PROGRAMMAZIONE C – IL MULTITHREADING IN AMBIENTE WINDOWS

Introduzione

Un processo è un programma in esecuzione, all'interno di un processo possono coesistere uno o più thread, ciascun thread è un flusso di istruzioni che verrà eseguito dal processore. L'uso dei thread è un modo per praticare il parallelismo.

Il **thread** utilizza parte delle risorse e delle informazioni del processo, che sono condivise anche con gli altri thread, come l'area dati, i descrittori dei file, le istruzioni del programma, eccetera. Alcune informazioni sono però proprie di ogni singolo thread, come il program counter e lo stack.

Creare il Thread

Un thread inizia con una chiamata ad una funzione, definita *thread function*, e l'esecuzione del thread continua finché la funzione non termina e restituisce il controllo al **main**.

Una **thread function** deve avere questo prototipo:

```
DWORD WINAPI funzionethread(LPVOID param);
```

Per creare un thread si usa la funzione **CreateThread**.

Nel programma che segue si creeranno due thread, in ciascuno dei quali saranno stampati dei messaggi sul video.

thread1.c

```
01. #include <windows.h>
02.
03. HANDLE Thread1, Thread2;
04. DWORD dwThrdId;
05.
06. DWORD WINAPI Thread_Method_1(LPVOID param)
07. {
08.     while (1) printf("Thread Numero 1\n");
09. }
10.
11. DWORD WINAPI Thread_Method_2(LPVOID param)
12. {
13.     while (1) printf("Thread Numero 2\n");
14. }
15.
16. int main(int argc, char* argv[])
17. {
18.     Thread1 = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Thread_Method_1, 0,
19.                            0, &dwThrdId);
20.
21.     Thread2 = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Thread_Method_2, 0,
22.                            0, &dwThrdId);
23.
24.     system("pause");
25.     return 0;
26. }
```

Vengono definite due funzioni, **Thread_Method_1** e **Thread_Method_2**.

I thread vengono creati tramite la funzione **CreateThread**, come indirizzo di partenza del primo thread viene indicato l'indirizzo iniziale della funzione **Thread_Method_1**, come indirizzo di partenza del secondo thread viene indicato l'indirizzo iniziale della funzione **Thread_Method_2**.

Dopo la creazione del thread inizia l'esecuzione del flusso di istruzioni associato al thread, in questo caso l'esecuzione della funzione **Thread_Method_1** per il primo thread, e della funzione **Thread_Method_2** per il secondo thread. Contemporaneamente sono in esecuzione i due thread ed il thread del main. Ciascun thread procede nella propria esecuzione stampando il messaggio.

E:\thread1.exe

```
Thread Numero 1
Thread Numero 1
Thread Numero 2
Thread Numero 2
Thread Numero 2
Thread Numero 2
Thread Numero 1
Thread Numero 2
Thread Numero 1
Thread Numero 2
Thread Numero 1
Thread Numero 2
Thread Numero 2
Thread Numero 2
Thread Numero 1
Thread Numero 1
Thread Numero 1
Thread Numero 1
Thread Numero 2
Thread Numero 2
Thread Numero 2
Thread Numero 1
Thread Numero 1
Thread Numero 1
```

L'esecuzione va avanti indefinitamente finché il programma non viene interrotto. La funzione **CreateThread** crea un thread da eseguire all'interno dello spazio virtuale del processo chiamante.

```
HANDLE WINAPI CreateThread(
                                __in_opt LPSECURITY_ATTRIBUTES lpThreadAttributes,
                                __in SIZE_T dwStackSize,
                                __in LPTHREAD_START_ROUTINE lpStartAddress,
                                __in_opt LPVOID lpParameter,
                                __in DWORD dwCreationFlags,
                                __out_opt LPDWORD lpThreadId
);
```

lpThreadAttributes

Puntatore ad una struttura **SECURITY_ATTRIBUTES** che determina se l'handle restituito può essere ereditato dai processi figlio. Se **lpThreadAttributes** è **NULL**, l'handle non può essere ereditato.

dwStackSize

Indica la dimensione iniziale dello stack in byte. Se è pari a zero si usa la dimensione di default.

lpStartAddress

Puntatore alla funzione che sarà eseguita dal thread. Questo puntatore indica l'indirizzo di partenza del thread.

lpParameter

Puntatore ad una variabile da passare al thread.

dwCreationFlags

Flags che controllano la creazione del thread.

lpThreadId

Puntatore ad una variabile che riceverà l'identificatore del thread. Se è **NULL** l'identificatore del thread non è stato restituito.

Se la funzione ha successo viene restituito l'handle del nuovo thread, altrimenti viene restituito **NULL**.
Per creare un thread da eseguirsi nello spazio virtuale di un altro processo si deve utilizzare la funzione **CreateRemoteThread**.

Nelle funzioni **Thread_Method_1** e **Thread_Method_2** del programma che segue, a differenza del programma precedente, non sono presenti loop infiniti.

thread2.c

```
01. #include <windows.h>
02.
03. HANDLE Thread1, Thread2;
04. DWORD dwThrdId;
05.
06. DWORD WINAPI Thread_Method_1(LPVOID param)
07. {
08.     int i = 0;
09.
10.     while (i <= 5)
11.     {
12.         printf("Thread Numero 1\n");
13.         i+=1;
14.     }
15.
16.     printf("Thread Numero 1--> Finito\n");
17. }
18.
19. DWORD WINAPI Thread_Method_2(LPVOID param)
20. {
21.     int i = 0;
22.
23.     while (i <= 10)
24.     {
25.         printf("Thread Numero 2\n");
26.         i+=1;
27.     }
28.
29.     printf("Thread Numero 2--> Finito\n");
30. }
31.
32. int main(int argc, char* argv[])
33. {
34.     Thread1 = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Thread_Method_1, 0,
35.                            0, &dwThrdId);
36.
37.     Thread2 = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Thread_Method_2, 0,
38.                            0, &dwThrdId);
39.
40.     system("pause");
41.     return 0;
42. }
```

All'interno del **main** vengono creati i due thread ciascuno dei quali stamperà un messaggio:

E:\thread2.exe

```
Thread Numero 1
Thread Numero 1
Thread Numero 1
Thread Numero 1
Thread Numero 2
Thread Numero 1
Thread Numero 1
Thread Numero 2
Thread Numero 1--> Finito
Thread Numero 2
Thread Numero 2
Thread Numero 2
Thread Numero 2
Thread Numero 2
Thread Numero 2
Thread Numero 2
Thread Numero 2
Thread Numero 2
Thread Numero 2--> Finito
Premere un tasto per continuare . . .
```

L'esecuzione delle due funzioni procede contemporaneamente (i due **thread** procedono contemporaneamente), ciascuna funzione stampa un certo numero di messaggi e quindi termina. Insieme alla funzione termina anche il thread. Ciascuna funzione prima di terminare stampa un messaggio.

Al termine delle due funzioni (*e dei relativi thread*) rimane solo il thread relativo al main che termina anch'esso.