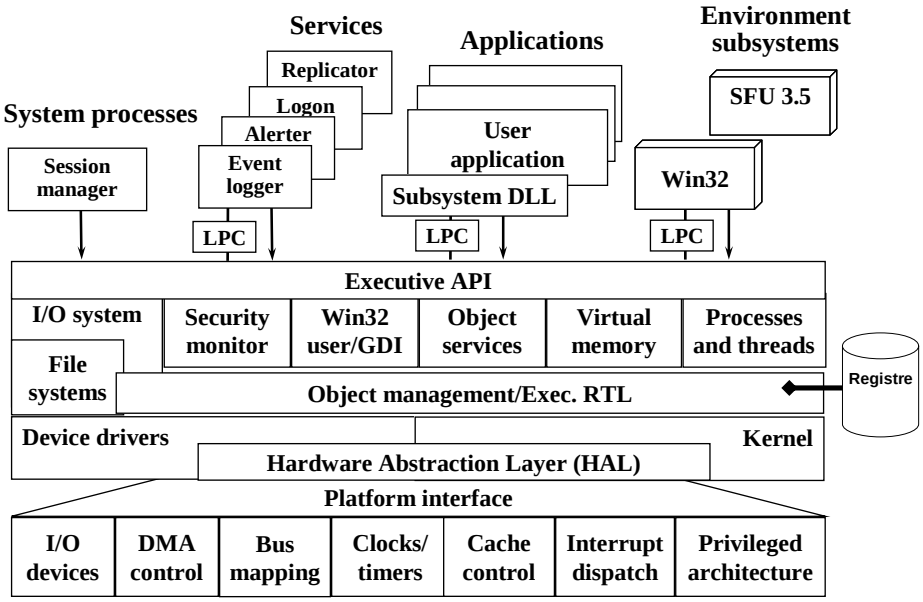


Process Explorer



Architecture détaillée

Architecture détaillée XP



Composants du système (1)

- Services du système XP
  - Couche hute de l'OS
  - Fournit des fonctions génériques
    - Création/destruction de processus et de threads
    - Gestion de la mémoire
    - Entrées/Sorties
    - Communication inter-processus (IPC)
    - Sécurité
  - Tourne en mode Kernel
  - C'est la véritable "API système" de XP !!
    - Elle est utilisée par les développeurs de sous-systèmes

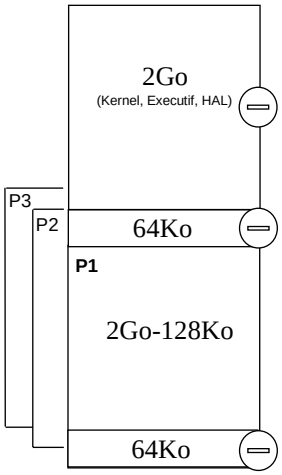
Composants du système (2)

- Noyau Windows XP
  - Fonctions de bas niveau
    - Ordonnancement des threads
    - Gestionnaires d'interruptions et d'exceptions
    - Synchronisation multiprocesseur
- HAL (Hardware Abstraction Layer)
  - Isole le code du système des spécificités matérielles
    - Ex: différences entre les cartes mères
  - Présente un modèle unifié d'entrées/Sorties aux pilotres de périphériques

Composants du système (3)

- Pilotes de périphériques
  - Modules "kernel" chargeables dans le contexte du composant "I/O manager"
    - Systèmes de fichiers, protocoles réseau, matériels …
  - Les pilotes appelent les fonctions de la HAL pour dialoguer avec le matériel
  - On peut voir les pilotes chargés avec l'outil de gestion de Windows XP
- Win32 User/GDI (Graphics Device Interface)
  - Implémente l'interface graphique (GUI)
  - Gestion des fenêtres et des contrôles
  - Fonctions de dessin

Mapping mémoire XP



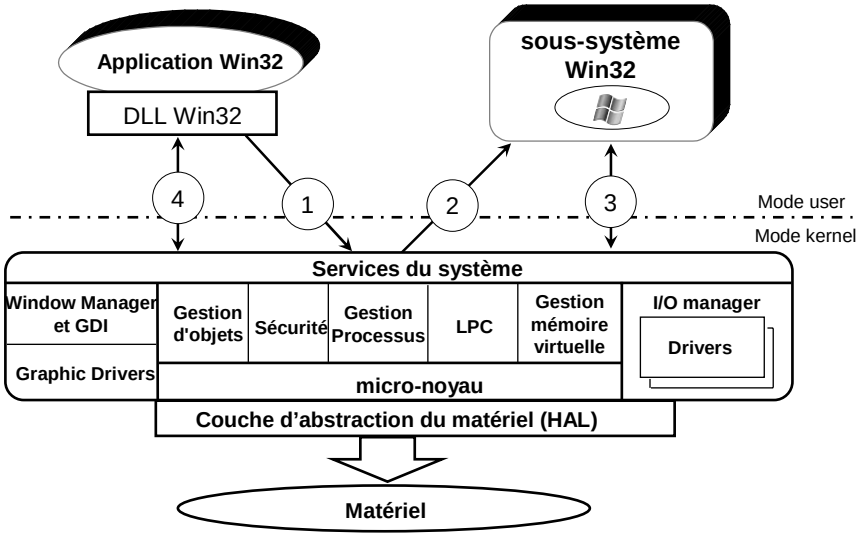
Zone réservée à Windows XP
Tout accès se traduit par un arrêt du processus fautif

Zone réservée à la détection des pointeurs invalides - Inutilisée.

Zone d'adressage privée du processus

Zone réservée à la détection des pointeurs invalides - Inutilisée.

