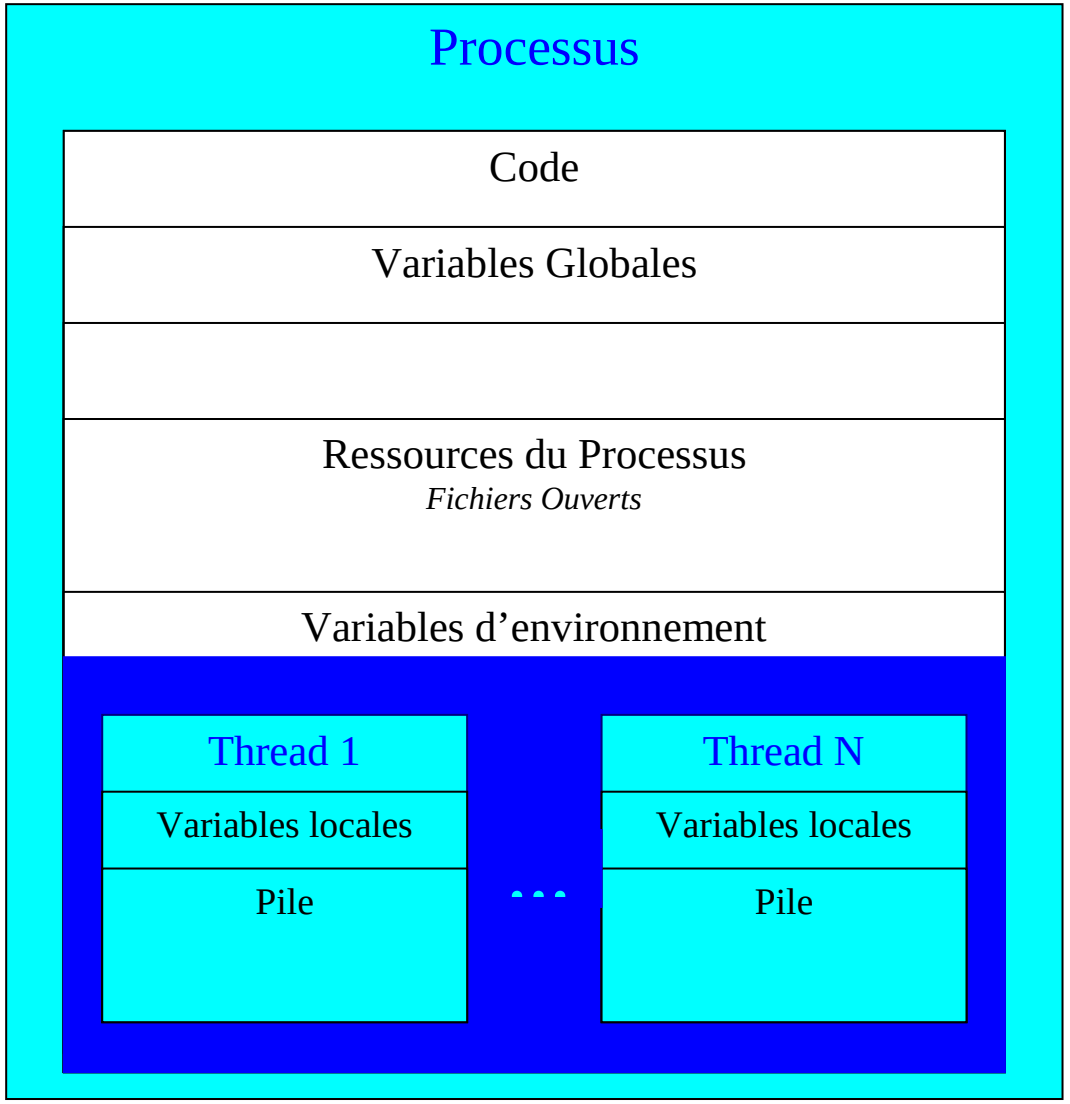
12. Processus sous NT

12.1 Processus sous NT



Processus sous NT (suite)

- Un processus est une instance d'exécution d'une application
- Un processus est composé de :
  - De code chargé en mémoire depuis des fichiers exécutables et des DLLs
  - 4GB d'espace d'adressage
  - Ressources telles que ses threads, fichiers, mémoire dynamique ..etc.
- Le processus n'exécute pas de code, c'est un espace mémoire de code et de ressources.
- Le code du processus est exécuté par des threads
- Il n'y a pas de relations père/fils entre les processus

12.2 Gestion de Processus

- CreateProcess
- Structure STARTUPINFO
- Structure PROCESS_INFORMATION
- OpenProcess
- TerminateProcess
- GetCurrentProcess
- GetCurrentProcessId
- GetProcessTimes
- WaitForSingleObject, WaitForMultipleObjects

