1516 L2 SEPS TP 2 - Multiprocessus Windows

1. Familiarisation avec les termes usités
   1. #include <windows.h>

Inclus la libraire « *Windows* » pour gérer les processus et les threads. Elle contient toutes les fonctions et toutes les déclarations nécessaires telles que celles présentées ci-dessous.

* 1. STARTUPINFO si

Type / structure représentant les informations de démarrage d’un processus.

Typedef struct \_STARTUPINFO{

DWORD cb ;

…

HANDLE hStdInput ;

HANDLE hStdOutput ;

HANDLE hStdError ;

} STARTUPINFO, \*LP STARTUPINFO ;

* 1. PROCESS\_INFORMATION pi

Type servant au processus contenant ses informations.

Typedef struct \_PROCESS\_INFORMATION  {

HANDLE hProcess ;

HANDLE hThread ;

DWORD dwProcessId ;

DWORD dwThreadId ;

} PROCESS\_INFORMATION, \*LPPROCESS\_INFORMATION ;

* 1. ZeroMemory()

Initialisation à NULL de la mémoire. Dans notre contexte, cela permet de mettre la Variable de type STARTUPINFO à nul ;

* 1. CreateProcess("programToRun", "commandLine", NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)

Crée un processus à l’aide d’un chemin vers un fichier exécutable et le lance avec certains paramètres que l’on choisit en donnant les variables STARTUPINFO et PROCESSINFO.

* 1. WaitForSingleObject(pi.hProcess, INFINITE)

Attend que le processus se soit arrêté pour continuer le code. Cela permet de synchroniser les applications.

* 1. CloseHandle(pi.hProcess) ou CloseHandle(pi.hThread)

Ferme le processus ou le thread ciblé.

* 1. Sleep( t\_En\_Milli\_Secondes ) (C++) ou sleep(t\_En \_Secondes ) (c)

Permet de mettre le processus qui l’emploie en pause pendant un temps déterminé.

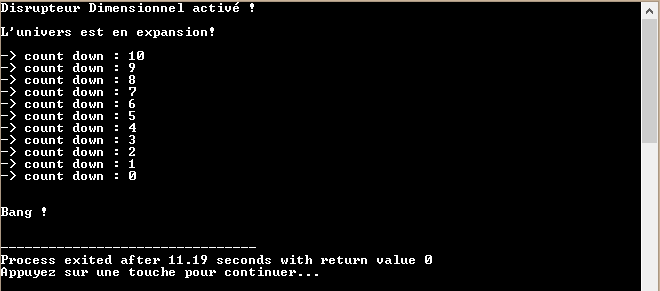
* 1. WaitForMultipleObject(…)

Elle est la version multiprocessus de la fonction WaitForSingleObject(…).

Elle prend comme paramètre le nombre de processus, un tableau avec ces processus, la manière dont elle réagit quand un processus à terminer et par un temps.

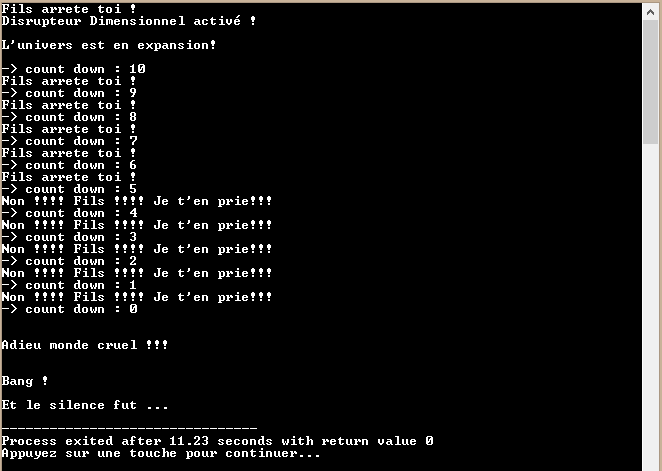
1. « Mec, elle est où ma caisse »
   1. Dis rupteur Dimensionnel

Le programme utilise simplement les sleep(time) pour retarder l’affichage du compteur et la décrémentation. Ce qui permet d’obtenir :



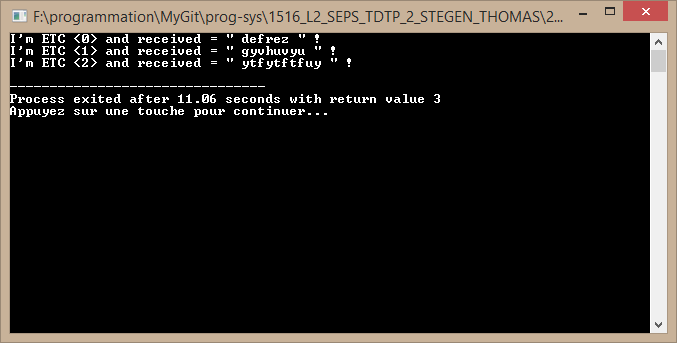
* 1. JessieEtChester

Ce programme génère un nouveau processus en faisant appel à la fonction CreateProcess(…) en vérifiant qu’elle est bien créée pour simuler un père qui supplie son fils de pas détruire l’univers. Ce qui en jouant un peu avec les Sleep(…) donne ceci :



