# C:\Users\stefano.maggi.CONBIPELSPA\Desktop\prom38.png

**Date**: 17/07/2012

**Procedure:** Programmazione C - Il MultiThreading in ambiente Windows

**Source:** [**LINK**](http://www.swprog.com/articoli/cmultithreadwin.php)

**Permalink:** [**LINK**](http://heelpbook.altervista.org/2012/programmazione-c-il-multithreading-in-ambiente-windows/)

**Created by:** HeelpBook Staff

**Document Version:** 1.0

# [**Programmazione C – Il MultiThreading in ambiente Windows**](http://heelpbook.altervista.org/2012/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.

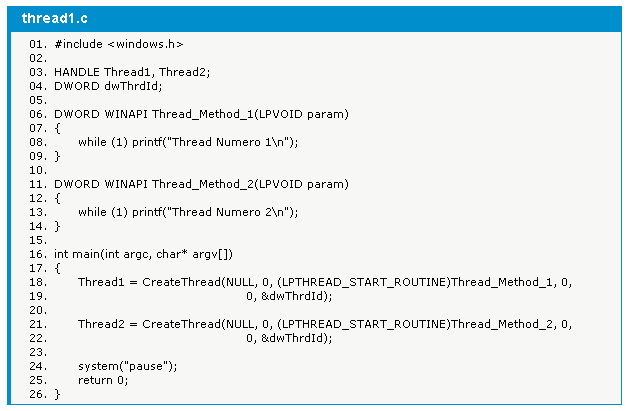
**Creare il Thread**

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);

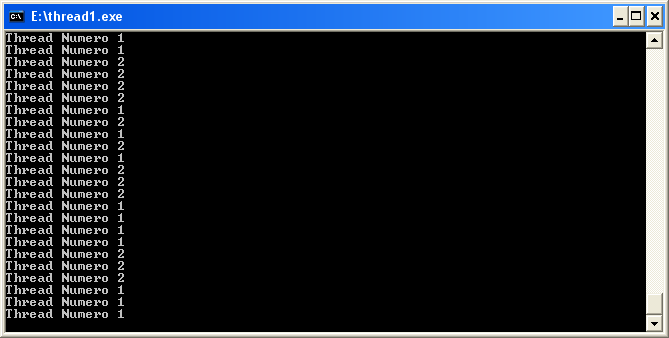
Per creare un thread si usa la funzione **CreateThread**.  
  
Nel programma che segue si creeranno due thread, in ciascuno dei quali saranno stampati dei messaggi sul video.



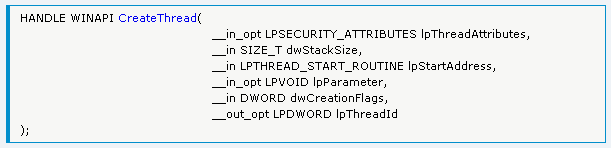
Vengono definite due funzioni, **Thread\_Method\_1** e **Thread\_Method\_2**.

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.



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.



**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.

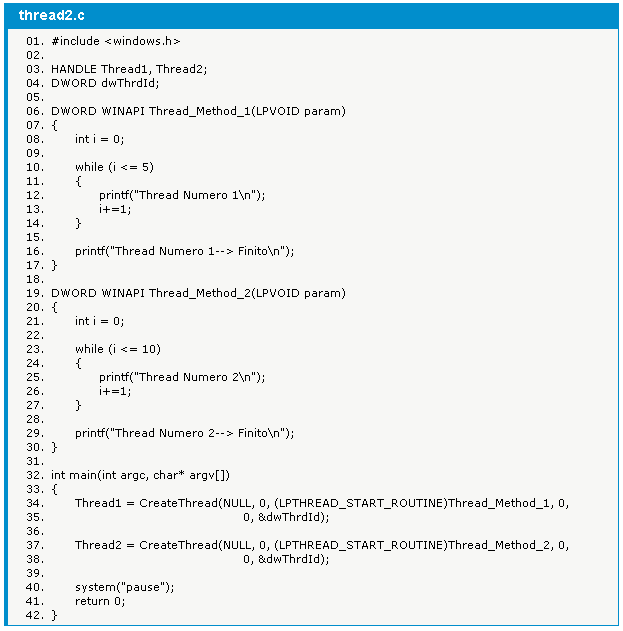
**lpThreadId**

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

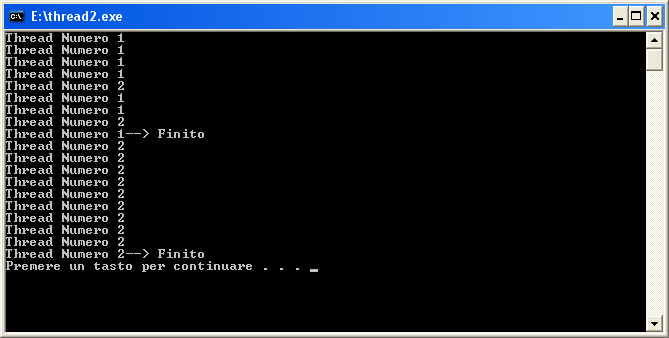
Se la funzione ha successo viene restituito l’handle del nuovo thread, altrimenti viene restituito **NULL**.

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.



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



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.