12.3 Création de Processus

- La fonction **CreateProcess** crée un nouveau processus et sa première thread. Le nouveau processus exécute le fichier exécutable spécifié .
- Syntaxe :

```
BOOL CreateProcess( LPCTSTR lpApplicationName, LPTSTR
lpCommandLine, LPSECURITY_ATTRIBUTES lpProcessAttributes,
LPSECURITY_ATTRIBUTES lpThreadAttributes, BOOL bInheritHandles,
DWORD dwCreationFlags, LPVOID lpEnvironment, LPCTSTR
lpCurrentDirectory, LPSTARTUPINFO lpStartupInfo,
LPPROCESS_INFORMATION lpProcessInformation);
```

- Arguments :

- *LPCTSTR lpApplicationName, pointeur sur le nom du fichier exécutable (si NULL, utilise alors lpCommandLine)*
- *LPTSTR lpCommandLine, pointeur sur la chaîne ligne de commande*
- *LPSECURITY_ATTRIBUTES lpProcessAttributes, pointeur sur les attributs de sécurité du processus (si NULL utilise les attributs de sécurité du processus appelant)*

- *LPSECURITY_ATTRIBUTES lpThreadAttributes, pointeur sur les attributs de sécurité du thread (si NULL utilise les attributs de sécurité du processus appelant)*
- *BOOL bInheritHandles, flag d'héritage des handles (si TRUE le nouveau processus hérite des Handles du processus appelant, FALSE sinon)*
- *DWORD dwCreationFlags, flags de création et de classe de priorité du processus (CREATE_DEFAULT_ERROR_MODE, CREATE_NEW_CONSOLE CREATE_NEW_PROCESS_GROUP CREATE_SEPARATE_WOW_VDM CREATE_SHARED_WOW_VDM CREATE_SUSPENDED CREATE_UNICODE_ENVIRONMENT DEBUG_PROCESS DEBUG_ONLY_THIS_PROCESS DETACHED_PROCESS ..) & (HIGH_PRIORITY_CLASS / IDLE_PRIORITY_CLASS / NORMAL_PRIORITY_CLASS / REALTIME_PRIORITY_CLASS)*
- *LPVOID lpEnvironment, pointeur sur le nouveau block d'environnement (si NULL le nouveau processus utilise l'environnement du processus appelant)*
- *LPCTSTR lpCurrentDirectory, pointeur sur le nom du répertoire courant (si NULL le nouveau processus utilise le même répertoire courant que le processus appelant)*
- *LPSTARTUPINFO lpStartupInfo, pointeur sur STARTUPINFO*

- *LPPROCESS_INFORMATION lpProcessInformation, pointeur sur PROCESS_INFORMATION*

♦ Structure STARTUPINFO :

- *typedef struct _STARTUPINFO {*
- *DWORD cb; Taille de la structure en octets*
- *LPTSTR lpReserved; non documenté, à NULL*
- *LPTSTR lpDesktop; si NULL hérite du bureau et de la station window du processus appelant.*
- *LPTSTR lpTitle; barre de titre de la nouvelle fenêtre console*
- *DWORD dwX; déplacement en X depuis le coin gauche supérieur de l 'écran pour l'ouverture de la nouvelle fenêtre*
- *DWORD dwY; déplacement en Y depuis le coin gauche supérieur de l 'écran pour l'ouverture de la nouvelle fenêtre*
- *DWORD dwXSize; Taille sur X de la nouvelle fenêtre*
- *DWORD dwYSize; Taille sur Y de la nouvelle fenêtre*
- *DWORD dwXCountChars; Taille du buffer de caractères colonnes*

- *DWORD dwYCountChars; Taille du buffer de caractères lignes*
- *DWORD dwFillAttribute; Couleurs de fond et de texte. Cette valeur peut être une combinaison des valeurs suivantes : FOREGROUND_BLUE, FOREGROUND_GREEN, FOREGROUND_RED, FOREGROUND_INTENSITY, BACKGROUND_BLUE, BACKGROUND_GREEN, BACKGROUND_RED, et BACKGROUND_INTENSITY. Par exemple, la combinaison suivante produit un texte rouge sur un fond blanc: FOREGROUND_RED | BACKGROUND_RED | BACKGROUND_GREEN | BACKGROUND_BLUE*
- *DWORD dwFlags; définit quels champs de cette structure sont utilisés à la création du processus. C'est une combinaison des valeurs suivantes : STARTF_USESHOWWINDOW (wShowWindow), STARTF_USEPOSITION (dwX et dwY) STARTF_USESIZE (dwXSize et dwYSize) STARTF_USECOUNTCHARS (dwXCountChars et dwYCountChars) STARTF_USEFILLATTRIBUTE (dwFillAttribute) ...)*
- *WORD wShowWindow; une des valeurs constantes de WINUSER.H*
- *WORD cbReserved2; non documenté, à NULL*
- *LPBYTE lpReserved2; non documenté, à NULL*
- *HANDLE hStdInput; Handle d'entrées standard*
- *HANDLE hStdOutput; Handle sorties standard*

- *HANDLE hStdError; Handle d'erreurs standard*
- *} STARTUPINFO ;*

♦ La fonction GetStartupInfo récupère le contenu de la structure STARTUPINFO du processus courant (à utiliser pour récupérer la structure du processus appelant).