Appel des sous-systèmes



Composants sous-système Win32

- DLLs de l'API Win32
  - Kernel32.DLL
  - Gdi32.DLL
  - User32.DLL
- Processus du sous-système Win32
  - CSRSS.EXE
- Code GDI en mode kernel

Rôle des Composants

- DLLs de l'API Win32
  - Exportent les fonctions de l'API
  - USER32, KERNEL32, GDI32 → NTOSKRNL
  - Implémentent les fonctions en utilisant les services "natifs", ou en demandant au sous-système de le faire
- Processus du sous-système Win32
  - Maintient l'état du système
- Win32K.SYS
  - Impléments les fonctions du GDI Win32
  - Utilisé par les autres sous systèmes

Types de fichiers .EXE

- Sous-système spécifié dans l'entête de chaque EXE

IMAGE_SUBSYSTEM_UNKNOWN	0	Unknown Subsystem
IMAGE_SUBSYSTEM_NATIVE	1	Image doesn't require subsystem
IMAGE_SUBSYSTEM_WINDOWS_GUI	2	Win32 subsystem (graphical app)
IMAGE_SUBSYSTEM_WINDOWS_CUI	3	Win32 subsystem (character app)
IMAGE_SUBSYSTEM_OS2_CUI	5	OS/2 subsystem
IMAGE_SUBSYSTEM_POSIX_CUI	7	POSIX subsystem

- EXE utilisant directement l'exécutif
  - Smss.exe (Session Manager)
  - Csrss.exe (Win32 subsystem)

Accès au Noyau XP

NTOSKRNL.EXE

- Coeur du système XP
  - Contient le noyau et l'exécutif
  - Les fonctions sont exposées aux processus "user" via NtDll.Dll et les sous-systèmes
  - Quatre versions :
    - NTOSKRNL.EXE Uniprocesseur
    - NTKRNLMP Multiprocesseur
    - NTKRNLPA Uniprocesseur avec PAE*
    - NTKRPAMP Multiprocesseur avec PAE*

**PAE = Physical Address Extensions*

Préfixes des fonctions "kernel"

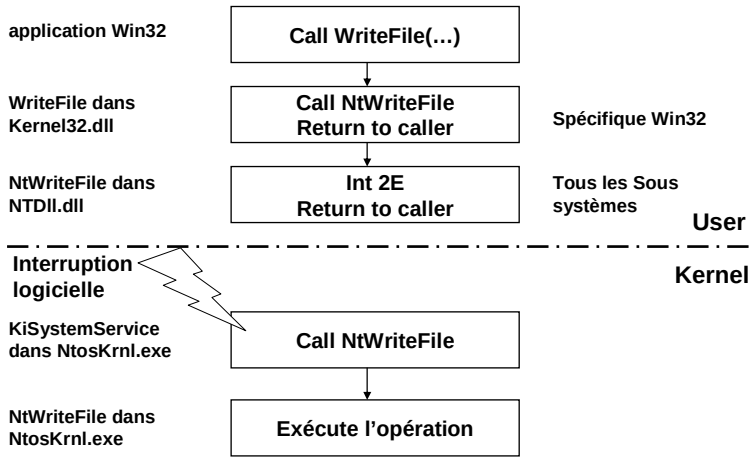
- Deux ou trois premières lettres des noms de fonctions :

Prefix	Component
Cc	Cache Manager
Ex	Executive Support Routines
Hal	Hardware Abstraction Layer
Io	I/O Subsystem
Ke	Kernel
Lsa	Security Authentication
Mm	Memory Manager
Ps	Process support
Rtl	Run-time library

Appel des fonctions du “kernel” depuis le mode “user”

- Utilisation d’un mécanisme protégé :
  - x86: INT 2E (décimal 46)
  - Sur un appel de service système depuis le mode user, la dernière chose qui se passe est une instruction “basculer en mode kernel”
  - Provoque une interruption qui est prise en charge en mode “kernel” par le gestionnaire du système
  - Le retour en mode “user” est fait lors du retour de l’interruption

Appel d’une API “Kernel”



Appel de fonction système en mode user

- La fonction système est désignée par son “numéro de service système”
  - Chaque fonction exposée au monde “user” a un numéro unique
  - Le numéro est mis sur la pile juste avant la demande de commutation en mode “kernel”
  - Les numéros ne sont pas documentés
    - “Encapsulés” dans les fonctions de NTDLL.DLL, USER32.DLL et GDI32.DLL

Différence d’API

- Win32 vs. NtDll.Dll
  - Les API Win32 “kernel” exportées par Kernel32.dll sont différentes des “API natives” dans NtDll.Dll
    - Arguments différents (mais cohérents)
  - Les fonctions de Kernel32.dll ré-arrangent les arguments avant d’appeler NtDll.dll
  - NtDll.dll utilise la commutation de mode (INT 2E) pour passer en mode ”kernel”

Qui fait quoi?

Fichier	*Composants*
NTOSKRNL.EXE	Executif et Noyau
HAL.DLL	Hardware Abstraction Layer
WIN32K.SYS	Partie kernel du sous système Win32
NTDLL.DLL	Fonction internes / dispatch vers l'exécutif
KERNEL32.DLL, ADVAPI32.DLL, USER32.DLL, GDI32.DLL	DLL du sous système Win32 Exportent les points d'entrée Win32

Démarrage de XP (1)

- Les deux premiers processus n'en sont pas !!
  - Ne correspondent pas à un EXE
  - Pas d'espace d'adressage en mode user

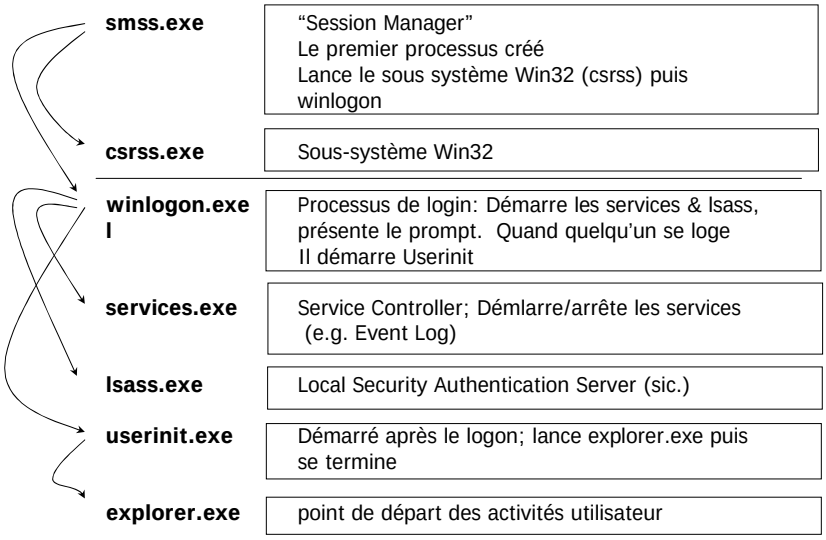
(Idle)

Process id = 0
Fait partie de l'image chargée
Héberge les threads inactifs
Souvent désigné comme "System Process"

(System)

Process id = 8
Fait partie de l'image chargée
Héberge les threads du kernel
Le Thread 0 démarre le premier "vrail" processus, en lançant smss.exe (Session Manager)

Démarrage de XP (2)



