

© Copyright 2010 tv <tvaira@free.fr>

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License,

Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover.

☞ can obtain a copy of the GNU General Public License : write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

---

## SOMMAIRE

---

Séquence 1 : Correction.....	3
I . Introduction.....	3
II . L'appel CreateProcess.....	3
III . Les signaux.....	6
IV . Les évènements.....	6

## SÉQUENCE 1 : CORRECTION

---

### I . Introduction

Sous Windows, il y avait principalement trois options pour écrire des programmes multi-tâches :

- Utiliser la **Win32 Application Programming Interface (API)** en conjonction avec la **C run-time library (CRT)**. Comme son nom l'indique, l'API est implémentée en C et elle intègre les fonctions d'appels systèmes de plus bas niveau accessibles au développeur d'applications windows actuel.
- Utiliser la **Microsoft Foundation Class (MFC)** en conjonction avec le C++. La MFC a été développée comme un ensemble de classes C++ qui agissent en tant que « wrappers » de l'API Win32, afin d'en faciliter l'utilisation et l'intégration dans le cadre d'un développement orienté-objet.
- Utiliser la plateforme **.NET** qui offre plusieurs options aussi pour décomposer une application en processus légers. Le « namespace » **System.Threading** en est une.

Évidemment, d'autres framework sont disponibles. On peut citer : Qt, CLX/VCL de Borland, ...

### II . L'appel `CreateProcess`

Cet appel système permet la création et l'exécution d'un processus enfant.

*Remarque : si on veut comparer avec Unix, l'appel `CreateProcess()` est l'équivalent des appels `fork()` et `exec()` regroupés.*

**Exemple :**

```

#include <windows.h>
#include <stdio.h>

int main( VOID ) {
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    ZeroMemory( &si, sizeof(si) );
    si.cb = sizeof(si);
    ZeroMemory( &pi, sizeof(pi) );

    // Start the child process.
    if( !CreateProcess( ".\\Child.exe", //NULL then use command line
        NULL, //TEXT("Child"), // Command line.
        NULL, // Process handle not inheritable.
        NULL, // Thread handle not inheritable.
        FALSE, // Set handle inheritance to FALSE.
        0, // No creation flags.
        NULL, // Use parent's environment block.
        NULL, // Use parent's starting directory.
        &si, // Pointer to STARTUPINFO structure.
        &pi ) // Pointer to PROCESS_INFORMATION structure.
    )
    {
        printf( "CreateProcess failed (%d).\n", GetLastError() );
        return;
    }

    // Wait until child process exits.
    WaitForSingleObject( pi.hProcess, INFINITE );

    // Close process and thread handles.
    CloseHandle( pi.hProcess );
    CloseHandle( pi.hThread );
}

```

**Code source d'un fils :**

```

#include <windows.h>
#include <stdio.h>
#include <tchar.h>
int main()
{
    printf("I am the child. My Pid and Tid:  (%d, %d).\n",
    GetCurrentProcessId(), GetCurrentThreadId());
    printf( "I am the child. Going to sleep for 5 seconds. \n");
    Sleep(5000);
    ExitProcess(10); // exit code for all threads
}

```

Référence :

<http://msdn.microsoft.com/fr-fr/library/ms682425%28vs.85%29.aspx>

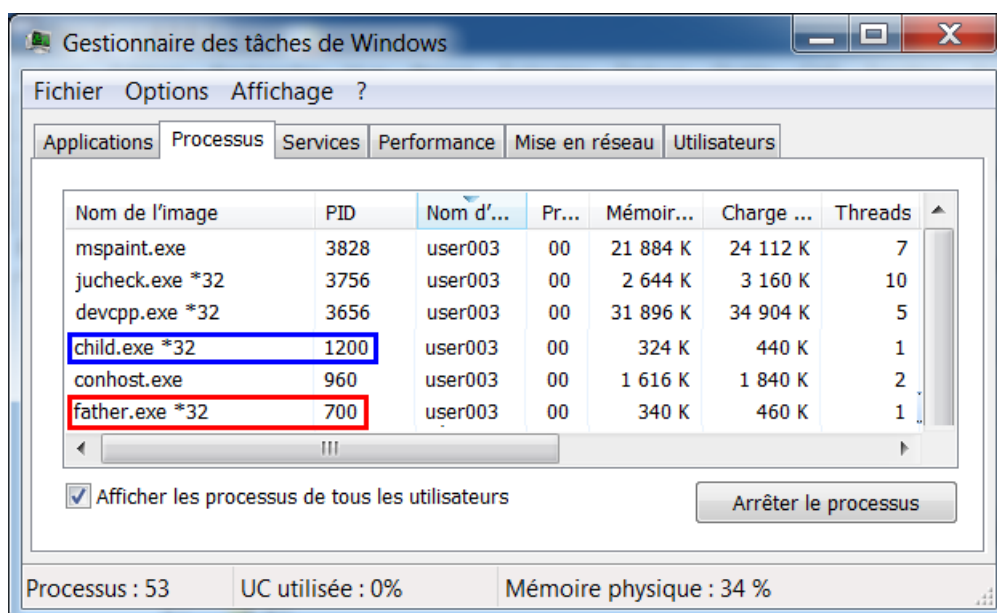
Remarques :