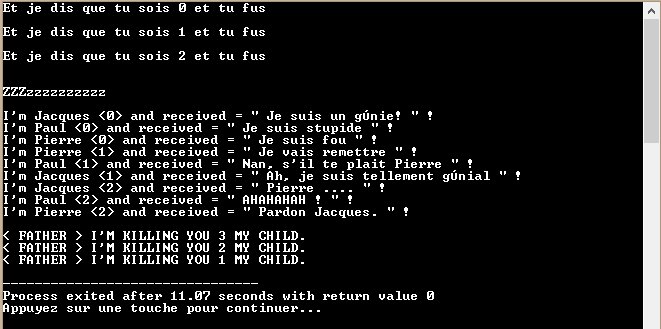
1. Pierre, Paul et Jacques
   1. Fils

Ici, on crée un processus qui demande des paramètres et à l’aide de argc et argv, on affiche les chaines une après l’autre. Pour faire cela :

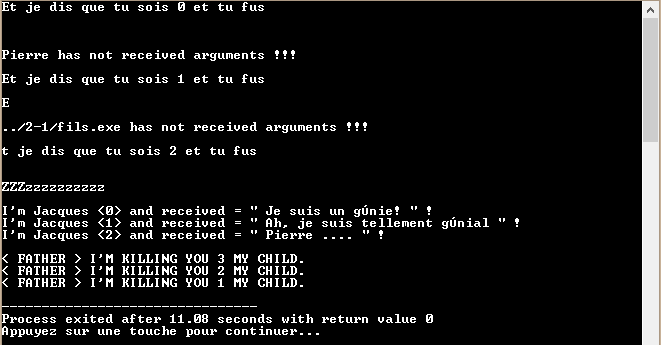


* 1. Père

Le processus père va simplement créer à la chaine 3 processus en enregistrant leur identifiant dans un tableau pour attendre la fin de chacun à l’aide de la fonction WaitForMultipleObject(…) et les détruire par après. Ce qui donne cette conversation un peu aléatoire ou plutôt choisie par le scheduller :



Maintenant si l’on ne lui passe pas de paramètre en plus du nom, il va vous prévenir de cette manière :



1. My\_Sweet\_Little\_Multi\_Process
   1. Comment récupérer les résultats des processus fils?

Il faut terminer le processus par la fonction ExitProcess(…) en lui passant le résultat en paramètre.

Pour récupérer la valeur, il faut utiliser la fonction GetExitProcess() en lui passant en paramètre l’adresse de notre variable.

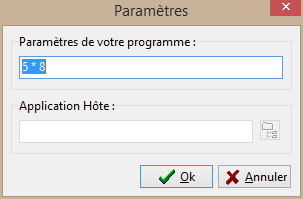
* 1. Le programme personnalisé

J’ai choisi de simple pour montrer l’efficacité des processus car grâce à eux, nous pouvons découpe nos grosses tâches en plusieurs petites tout en permettant au processus principale de continuer ce qu’il a à faire mais je ne vais pas jusque-là.

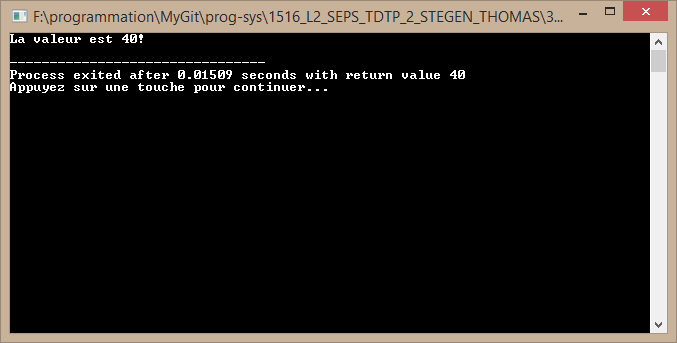
Ici, le père demande simplement à l’utilisateur une opération élémentaire que le fils va réaliser et père va récupérer le résultat.

Un autre moyen aurait été d’utiliser un pipe et des mutexs mais cela n’a pas encore été vu au cours.

* **Les paramètres pour le fils**



* **Le fils**



* **Le père**