- *VOID GetStartupInfo(LPSTARTUPINFO lpStartupInfo);*

♦ Structure PROCESS_INFORMATION :

- *typedef struct _PROCESS_INFORMATION {*
- *HANDLE hProcess; Handle sur le nouveau processus créé*
- *HANDLE hThread; Handle sur le premier thread du processus créé*
- *DWORD dwProcessId; identificateur global du processus créé*
- *DWORD dwThreadId; identificateur global du premier thread du processus créé*
- *} PROCESS_INFORMATION;*

12.4 Exemples de Création de Processus

♦ Création de processus sans arguments

- *CreateProcess(_T("D:\\Users\\tigli\\Cours\\Win32\\TD2_PREPA\\processus2\\procid.exe"),NULL, NULL, NULL, FALSE, CREATE_NEW_CONSOLE|CREATE_SUSPENDED, NULL, NULL, &si, pi)*

♦ Création de processus avec arguments

- *CreateProcess(NULL, _T("D:\\Users\\tigli\\Cours\\Win32\\TD2_PREPA\\processus2\\procid.exe -b 2000"), NULL, NULL, FALSE, CREATE_NEW_CONSOLE, NULL, NULL, &si, pi)*

♦ Création de processus sans console

- *CreateProcess(NULL, _T("D:\\Users\\tigli\\Cours\\Win32\\TD2_PREPA\\processus2\\procid.exe -b 2000"), NULL, NULL, FALSE, DETACHED_PROCESS, NULL, NULL, &si, pi)*

♦ Création de processus avec premier thread en attente

- *CreateProcess(NULL, _T("D:\\Users\\tigli\\Cours\\Win32\\TD2_PREPA\\processus2\\procid.exe -b 2000"), NULL, NULL, FALSE, CREATE_SUSPENDED, NULL, NULL, &si, pi)*

12.5 OpenProcess

- Retourne un Handle sur un objet processus existant, NULL sinon (utile pour gérer l'attente d'un processus par un autre)
- Syntaxe

  - *HANDLE OpenProcess(DWORD dwDesiredAccess, BOOL bInheritHandle, DWORD dwProcessId);*

- Arguments

  - *DWORD dwDesiredAccess, Flag d' accès (exemples : PROCESS_ALL_ACCESS, PROCESS_TERMINATE ...)*
  - *BOOL bInheritHandle, Flag d'héritage des Handles*
  - *DWORD dwProcessId, identificateur du processus*

12.6 Terminaison de Processus

12.6.1 TerminateProcess

- Termine le processus spécifié et tous ses threads, retourne une valeur non nulle, 0 en cas d'échec.
- Syntaxe

  - *BOOL TerminateProcess(HANDLE hProcess, UINT uExitCode);*

- Arguments

  - *HANDLE hProcess, identifie le processus à terminer*
  - *UINT uExitCode, code de retour du processus*

- Pb : Les DLLs attachées au processus ne sont pas gérées

12.6.2 ExitProcess

- Termine le processus courant et tous ses threads, ne retourne pas de valeur.
- Syntaxe

  - *VOID ExitProcess(UINT uExitCode);*

- Arguments

  - *UINT uExitCode, code de retour du processus*

- Les DLLs attachées au processus sont gérées (préférable à TerminateProcess)

12.7 GetExitCodeProcess

- Récupère le code de retour du processus spécifié
- Syntaxe

  - *BOOL GetExitCodeProcess(HANDLE hProcess, LPDWORD lpExitCode);*

- Arguments

  - *HANDLE hProcess, Handle du processus spécifié*
  - *LPDWORD lpExitCode, pointeur sur le code de retour du processus*

12.8 Identité de Processus

- GetCurrentProcess
- Retourne un handle sur le processus courant

  - *HANDLE GetCurrentProcess(VOID)*

- GetCurrentProcessId
- Retourne l'identificateur du processus courant

  - *DWORD GetCurrentProcessId(VOID)*

12.9 Temps d'exécution de Processus

- GetProcessTimes, temps d'exécution d'un processus (écoulé, en mode noyau, en mode utilisateur)

- syntaxe

  - *BOOL GetProcessTimes(HANDLE hProcess, LPFILETIME lpCreationTime, LPFILETIME lpExitTime, LPFILETIME lpKernelTime, LPFILETIME lpUserTime) ;*