Sous Windows, tout processus dispose d'au moins un thread, de manière à exécuter le point d'entrée du programme. Ce thread ne pourra jamais être terminé sans que l'application ne soit elle-même terminée. Le processus ne doit en aucun cas être assimilé à ce thread. Le thread constitue simplement l'unité d'exécution de base du processus. Le système ne communiquera jamais avec le processus mais toujours avec l'un des threads de ce processus. En effet, le processus n'étant pas une unité d'exécution il ne réalise aucune tâche. Le premier thread est créé par le système. La création de nouveaux threads devra être explicite par l'appel `CreateThread`. Un processus peut donc posséder plusieurs threads.

```

C:\Dev-Cpp\Examples\processus>father
I am the child. My Pid and Tid: (2440, 1032).
I am the child. Going to sleep for 5 seconds.
I am the father. My child [2440,1032] has terminated with (10).
I am the father. Going to terminate.

C:\Dev-Cpp\Examples\processus>father
I am the child. My Pid and Tid: (1200, 3676).
I am the child. Going to sleep for 5 seconds.
  
```



### III . Les signaux

L'API Win32 ne supporte pas les signaux.

Le concept le plus proche disponible dans l'API Win32 est la notion d'évènement.

### IV . Les évènements

L'API Windows fournit des fonctions permettant de créer des évènements de manière explicite. L'état d'un évènement peut être signalé ou non signalé. Lorsqu'un évènement est en état non signalé, l'appel à une fonction d'attente provoquera la suspension du thread appelant jusqu'à ce que l'évènement passe en état signalé.

Les fonctions **CreateEvent()** et **CloseHandle()** peuvent être utilisées pour la création et la destruction d'évènements. Les fonctions **SetEvent()**, **ResetEvent()** et **PulseEvent()** pourront ensuite être utilisées pour modifier l'état de l'objet.

Les évènements peuvent être configurés de différentes manières. La configuration d'un évènement se fait lors de sa création.

- Si l'objet est en mode 'réinitialisation manuelle' il restera en état signalé tant que son état ne sera pas explicitement réinitialisé grâce à la fonction **ResetEvent()**. Si plusieurs threads étaient en attente de cet objet, ils seront alors tous libérés au moment où l'état de l'objet passera à 'signalé'.
- Si l'objet est en mode 'réinitialisation automatique', son état sera automatiquement remplacé à 'non signalé' dès qu'un thread en attente sera libéré. Si plusieurs threads sont en attente d'un même évènement, alors un seul thread sera libéré au moment où l'état sera passé à 'signalé'.

Les évènements sont un moyen pratique de synchroniser plusieurs threads de manière explicite :

Par exemple, les évènements peuvent être utilisés pour mettre un thread en attente tandis qu'un autre thread initialise des données. Dès que les données seront initialisées, le ou les threads seront débloqués par le thread ayant initialisé les données.

Les événements peuvent être nommés ou non. Si l'événement est nommé, son nom doit être unique dans l'ensemble du système. Si une autre application tente de créer un événement du même nom, la fonction `CreateEvent()` retournera un `HANDLE` sur l'événement précédemment créé. Ceci peut être utilisé pour synchroniser des applications entre elles (voir exemple).