Mécanismes internes du noyau XP

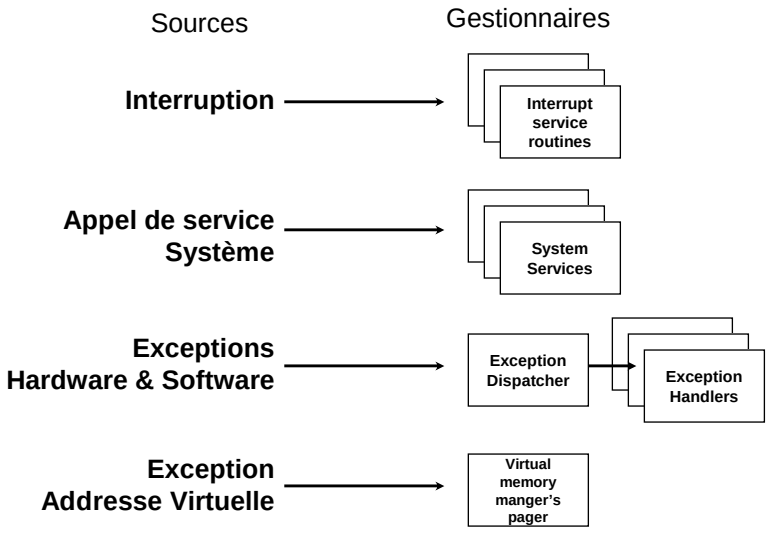
Appel de routines “kernel”

- Le code tourne en mode “kernel” pour une des trois raisons suivantes :
  - Appel depuis le mode “user”
    - Via le gestionnaire de services système (ci-avant)
  - Interruption depuis un périphérique
    - Les IT sont exécutées en mode “kernel”
    - Le gestionnaire d’IT de XP lance l’ISR (interrupt service routine)
  - Threads dédiés en mode “kernel”
    - Quelques threads du système restent toujours en mode “kernel” (la plupart sont dans le processus “System”)

Gestionnaire de “Trap”

- Les Interruptions et les exceptions déroutente processeur vers du code extérieur au chemin du programme en cours
- Peuvent être matérielles ou logicielles
- *Trap*
  - Mécanisme pour interrompre un thread en cours
  - Transfert le contrôle à une zone fixée de l’OS
- Windows XP
  - La CPU transfère le contrôle à un *"gestionnaire de trap”*
    - Puis donne le contrôle à d’autres fonctions de traitement
    - Ex: interruption hard – transfère le contrôle à l’ISR fournie par le pilote de périphérique

Gestion des “Trap”



Interruptions & Exceptions

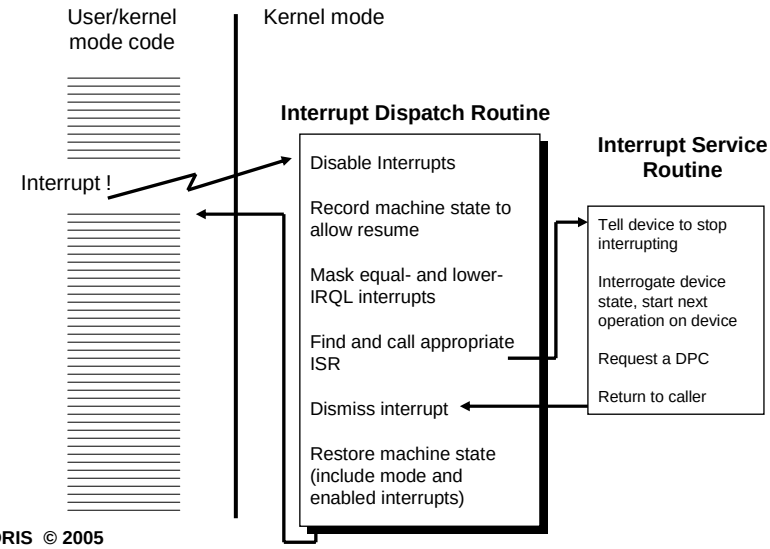
- Interruption
  - Asynchrones (arrive n’importe quand)
  - Générées par les périphériques d’E/S, les horloges CPU, les timers, etc.
- Exception
  - Synchrone
  - Resulte de l’exécution d’une instruction spécifique (INT)
  - Exemples
    - Memory Access Violation, Divide By Zero
- Les deux peuvent provenir du Hardware & Software
  - Exceptions: Bus Error, Divide-by-Zero
  - Interruptions: I/O Device, Software Interrupts (DPCs)
- Lorsqu’on a une interrupt/exception
  - La CPU stocke son état pour revenir au point de départ et continuer l’exécution ultérieurement

Gestionnaire d'interruptions (1)

- Les IT permettent d'optimiser l'usage CPU
  - Signale les threads pour effectuer les transferts avec le périphériques
    - D'autres travaux peuvent être exécutés pendant que le périphérique effectue ses transferts
    - Un périphérique notifie la CPU quand il en a besoin
- La souris, le Clavier, Les disques sont gérés par des interruptions
- Les pilotes de périphériques fournissent des ISR pour les traitement d'interruptions
- Le noyau fournit la gestion d'interruption pour les autres types

THEORIS © 2005 39

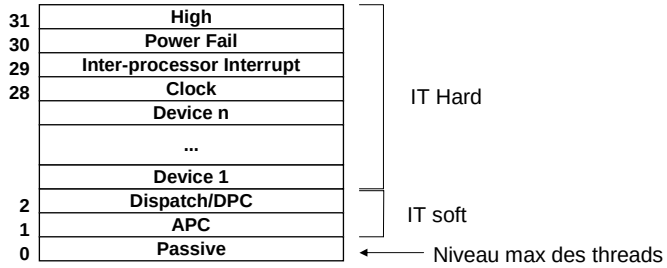
Gestionnaire d'interruptions (2)



THEORIS © 2005 40

Priorités des interruptions

- Windows XP a on propre schéma d'interruptions
  - IRQL = Interrupt Request Level (0 à 31)
- Des sources d'IT différentes ont de IRQL différentes (IT <> IRQ)
- Les interruptions sont traitées par priorités
  - Une IT "haute" pré-empte une IT "basse"
- Le traitement d'une IT relève le niveau d'IRQL à celle de l'IT en cours
  - Masquage des IT avec un IRQL de niveau inférieur ou égal

