

PRÁCTICA 4

SERVICIOS DE PROCESOS E HILOS EN WINDOWS

Objetivos

El objetivo de esta práctica es programar en C la creación de procesos e hilos, así como la sincronización entre hilo primario e hilos secundarios. Para ello se plantearán una serie de ejercicios donde se introducirán los siguientes servicios de Windows relacionados con procesos, hilos y su sincronización:

- Servicios relacionados con objetos en general: `CloseHandle`
- Identificación de procesos e hilos: `GetCurrentProcessId`, `GetCurrentThreadId`
- Creación de hilos: `CreateThread`
- Servicios de espera sobre objetos: `WaitForSingleObject`, `WaitForMultipleObjects`
- Creación de procesos: `CreateProcess`

Cuestiones previas

Para utilizar los servicios de Windows es necesario incluir el fichero de cabecera `windows.h`.

Para realizar adecuadamente esta práctica es necesario haber repasado previamente cómo se realiza en C el paso de parámetros al programa principal.

En las prácticas de programación en C sobre Windows utilizaremos el compilador `gcc` directamente desde la línea de comandos según la siguiente sintaxis:

```
gcc fichero_fuente.c -o fichero_ejecutable.exe
```

1. Objetos, descriptores e identificadores

Un *objeto* es una entidad del sistema que puede crearse, abrirse o cerrarse utilizando para ello *servicios* o *funciones* del sistema. Son objetos: procesos, hilos, mutex, ficheros,...

Un objeto se crea utilizando un servicio específico para cada tipo de objeto. Como resultado de la creación se obtiene el *descriptor* o *handle* que nos permite hacer referencia al objeto mientras este exista.

Una vez que el objeto no va a ser utilizado más, debe ser cerrado mediante el servicio:

```
BOOL CloseHandle(HANDLE hObject);
```

donde el parámetro `hObject` es el descriptor del objeto que queremos cerrar.

Procesos e hilos tienen además asignado un identificador único en el sistema. Los servicios que permiten obtener estos valores son los siguientes:

```
DWORD GetCurrentProcessId(void);  
DWORD GetCurrentThreadId(void);
```

que devuelven, respectivamente, los identificadores del proceso y del hilo que utiliza el servicio.

2. Creación de hilos

Cuando se crea un proceso en Windows, el sistema crea automáticamente el **hilo primario**, el cuál a su vez puede crear hilos adicionales (secundarios). La creación de hilos se realiza mediante el servicio:

```
HANDLE CreateThread(LPSECURITY_ATTRIBUTES  lpsaThread,
                   DWORD                     cbStack,
                   LPTHREAD_START_ROUTINE  lpStartAddr,
                   LPVOID                   lpvThreadParm,
                   DWORD                     fdwCreate,
                   LPDWORD                   lpIDThread );
```

Los parámetros de llamada de este servicio son:

- **lpsaThread**: Puntero a los atributos de seguridad del nuevo hilo (puede ser NULL).
- **cbStack**: Indicador del tamaño de la pila. 0 indica tamaño por defecto.
- **lpStartAddr**: Puntero a la *función* que contiene el código que ejecutará el hilo (nombre de la función).
- **lpvThreadParm**: Puntero a los *parámetros* de la función de inicio del hilo.
- **fdwCreate**: Indicadores que afectan a la forma de creación del hilo (puede ser NULL)
- **lpIDThread**: Puntero al *identificador* del hilo, una vez realizada la llamada (valor asignado durante la creación del hilo).

El servicio devuelve el descriptor del hilo creado.

3. Servicios de espera sobre objetos

Una vez creado un objeto, podemos obligar a que un hilo espere hasta que se de una determinada situación (por ejemplo, la terminación de uno o varios hilos secundarios). Para ello hay dos servicios que permiten implementar funciones de espera sobre objetos. Si esperamos a un único objeto utilizaremos:

```
DWORD WaitForSingleObject(HANDLE hObject,
                          DWORD dwTimeout );
```

Los parámetros de llamada de este servicio son:

- **hObject**: Descriptor del objeto por el que se espera.
- **dwTimeout**: Tiempo máximo que vamos a esperar (puede ser INFINITE).