- Arguments

  - *HANDLE hProcess, Handle sur le processus*
  - *LPFILETIME lpCreationTime, pointeur sur la date de création du processus (format FileTime)*
  - *LPFILETIME lpExitTime, pointeur sur la date de fin du processus (format FileTime)*
  - *LPFILETIME lpKernelTime, pointeur sur le temps ou le processus est resté en mode noyau (format FileTime)*

- *LPFILETIME lpUserTime, pointeur sur le temps ou le processus est resté en mode utilisateur (format FileTime)*

♦ Fonctions de conversion du temps (déjà vues)

- *BOOL **FileTimeToSystemTime** (CONST FILETIME *lpFileTime, LPSYSTEMTIME lpSystemTime)*
- *BOOL **SystemTimeToFileTime** (CONST SYSTEMTIME *lpSystemTime, LPFILETIME lpFileTime) est la fonction réciproque de FileTimeToSystemTime.*

12.10 Variables d'environnement

12.10.1 GetEnvironmentVariable

- Récupère la valeur d'une variable d'environnement du processus appelant, retourne le nombre de caractères de la valeur de la variable (sans le caractère nul de fin de chaîne). Si la variable n'est pas trouvée la valeur de retour est zéro.
- Syntaxe

  - *DWORD GetEnvironmentVariable(LPCTSTR lpName, LPTSTR lpBuffer, DWORD nSize);*

- Arguments

  - *LPCTSTR lpName, pointe sur le nom de la variable d'environnement*
  - *LPTSTR lpBuffer, pointe sur le buffer recevant la valeur de la variable d'environnement*
  - *DWORD nSize, taille en caractère de la valeur dans lpBuffer*

12.10.2 SetEnvironmentVariable

- Fixe la valeur d'une variable d'environnement du processus appelant, retourne une valeur nulle en cas d'échec
- Syntaxe

  - *DWORD SetEnvironmentVariable(LPCTSTR lpName, LPCTSTR lpValue);*

- Arguments

  - *LPCTSTR lpName, pointe sur le nom de la variable d'environnement*
  - *LPCTSTR lpValue, pointe sur la valeur de la variable d'environnement (si NULL alors la variable d'environnement correspondante est effacée)*

12.11 Attente de terminaison de processus

12.11.1 WaitForSingleObject

- La fonction WaitForSingleObject teste l'état courant de l'objet spécifié. Si l'état de l'objet est « nonsignaled », le thread appelant se met en attente active. Le thread prend peu de temps CPU dans cet état d'attente d'un état « signaled » d'un objet ou d'un timeout éventuel. Retourne WAIT_FAILED en cas d'échec, WAIT_TIMEOUT sur timeout
- Syntaxe

  - *DWORD WaitForSingleObject(HANDLE hHandle, DWORD dwMilliseconds);*

- Arguments

  - *HANDLE hHandle, Handle de l'objet à attendre*
  - *DWORD dwMilliseconds, Timeout en millisecondes (si la valeur est nulle le test est non-bloquant, sinon il s'agit d'un timeout en millisecondes, enfin INFINITE pour une attente infinie).*
  -

12.11.2 WaitForMultipleObjects

- La fonction WaitForMultipleObject teste l'état courant de l'objet spécifié. Si l'état de l'objet est « nonsignaled », le thread appelant se met en attente active. Le thread prend peu de temps CPU dans cet état d'attente d'un état « signaled » d'un objet ou d'un timeout éventuel. Retourne WAIT_FAILED en cas d'échec, WAIT_TIMEOUT sur timeout
- Syntaxe

  - *DWORD WaitForMultipleObjects(WORD nCount, CONST HANDLE *lpHandles, BOOL bWaitAll, DWORD dwMilliseconds);*

- Arguments

  - *WORD nCount, nombre d'objets handle dans le tableau lpHanldes (le maximum est MAXIMUM_WAIT_OBJECTS)*
  - *CONST HANDLE *lpHandles, pointe sur un tableau d'objets handles*

- *BOOL bWaitAll, Si TRUE attend que tous les objets du tableau soient dans l'état signaled, Si FALSE attend qu'un des objets du tableau soit dans l'état signaled*
- *DWORD dwMilliseconds, Timeout en millisecondes (si la valeur est nulle le test est non-bloquant, sinon il s'agit d'un timeout en millisecondes, enfin INFINITE pour une attente infinie.)*

♦ Exemple :