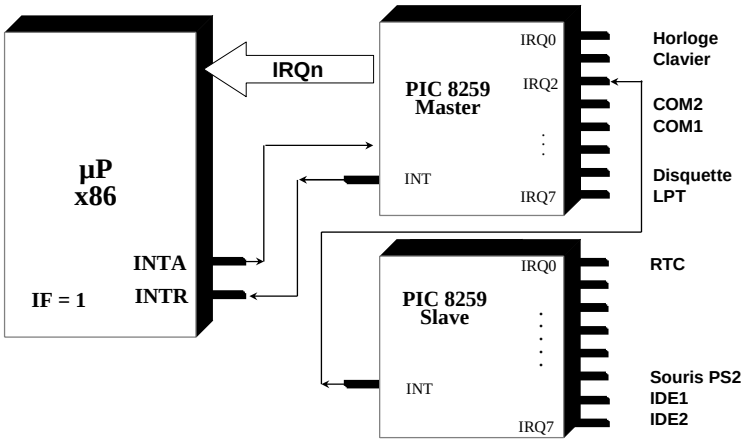


Interruptions logicielles

- Windows XP génère lui même des IT
- Lorsque le niveau d'IRQL est élevé, rien ne peut s'exécuter à niveau inférieur ou égal
  - Cela peut provoquer un retard dans la réponse à des évènements critiques
  - Windows XP évite cette siutuation en exécutanbt un maximum de code au plus bas niveau d'IRQL
- Deferred Procedure Calls (DPCs)
  - Utilisé pour reporter les traitements d'un niveau élevé (hard) vers un niveau préemptible (dispatch)
  - Les DPC servent à retarder les traitement moins urgents :
    - Les pilotes d'E/S terminent en DPC
  - Les DPC tournent quand IRQL atteint le niveau "dispatch"

Interruptions du PC

- Architecture matérielle des interruptions dans l'architecture PC-AT :
  - Broche INTR des interruptions masquables de l'UC
  - Deux contrôleurs d'interruptions 8259 en cascade →15 niveaux d'IT



IT matérielles sous XP

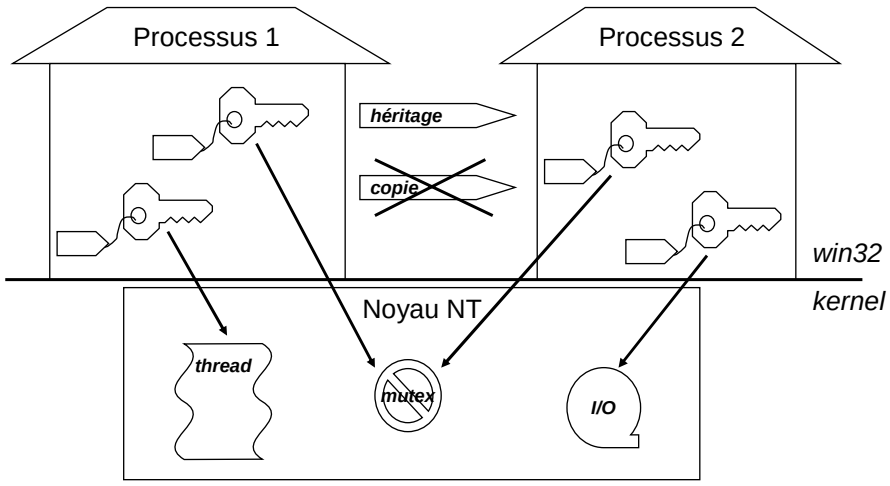
- Le périphérique présente une IT au contrôleur
- Le contrôleur interromp la CPU
- La CPU interroge le contrôleur pour avoir la source IRQ (interrupt request)
- Si l'IRQL courant est inférieure
- Appel du gestionnaire deTrap
- Le gestionnaire de Trap sauve le contexte (dont l'IRQL courante), dévalide les IT, entre dans le gestionnaire d'IT

IT matérielles sous XP

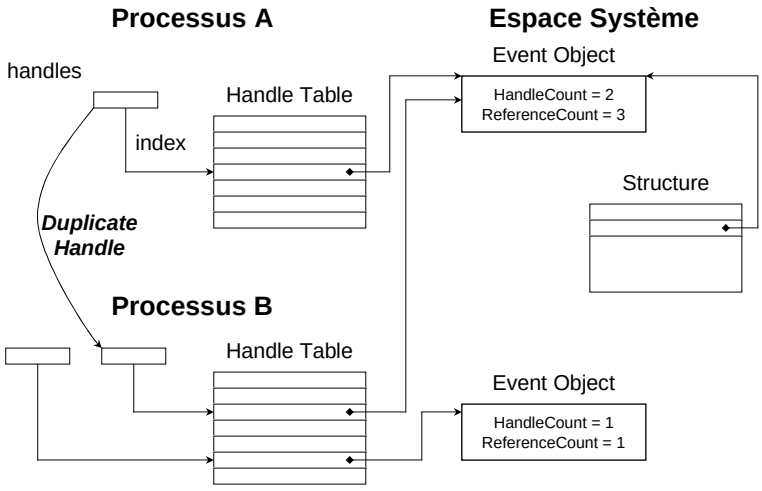
- Le gestionnaire d'IT relève le niveau d'IRQL puis alide les IT
- La correspondance est faite entre L'IRQ et le numéro d'interruption dans l'IDT (*Interrupt Dispatch Table*)
  - L'IDT est utilisée pour appeler la bonne routine d'interruption
  - L'IDT liste les pointeurs de routines "kernel" pour chaque IT
- La routine d'IT est appelée
- En sortie de la routine d'IT, l'IRQL est remis au niveau original et le contexte est rechargé