Référence :

[http://msdn.microsoft.com/fr-fr/library/ms682396\(v=VS.85\).aspx](http://msdn.microsoft.com/fr-fr/library/ms682396(v=VS.85).aspx)

**Exemple :**

```

#include <windows.h>
#include <stdio.h>

int main( VOID )
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    DWORD status ;
    HANDLE hEvent;

    ZeroMemory( &si, sizeof(si) );
    si.cb = sizeof(si);
    ZeroMemory( &pi, sizeof(pi) );

    system("cd"); // juste pour le plaisir

    // On crée un événement
    // - Non signalé par défaut
    // - A réinitialisation manuelle
    // - Nommé
    hEvent = CreateEvent( NULL, TRUE, FALSE, "Global\\Event1" );

    if( !CreateProcess(".\\Child.exe", NULL, NULL, NULL, FALSE, 0, NULL,
NULL, &si, &pi))
    {
        printf( "CreateProcess failed (%d).\n", GetLastError() );
        ExitProcess(1);
    }

    // Le fils est en attente
    printf("Pret ?\n"); Sleep(2000); printf("Partez !\n");

    // C'est parti ! On signale l'évènement au fils
    SetEvent(hEvent);

    WaitForSingleObject( pi.hProcess, INFINITE );

    if(GetExitCodeProcess( pi.hProcess, &status))
        printf("I am the father. My child [%d,%d] has terminated with
(%d). \n", pi.dwProcessId, pi.dwThreadId, status );
    else printf( "GetExitCodeProcess failed (%d).\n", GetLastError() );

    CloseHandle( pi.hProcess );
    CloseHandle( pi.hThread );
    // On détruit l'évènement
    CloseHandle(hEvent);
    printf( "I am the father. Going to terminate.\n");
    ExitProcess(0);
}

```



**Code source d'un fils :**

```
#include <windows.h>
#include <stdio.h>
#include <tchar.h>

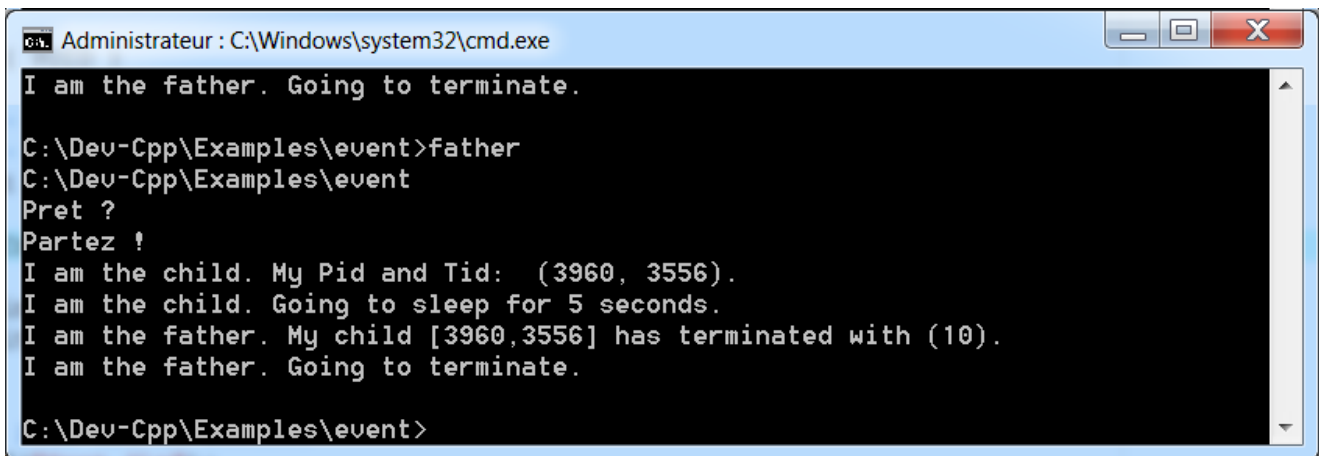
int main()
{
    HANDLE hEvent;

    hEvent = CreateEvent( NULL, TRUE, FALSE, "Global\\Event1" );

    // On attend l'autorisation pour démarrer
    WaitForSingleObject(hEvent, INFINITE);

    printf("I am the child. My Pid and Tid: (%d, %d).\n",
        GetCurrentProcessId(), GetCurrentThreadId());

    printf( "I am the child. Going to sleep for 5 seconds. \n");
    Sleep(5000);
    ExitProcess(10);    // exit code for all threads
}
```



```
Administrateur : C:\Windows\system32\cmd.exe
I am the father. Going to terminate.
C:\Dev-Cpp\Examples\event>father
C:\Dev-Cpp\Examples\event
Pret ?
Partez !
I am the child. My Pid and Tid: (3960, 3556).
I am the child. Going to sleep for 5 seconds.
I am the father. My child [3960,3556] has terminated with (10).
I am the father. Going to terminate.
C:\Dev-Cpp\Examples\event>
```