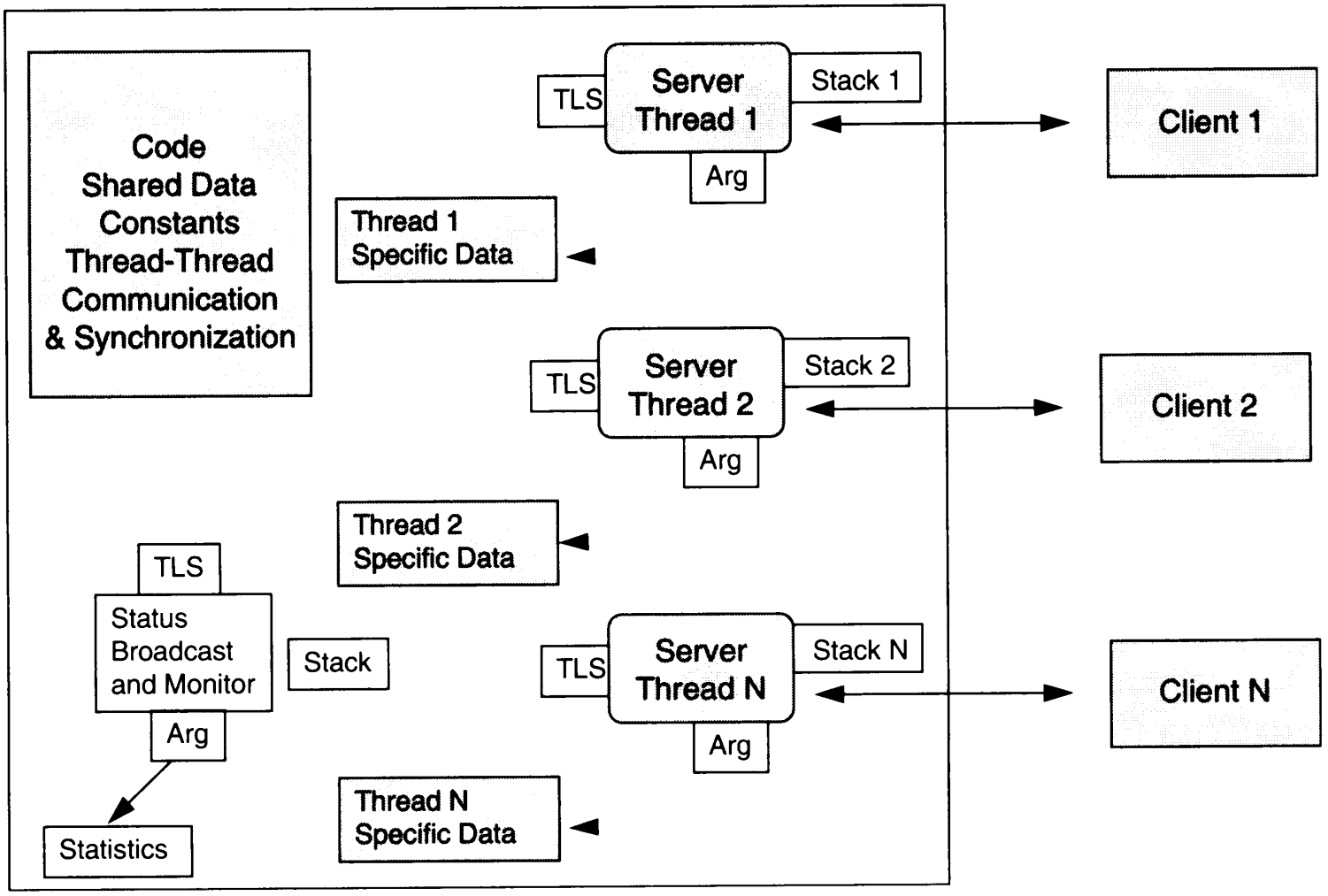
```
...
   ProcessList[0]=pi[0].hProcess;
ProcessList[1]=pi[1].hProcess;
WaitForMultipleObjects(2,ProcessList,FALSE,10000);
...
```

13. Les Threads

13. Les Threads

- Les Threads dans un processus partage les mêmes données et le même code
- Chaque Thread a sa propre pile
- Le processus appelant peut transmettre un argument au thread (pointeur 32 bits)
- Chaque Thread a ses propres données locales (TLS : Thread Local Storage)
- Windows NT gère un système Multi-Processeurs en distribuant les threads (parfois d'un même processus) sur des processeurs séparés.