Handles
Processus & Threads

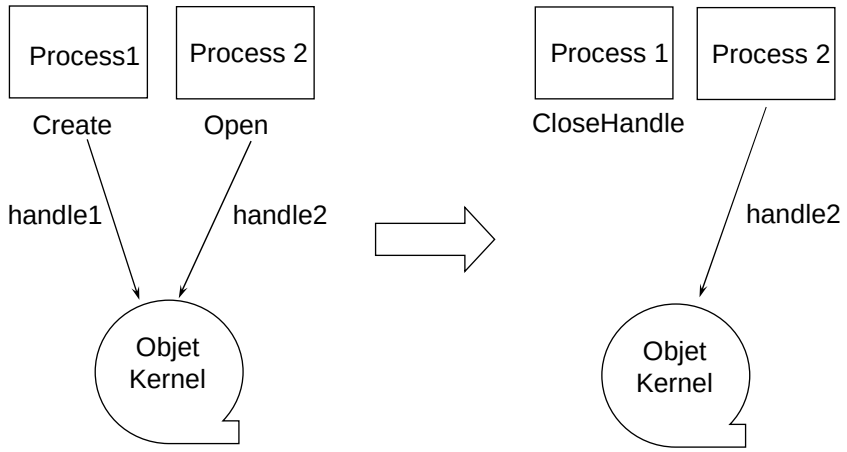
Objets kernel et handles



Comptage des références

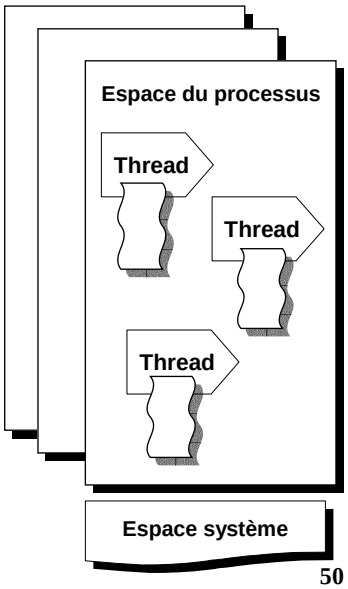


Existence d'un objet kernel



Processus, Threads et Jobs

- Un Processus
  - Représente une instance d'un program en cours d'exécution
  - Le lancement d'une application crée un processus
- Un Thread
  - Un contexte d'exécution au sein d'un processus
  - Tous le threads d'un même processus partagent le même espace d'adressage
- Un Job
  - Permet de gérer un groupe de processus de manière unifiée



Le Processus Win32

- Chaque processus a :
  - Un espace d'adresage virtuel
    - Les processus ne peuvent pas se corrompre entre eux
  - Un "Working set"
    - Zone de mémoire physique attribuée
  - Des jetons d'accès (tokens)
    - Inclus les identifiants de sécurité
  - Une table de Handles
- Commun à tous les threads du processus – mais separé et protégé entre processus

Le Thread Win32

- Chaque Thread a :
  - Une pile (Stack)
  - Un état d'ordonnancement (Wait, Ready, Running, etc..)
  - Une priorité d'ordonnancement
  - Un état courant ("user" ou "kernel")
  - Un contexte sauvé s'il ne tourne pas

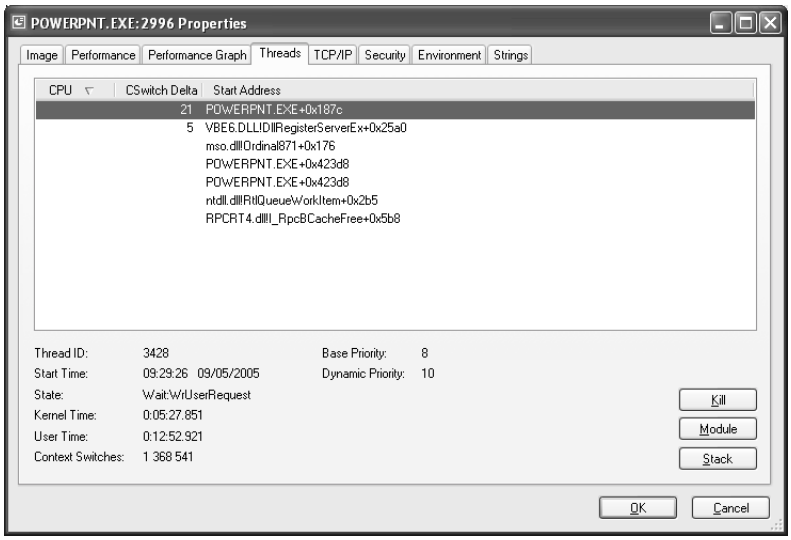
API Win32 Processus

- *CreateProcess*
- *OpenProcess*
- *GetCurrentProcessId* – donne un ID global
- *GetCurrentProcess* – donne le handle
- *ExitProcess*
- *TerminateProcess* – pas de notification aux DLL
- *Get/SetProcessShutdownParameters*
- *GetExitCodeProcess*
- *GetProcessTimes*
- *GetStartupInfo*

API Win32 Thread

- *CreateThread*
- *CreateRemoteThread* – Crée un thread dans un autre processus
- *GetCurrentThreadId* – donne un ID global
- *ExitThread* – Termine normalement
- *TerminateThread* – pas de notification aux DLL
- *GetExitCodeThread* – return post mortem
- *GetThreadTimes* – donne les temps d'exécution
- *Get/SetThreadContext* – donne ou change les registres

Les Threads d'un processus



Ordonnancement XP

- Les threads demandent la CPU
  - Qui tourne ?
- DU point de vue Win32
  - Chaque processus a une classe de priorité :
    - Idle, Below Normal, Normal, Above Normal, High, Realtime
  - Chaque Threads a une priorité relative dans la classe
    - Idle, Lowest, Below_Normal, Normal, Above_Normal, Highest, Time_Critical

Priorités Win32

THREAD \ PROCESSUS	Real Time	High	Above Normal	Normal	Below Normal	Idle
Time-critical	31	15	15	15	15	15
Highest	26	15	12	10	8	6
Above-normal	25	14	11	9	7	5
Normal	24	13	10	8	6	4
Below-Normal	23	12	9	7	5	3
Lowest	22	11	8	6	4	2
Idle	16	1	1	1	1	1

Ordonnancement des threads

- Strictement défini par la priorité
  - 32 files (FIFO) de threads prêts (*ready*)
    - Niveau de priorité
    - Le files sont communes à toutes les CPU
  - Quan:d un thread devient prêt :
    - Il tourne immédiatment, ou bien :
    - Il est inséré en queue de file à son niveau de priorité courante
  - Sur un système mono-CPU c'est toujours le thread le plus prioritaire qui tourne
- Partage de temps en round-robin dans un niveau de priorité donné

CPU multiples

- N CPUs → les N threads de priorité haute tournent (sauf *Affinity*)
- Pas d'ordonnancement prenant en compte les processus, seulement les threads
- Essaie de conserver un thread sur la même CPU

API Win32 d'ordonnancement

- *Get/SetPriorityClass*
- *Get/SetThreadPriority* – relatif
- *Get/SetProcessAffinityMask*
- *SetThreadAffinityMask* – doit être un subset du processus
- *SetThreadIdealProcessor* – CPU *préféré*
- *Get/SetProcessPriorityBoost*
- *Suspend/ResumeThread*