Cuando la espera finaliza de forma correcta el servicio devuelve **WAIT_OBJECT_0** o **WAIT_TIMEOUT** (si se ha cumplido el tiempo de espera máximo). En caso de error puede retornar **WAIT_ABANDONED** o **WAIT_FAILED**.

Si necesitamos esperar a varios objetos utilizaremos:

```
DWORD WaitForMultipleObjects(HANDLE cObjects,
                             LPHANDLE lpHandles,
                             BOOL bWaitAll,
                             DWORD dwTimeout );
```

Los parámetros de llamada de este servicio son:

- **cObjects**: Indica el número de objetos por los que se espera.
- **lpHandles**: Puntero a un vector con los descriptors de los diversos objetos por los que se espera.
- **bWaitAll**: Indica si se espera hasta que todos los objetos estén señalados (TRUE) o solo se espera por el primero (FALSE).
- **dwTimeout**: Especifica el tiempo máximo de espera.

El valor retornado es idéntico al caso anterior.

Ejemplo de creación de un hilo

Este código crea un hilo secundario que muestra por pantalla un valor numérico que se ha pasado como parámetro al lanzar la ejecución del programa.

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

/* Definición de variables globales */
HANDLE hThread;
DWORD IDThread;

/* Código que ejecuta el hilo secundario */
DWORD HiloSecundario(LPDWORD lpdwParam)
{
    printf("Valor del parametro: %d\n", *lpdwParam);
    return (0) ;
}

/* Programa principal */
int main(int argc, char *argv[])
{
    DWORD parametro;

    if (argc<2) {
        printf("Falta el parametro de llamada\n");
        exit(1);
    }

    parametro=atoi(argv[1]);

    /* Se crea el hilo secundario */
    hThread=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)HiloSecundario,&parametro,0,&IDThread);

    /* Esperamos la finalización del hilo */
    WaitForSingleObject(hThread, INFINITE) ;

    /* Cerramos el descriptor del hilo */
    CloseHandle(hThread);
    return 0;
}
```

Ejercicio 1 Escribe un programa que cree un hilo secundario que recibe como parámetro un valor entero N pasado en la línea de órdenes (consola de Windows). El hilo secundario debe ejecutar N veces un bucle donde suma 1 a una variable local inicializada a 0. El hilo primario debe esperar hasta que finalice el hilo secundario y entonces cerrar el descriptor.

Localiza en el disco tanto el fichero fuente como el ejecutable y realiza la ejecución del programa desde la consola de Windows.

Ejemplo de salida por pantalla:

```
Soy el proceso 2916
Comienza el hilo primario (ID 1716)
Comienza el hilo secundario (ID 1136)
Valor final de la variable: 30
Finaliza el hilo secundario (ID 1136)
Finaliza el hilo primario (ID 1716)
```

Ejercicio 2 Escribir un programa que cree dos hilos que ejecuten la misma función. Cada uno de los hilos debe recibir como parámetro un valor distinto (1 y 2). Tanto el hilo primario como los dos hilos creados deben mostrar por pantalla (en el momento oportuno) los mensajes *Comienza el hilo X* y *Finaliza el hilo X*, donde X es primario, 1 o 2. El hilo primario debe esperar hasta que finalicen los hilos secundarios y entonces cerrar los descriptores.

Por su parte los hilos secundarios ejecutan N veces un bucle donde deben modificar una variable compartida cuyo valor inicial es 0: el hilo 1 incrementa la variable en un unidad y el hilo 2 la decrementa también en una unidad. Después de cada modificación de la variable cada hilo debe mostrar un mensaje indicando quién es (1 o 2), que operación ha realizado (incremento, decremento) y el valor de la variable después de la operación.

Ejemplo de salida por pantalla:

```
Soy el proceso 2824
Comienza el hilo primario (ID 2756)
Comienza el hilo 2 (ID 2772)
Comienza el hilo 1 (ID 2076)
Hilo 2, decrementa, -1
Hilo 2, decrementa, -2
Finaliza el hilo 2 (ID 2772)
Hilo 1, incrementa, -1
Hilo 1, incrementa, 0
Finaliza el hilo 1 (ID 2076)
Finaliza el hilo primario (ID 2756)
```

4. Creación de procesos

Un proceso se crea mediante el servicio:

```
BOOL CreateProcess(LPCTSTR          lpszImageName,
                  LPCTSTR          lpszCommandLine,
                  LPSECURITY_ATTRIBUTES lpsaProcess,
                  LPSECURITY_ATTRIBUTES lpsaThread,
                  BOOL              fInheritHandles,
                  DWORD             fdwCreate,
                  LPVOID            lpvEnvironment,
                  LPTSTR            lpszCurDir,
                  LPSTARTUPINFO     lpsiStartInfo,
                  LPPROCESS_INFORMATION lppiProcInfo) ;
```

Los parámetros de llamada de este servicio son:

- **lpszImageName**: Puntero al nombre del archivo ejecutable (puede ser NULL).
- **lpszCommandLine**: Puede ser un puntero al nombre de la aplicación y sus parámetros o solamente a los parámetros si el nombre de la aplicación se puso en el argumento anterior (puede ser NULL).
- **lpsaProcess**: Puntero a los atributos de seguridad del nuevo objeto del proceso (puede ser NULL).
- **lpsaThread**: Puntero a los atributos de seguridad del nuevo objeto del hilo primario (puede ser NULL).
- **fInheritHandles**: Indicación de si los handles creados por el proceso serán o no heredados por el procesos hijo (TRUE o FALSE).
- **fdwCreate**: Indicadores que afectan a la forma de creación del proceso y su prioridad (puede ser 0).
- **lpvEnvironment**: Puntero a una zona de memoria que contiene variables de entorno. Si es NULL heredará los del proceso creador.

- `lpzCurDir`: Puntero a la unidad y directorio de trabajo actual para el proceso. Si es NULL heredará los del proceso creador.
- `lpStartupInfo`: Puntero a una estructura `STARTUPINFO` con información sobre la inicialización del proceso.
- `lpProcessInfo`: Puntero a una estructura `PROCESS_INFORMATION` que contiene los descriptores de los objetos proceso e hilo primario creados, y los identificadores únicos de proceso e hilo primario.

```
typedef struct _PROCESS_INFORMATION {
    HANDLE hProcess;
    HANDLE hThread;
    DWORD  dwProcessId;
    DWORD  dwThreadId;
} PROCESS_INFORMATION;
```

Ejemplo de creación de un proceso

En este ejemplo el hilo primario del proceso original crea un nuevo proceso y su hilo primario ejecuta el Bloc de notas (notepad) con el fichero ejemplo.txt. El proceso original espera hasta que termine el nuevo proceso y cierra los descriptores.

```
STARTUPINFO siStartupInfo;
PROCESS_INFORMATION piProcessInfo;

// inicializa a 0 las estructuras
memset(&siStartupInfo, 0, sizeof(siStartupInfo));
memset(&piProcessInfo, 0, sizeof(piProcessInfo));

siStartupInfo.cb = sizeof(siStartupInfo); // rellena el campo con el tamaño de la estructura

if( !CreateProcess(NULL, // el nombre de la aplicacion lo ponemos en el siguiente argumento
    "c:\\windows\\notepad.exe ejemplo.txt", // aplicacion + fichero a abrir
    NULL,
    NULL,
    FALSE,
    CREATE_DEFAULT_ERROR_MODE,
    NULL,
    NULL,
    &siStartupInfo,
    &piProcessInfo)) // Si no puede crear el proceso
{
    printf( "Error en CreateProcess (%d).\n", GetLastError() );
    return;
}

// Espera hasta que termine el proceso hijo
WaitForSingleObject(piProcessInfo.hProcess, INFINITE);

// Cierra los handles de proceso e hilo
CloseHandle(piProcessInfo.hThread);
CloseHandle(piProcessInfo.hProcess);
```

Ejercicio 3 Escribir un programa que cree dos procesos y espere hasta que ambos terminen. Uno de los procesos debe invocar al programa del ejercicio 1 y el otro al del ejercicio 2.

Ejemplo de salida por pantalla:

```
-->Soy el proceso 1656
-->Creado el proceso 2916
-->Creado el hilo primario 1716
... Aqui va la informacion que muestre el ejercicio 1 ...
-->Creado el proceso 2824.
-->Creado el hilo primario 2756
... Aqui va la informacion que muestre el ejercicio 2 ...
-->Finalizado el proceso 2916
-->Finalizado el proceso 2824
```