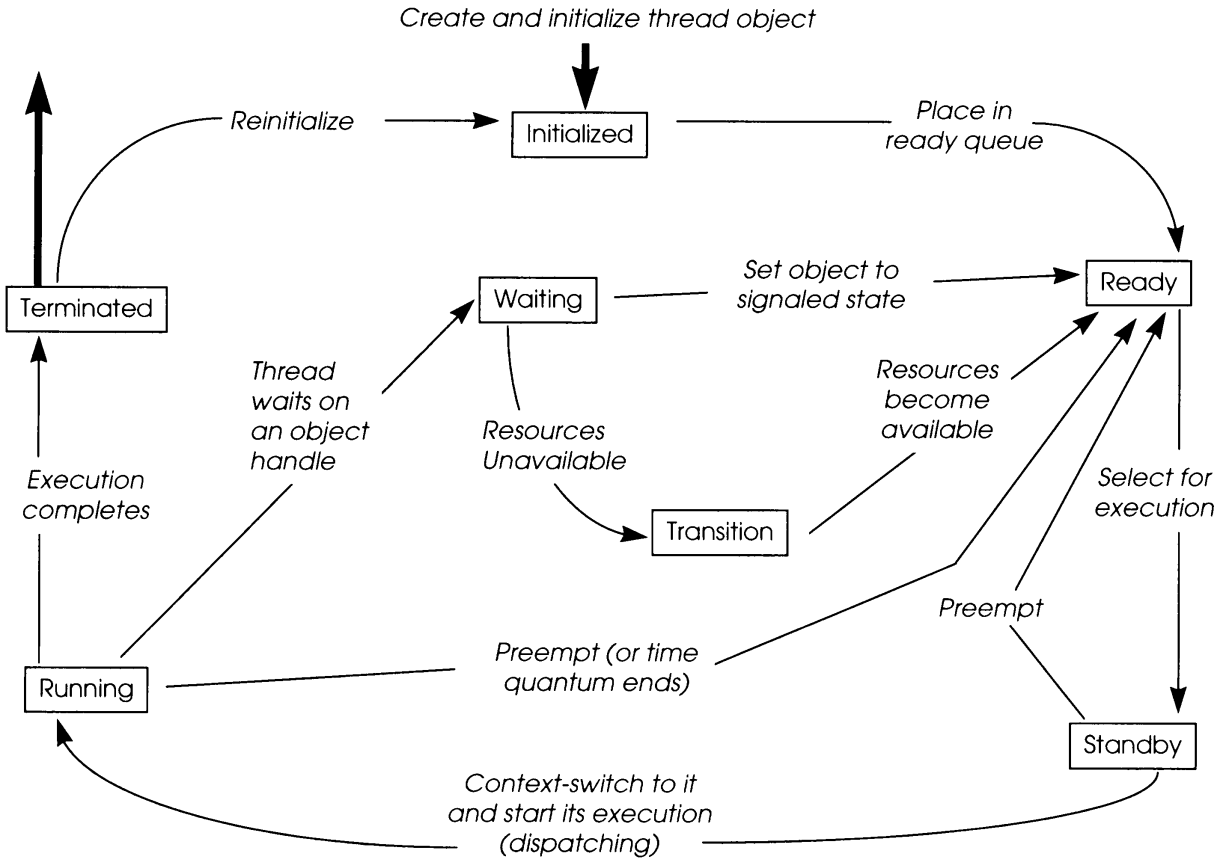
13.1 Exemple : Serveur Multi-Threads



13.2 Ordonnancement de Threads

♦ Les états d'un Thread

- *Running* : en train de s'exécuter sur un processeur
- *Waiting & Transition* : en attente
- *Ready & Standby* : prêt à s'exécuter
- *Suspended : mis en attente (Resumed)*
- *Initialized* : initialisée
- *Terminated* : terminée



13.3 Création d'un Thread : CreateThread

- La fonction CreateThread, crée un thread qui s'exécute dans l'espace d'adresses du processus appelant. Retourne un Handle sur le nouvel objet thread, NULL sinon.
- Syntaxe

```
HANDLE CreateThread(LPSECURITY_ATTRIBUTES lpThreadAttributes,
DWORD dwStackSize, LPTHREAD_START_ROUTINE lpStartAddress,
LPVOID lpParameter, DWORD dwCreationFlags, LPDWORD lpThreadId);
```

- Arguments

LPSECURITY_ATTRIBUTES lpThreadAttributes, pointeur sur les attributs de sécurité du thread (si NULL prend les attributs de sécurité du thread appelant)

DWORD dwStackSize, taille de la pile initiale du thread en octets

LPTHREAD_START_ROUTINE lpStartAddress, pointeur sur la fonction du thread (DWORD WINAPI ThreadFunc(LPVOID))

LPVOID lpParameter, pointeur sur l'argument du nouveau thread

DWORD dwCreationFlags, flags de création

- *LPDWORD lpThreadId, pointeur sur l'identificateur du nouveau thread*

♦ Exemple

```
HThread[cThread]=CreateThread(

NULL, /*pointeur sur la structure SECURITY_ATTRIBUTES, attributs de sécurité,
si NULL alors attributs de sécurité par défaut */

0, /* taille initiale de la pile, si 0 alors la taille de la pile du nouveau thread sera
la même que celle du premier thread */

(LPTHREAD_START_ROUTINE) Print, /* l'adresse de la fonction où le thread
commencera à s'exécuter */

(void*)cThread,/* un paramètre 32 bits passé à la fonction du nouveau thread
*/

0,/* un flag indiquant comment le thread est créé : CREATE_SUSPENDED, met
en attente le nouveau thread ou (jusqu'au premier ResumeThread vers lui),  0
l'autorise à se lancer immédiatement. */

&nThreadID[cThread]/* l'adresse de la variable 32 bits qui retourne
l'Identificateur du nouveau thread */

);
```