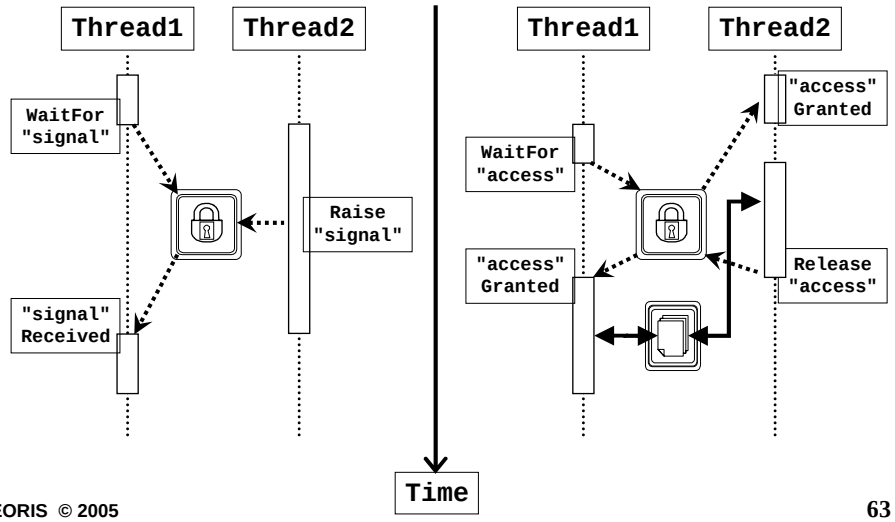
Etats d'un thread

- *Init* (0) – Thread "en construction"
- *Ready* (1) – thread elligible pour l'ordonnanceur
- *Running* (2)
- *Standby* (3) – thread sélectionné
- *Terminate* (4) – code terminé, en attente de disparition
- *Waiting* (5) – Attente sur un handle
- *Transition* (6) – test de famine

Coup de pouce (boost)

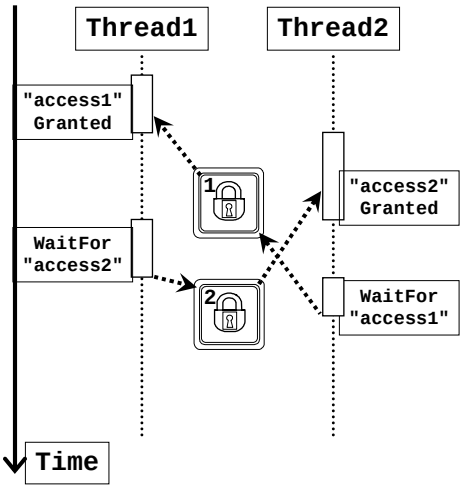
- Le coup de pouce de priorité a lieu en sortie d'un wait
  - Arrive quand un wait (I/O) est résolu
    - Périphériques lents / wait long = gros coup de pouce
    - Périphériques rapides / wait court = petit coup de pouce
  - Le boost est appliqué sur la priorité de base du thread
    - N'excède jamais 15
  - Garde les I/O actives
- Après le coup de pouce:
  - La priorité descend de un niveau par quantum, jusqu'au niveau de base

Signalisation vs Protection



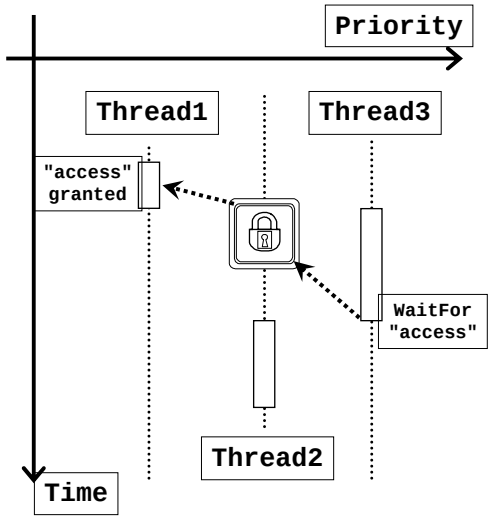
Deadlock

- Causes :
  - Mauvaise conception
  - Portage depuis un modèle coopératif
- Effets :
  - Blocage application
- Solutions :
  - Revoir la conception



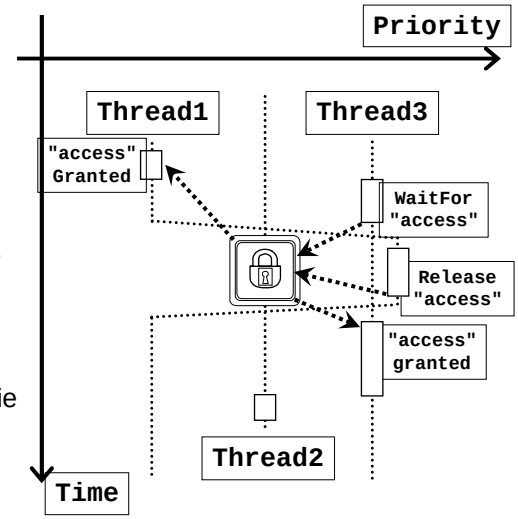
Inversion de priorité – Pourquoi ?

- Trois threads de priorités différentes
- Un objet de synchronisation :
  - Le plus faible le détient
  - Le plus fort l'attend
  - L'intermédiaire masque le plus faible



Inversion de priorité – Comment ?

- Mécanisme :
  - Le plus faible devient le plus fort
  - Il libère l'objet de synchronisation
  - Il redevient le plus faible
- Impacte :
  - Ordonnancement
  - Contraire à la philosophie TR



Composants de synchronisation

- Les sections critiques
  - Réservée à la synchronisation au sein du même processus.
- Les objets du noyau
  - L'état signalé/non signalé d'un objet du noyau
  - Les objets de synchronisation
    - Mutex, Sémaphore, Evénement
  - Les autres objets du noyau utilisables
    - Processus, Thread, Fichier, ...
- Les primitives de synchronisation
    - WaitForSingleObject, WaitForMultipleObjects, ...

Primitives d'attente WaitFor

- WaitForSingleObject
  - L'attente s'effectue sur un seul objet à la fois
  - L'attente peut être limitée dans le temps ou infinie
  - L'attente d'un mutex abandonné est signalée
- WaitForMultipleObjects
  - Attente de plusieurs événements simultanés
  - Attente d'un événement parmi plusieurs
- SignalObjectAndWait
  - Signale un objet (événement, mutex ou sémaphore) puis attend qu'un autre objet soit signalé

D’autres synchronisations

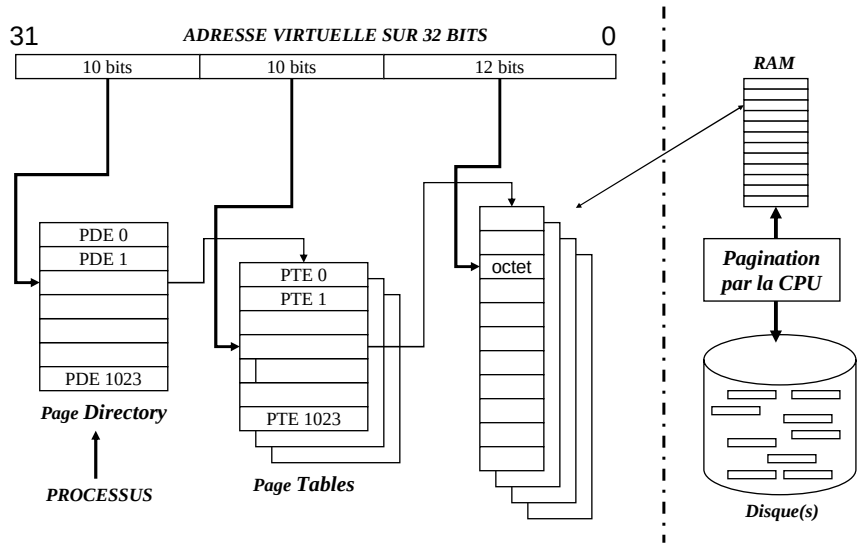
- Les processus et threads
  - Ces 2 types objets sont signalés lorsqu’ils sont terminés
  - SuspendThread, ResumeThread et Sleep
    - Exécuter un thread suspendu à la création
    - Les suspensions peuvent être imbriquées
    - Sleep(0) permet au thread d’abandonner sa plage de temps et de se suspendre pour un durée déterminée.
- Les fichiers
  - A l’issue d’E/S asynchrones il est possible d’utiliser les handles de fichier ou d’événement pour se resynchroniser

Gestion de la Mémoire XP

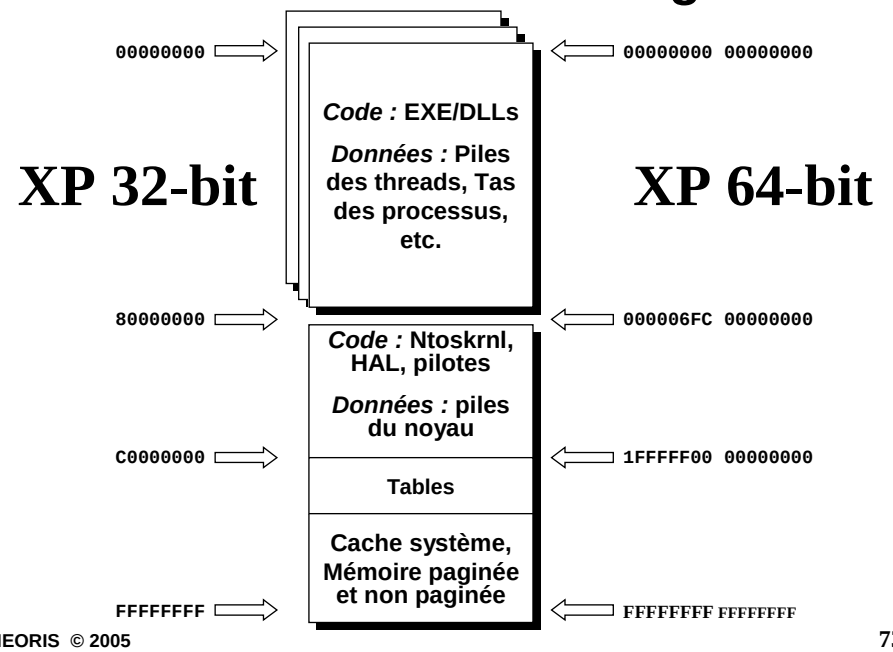
Accès à la mémoire

- Mapping mémoire
  - 32 bits d'adressage = 4Go pour chaque processus
  - une mémoire segmentée et protégée
- Mémoire virtuelle: réservation
  - Réserver une partie de l'espace d'adressage du processus.
- Mémoire physique: allocation
  - S'approprier une partie de l'espace physique du système.
  - Cette mémoire est très loin d'atteindre 4Go par processus!
- Les tas Win32 (heaps)

Mécanisme de pagination



La mémoire – Adressage



THEORIS © 2005 73

Fichier de swap

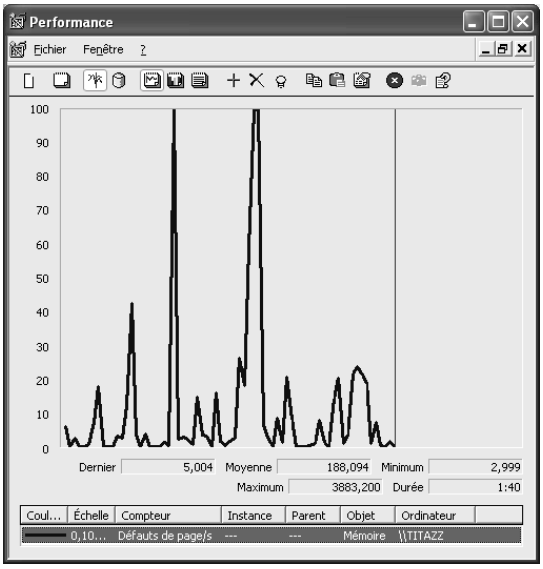
- Utilisé uniquement quand la demande excède l'offre
- La taile dépend du contexte d'utilisation de la machine
  - Le disque ne coûte pas cher
- Il est souhaitable d'avoir un fichier contigu
  - C'est ce qui est fait à la création
  - Sinon il faut défragmenter
- Quand l'espace disque vient à manquer :
  - 1. "System running low on virtual memory"
    - Première fois : avant l'extension du swap
    - Deuxième fois: Quan les octets en "commit" arrivent à la limite
  - 2. "System out of virtual memory"
    - Plus de place sur le disque

THEORIS © 2005 74

Défauts de Pages

- Exception levée quand on accède à une page qui n'est plus en RAM (`*MyPtr = 33 ;`)
- Le système va lire le contenu de la page dans le fichier de swap :
  - La page Physique est allouée
  - Le bloc est lu dans la page
  - L'entrée dans la table des pages est mise à jour
  - L'exception est résolue
  - La CPU re-exécute l'instruction
- La page est maintenant dans le "working set" du processus
- Les Pages ne viennent en mémoire que suite à des "Page faults"