13.4 Terminaison d'un Thread

13.4.1 ExitThread

- La fonction ExitThread, termine un thread. Cette fonction ne retourne pas de valeur.
- Syntaxe

  - *VOID ExitThread(DWORD dwExitCode);*

- Arguments

  - *DWORD dwExitCode, spécifie le code de retour du thread*

13.5 GetExitCodeThread

- Récupère le code de retour du thread spécifié. Retourne une valeur nulle en cas d'échec.
- Syntaxe

  - *BOOL GetExitCodeThread(HANDLE hThread, LPDWORD lpExitCode);*

- Arguments

  - *HANDLE hThread, handle du thread spécifié*
  - *LPDWORD lpExitCode , pointeur sur le code de retour*

13.6 SuspendThread

- La fonction SuspendThread suspend le thread spécifié (incrémente le compteur de suspension du thread). Retourne la valeur précédente du compteur de suspension, 0xFFFFFFFF sinon. Le compteur de suspension à une valeur maximale MAXIMUM_SUSPEND_COUNT.
- Syntaxe

  - *DWORD SuspendThread(HANDLE hThread);*

- Argument

  - *HANDLE hThread, Handle du thread concerné*

13.7 ResumeThread

- La fonction ResumeThread décrémente le compteur de suspension du thread. Quand le compteur est à zéro, le thread est à nouveau exécutable. Retourne la valeur précédente du compteur de suspension, 0xFFFFFFFF sinon.
- Syntaxe

- *DWORD ResumeThread(HANDLE hThread);*

♦ Argument

- *HANDLE hThread, Handle du thread concerné*

13.8 Exemple

♦ Attention au compteur *suspend count*

```
SuspendThread(HThread[cThread]);
SuspendThread(HThread[cThread]);
ResumeThread(HThread[cThread]); /* thread pas encore actif */
ResumeThread(HThread[cThread]); /* thread maintenant actif*/
```

13.9 Attente de terminaison d'un Thread

13.9.1 WaitForSingleObject

- La fonction WaitForSingleObject teste l'état courant de l'objet spécifié. Si l'état de l'objet est nonsignaled, le thread appelant se met en attente active. Le thread prend peu de temps CPU dans cet état d'attente d'un état signaled d'un objet ou d'un timeout éventuel. Retourne WAIT_FAILED en cas d'échec, WAIT_TIMEOUT sur timeout
- Syntaxe

  - *DWORD WaitForSingleObject(HANDLE hHandle, DWORD dwMilliseconds);*

- Arguments

  - *HANDLE hHandle, Handle de l'objet à attendre*
  - *DWORD dwMilliseconds, Timeout en millisecondes (si la valeur est nulle le test est non-bloquant, sinon il s'agit d'un timeout en millisecondes, enfin INFINITE pour une attente infinie).*

13.9.2 WaitForMultipleObjects

- La fonction WaitForMultipleObject teste l'état courant de l'objet spécifié. Si l'état de l'objet est nonsignaled, le thread appelant se met en attente active. Le thread prend peu de temps CPU dans cet état d'attente d'un état signaled d'un objet ou d'un timeout éventuel. Retourne WAIT_FAILED en cas d'échec, WAIT_TIMEOUT sur timeout
- Syntaxe

  - *DWORD WaitForMultipleObjects(WORD nCount, CONST HANDLE *lpHandles, BOOL bWaitAll, DWORD dwMilliseconds);*

- Arguments

  - *WORD nCount, nombre d'objets handle de la tableau lpHanldes (le maximum est MAXIMUM_WAIT_OBJECTS)*
  - *CONST HANDLE *lpHandles, pointe sur un tableau d'objets handles*
  - *BOOL bWaitAll, Si TRUE attend que tous les objets du tableau soient dans l'état signaled, Si FALSE attend qu'un des objets du tableau soit dans l'état signaled*

- *DWORD dwMilliseconds, Timeout en millisecondes (si la valeur est nulle le test est non-bloquant, sinon il s'agit d'un timeout en millisecondes, enfin INFINITE pour une attente infinie.)*