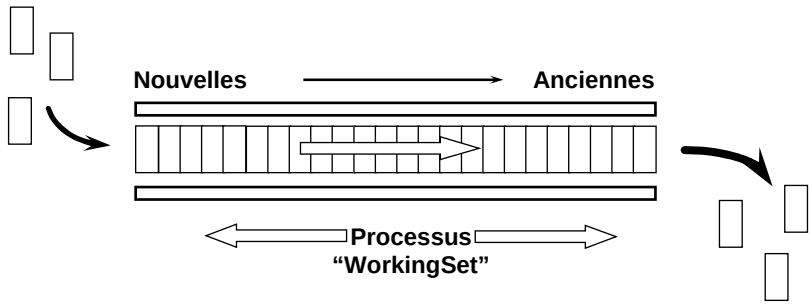
Défauts de pages



Working Set

- Working set = toute les pages physiques “propriété” d’un processus
  - Toute celle qu’il peut voir sans provoquer un “Page Fault”
- Un processus démarre toujours avec un Working Set vide
- Si la limite est atteinte, il faut libérer une page pour en avoir une nouvelle
- Le remplacement des pages se fait selon in ordre “FIFO de modification ”
  - Windows XP sur un x86 mono-CPU utilise un algorithme “least recently accessed”

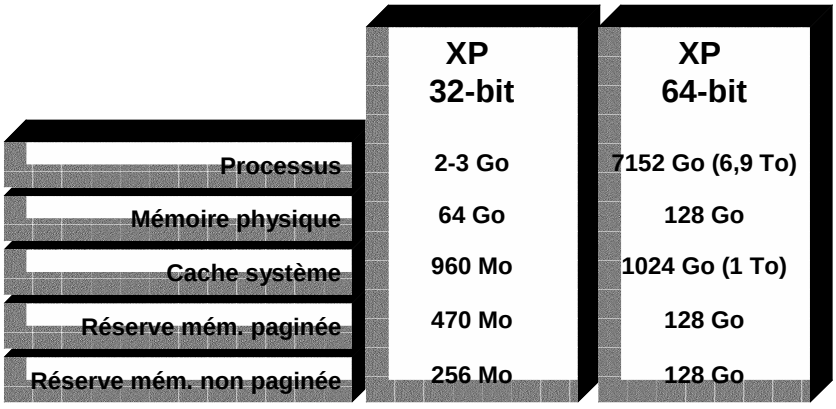
Le Working Set d’un processus



Bilan de la mémoire physique

- “working sets” des processus
  - Les pages partagées (shared) sont comptées dans le WS de chaque processus
    - Total > Physique !!
- Code système non paginé
  - NTOSKNL + drivers
- Données non paginées (NONPAGED-POOL)
- Listes et tables de pages
- Mémoire système paginable mais restant résidente

La mémoire - Limites



Accès Win32 à la mémoire

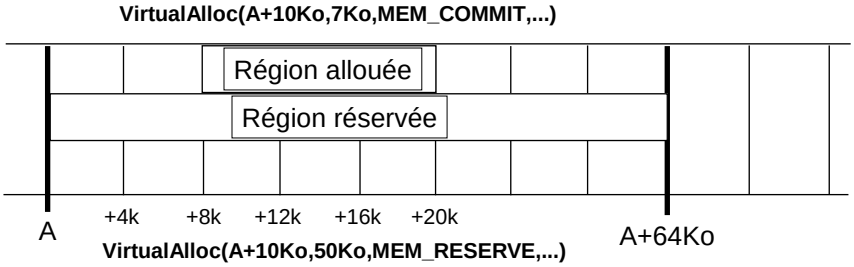
- Granularité de la mémoire
  - Une région est toujours alignée sur un multiple de l'unité d'allocation (64Ko)
  - La taille d'une région correspond toujours à un multiple pair de pages systèmes (4Ko ou 8Ko)
- Attributs de protection
  - Chaque page physique est dotée d'attributs d'accès.
  - Windows 95 limite le nombre d'attributs gérés.
  - Les attributs ne sont pas tous gérés sur tous les processeurs.

Mémoire virtuelle

- Informations sur la mémoire
  - GetSystemInfo
    - dwPageSize: 4Ko ou 8Ko (Alpha)
    - dwAllocationGranularity: 64Ko
  - GlobalMemoryStatus, VirtualQuery
- VirtualAlloc
  - MEM_RESERVE: réservation de mémoire virtuelle
  - L'offset est aligné sur l'unité d'allocation et la taille de la région est multiple de la taille de la page système
- VirtualFree
  - MEM_RELEASE: libération de mémoire virtuelle
  - La libération ne peut pas être partielle

Mémoire physique

- VirtualAlloc
  - MEM_COMMIT: allocation de mémoire physique
  - Allocation d' un nombre entier de pages
  - Positionne le véritable attribut de protection



Tas Win32 (heaps)

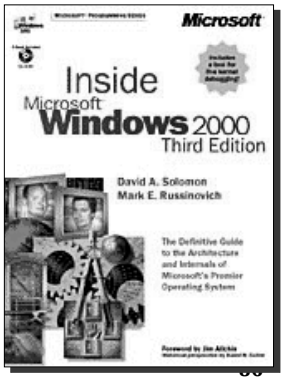
- Allocation simplifiée de mémoire physique
  - Win32 gère la réservation et l'allocation de mémoire
  - Win32 nous masque la notion de page
  - malloc() utilise le tas
- Heap et multitâches
  - Sérialisation automatique des demandes
  - Avec HEAP_NO_SERIALIZE le système ne protège plus le heap contre des accès concurrents => danger
- Un processus peut avoir plusieurs Heaps

Données locale au thread

- Données de l'instance «thread»
  - Le mécanisme de «Thread Local Storage» permet d'associer des données à une instance de thread.
  - Un minimum assuré de 64 mots de 32 bits
- Un simple tableau à accès réservé
  - TlsAlloc: réserver un index au niveau du processus pour tous les threads existants et à venir!!!
  - TlsSetValue: Mémoriser une valeur à l'index
  - TlsGetValue: Récupérer une valeur à l'index
  - TlsFree: libérer un index pour tous les threads du processus

Pour plus d'Info...

- http://www.sysinternals.com
- http://www.winnetmag.com/windowsnt20002003faq/
- http://www.smidgeonsoft.com/
- http://billsway.com
- http://www.ntfaqfr.com/
- Un livre de chevet…



www.theoris.fr