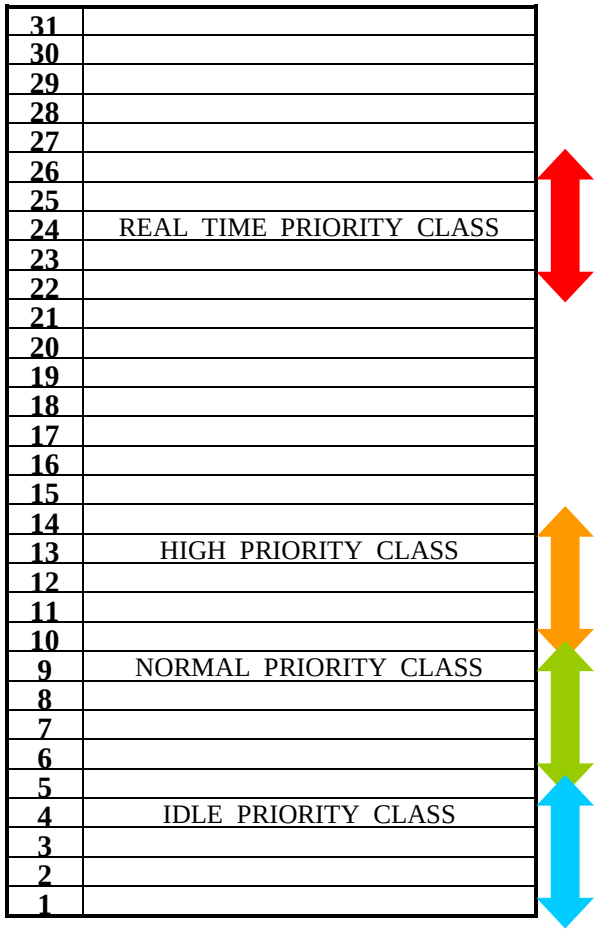
13.10 Priorités

- Les classes de Priorités par processus
  - IDLE_PRIORITY_CLASS (priorité de base 4)
  - NORMAL_PRIORITY_CLASS (priorité de base 9 ou 7)
  - HIGH_PRIORITY_CLASS (priorité de base 13)
  - REAL_TIME_PRIORITY_CLASS (priorité de base 24)

- Les priorités des threads

  - Dans l'intervalle de +/- 2 autour de la priorité de base :

  - THREAD_PRIORITY_LOWEST
  - THREAD_PRIORITY_BELOW_NORMAL
  - THREAD_PRIORITY_NORMAL
  - THREAD_PRIORITY_ABOVE_NORMAL
  - THREAD_PRIORITY_HIGHEST



13.10.1 Gestion des Priorités

- La fonction **SetPriorityClass** définit la classe de priorités du processus spécifié. Cette valeur définit la priorité de base des threads du processus. La fonction retourne une valeur nulle en cas d'échec.
- Syntaxe

  - *BOOL SetPriorityClass(HANDLE hProcess, DWORD dwPriorityClass);*

- Arguments

  - *HANDLE hProcess, handle du processus*
  - *DWORD dwPriorityClass, valeur de la classe de priorités*

- Les classes de priorités

  - *IDLE_PRIORITY_CLASS (priorité de base 4)*
  - *NORMAL_PRIORITY_CLASS (priorité de base 9 ou 7)*
  - *HIGH_PRIORITY_CLASS (priorité de base 13)*
  - *REAL_TIME_PRIORITY_CLASS (priorité de base 24)*

- La fonction **GetPriorityClass**, retourne la classe de priorités du processus spécifié, 0 en cas d'échec. Cette valeur définit la priorité de base des threads du processus.
- Syntaxe

  - *DWORD GetPriorityClass(HANDLE hProcess);*

- Arguments

  - *HANDLE hProcess, handle du processus*

- La fonction **SetThreadPriority** définit la priorité du thread spécifié. Cette valeur et la classe de priorités du processus, définissent le niveau de priorité de base du thread.
- Syntaxe

  - *BOOL SetThreadPriority(HANDLE hThread, int nPriority);*

- Arguments

  - *HANDLE hThread, handle sur le thread*
  - *int nPriority, niveau de priorité du thread*

♦ Les priorités

- *THREAD_PRIORITY_ABOVE_NORMAL (+1)*
- *THREAD_PRIORITY_BELOW_NORMAL (-1)*
- *THREAD_PRIORITY_HIGHEST (+2)*
- *THREAD_PRIORITY_IDLE (1 pour IDLE_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS, ou HIGH_PRIORITY_CLASS et 16 pour REALTIME_PRIORITY_CLASS)*
- *THREAD_PRIORITY_LOWEST (-2)*
- *THREAD_PRIORITY_NORMAL (+0)*
- *THREAD_PRIORITY_TIME_CRITICAL (15 pour IDLE_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS, ou HIGH_PRIORITY_CLASS , et 31 pour REALTIME_PRIORITY_CLASS).*

♦ La fonction **GetThreadPriority** retourne la priorité du thread spécifié. Cette valeur et la classe de priorités du processus, définissent le niveau de priorité de base du thread.

♦ Syntaxe :

- *int GetThreadPriority(HANDLE hThread);*

♦ Arguments :

- *HANDLE hThread , handle sur le thread spécifié*