Threads in Windows

Cosa è un Thread?

- Un thread è un percorso di esecuzione
- Un thread accede a tutte le risorse del processo in cui è contenuto
- Il ThreadID contraddistingue un thread nel sistema operativo, indipendentemente dal processo ospitante
- Un thread esiste fino a che
  - il percorso di esecuzione termina
  - viene chiuso con un ExitThread
  - viene chiuso/distrutto il processo ospitante

Creazione e Terminazione di un Thread

Ogni Sistema Operativo ha nei vari linguaggi di programmazione o ambiente di sviluppo delle funzioni specifiche per creare e terminare un thread.

In C/C++ esistono funzioni per creare thread sia in DOS che in WIN32:

DOS	Win32
_beginthread _endthread _beginthreadex _endthreadex	CreateThread TerminateThread

_beginthread

In c/c++ in ambiene DOS per creare un thread si usa la funzione _beginthread che ha la seguente sintassi

```
uintptr_t _beginthread(
void( __cdecl *start_address ) ( void * ),
unsigned stack_size,
void *arglist
);
```

con le tre variabili di input rispettivamente:

1. void(__cdecl *start_address)(void *) che rappresenta la funzione chiamata,
2. unsigned stack_size che è la grandezza dello stack di memoria
3. void *arglist che rappresenta gli argomenti da passare alla funzione.

__cdecl è la calling convention del C/C++ ossia, come il C/C++ implementa una chiamata a una funzione e come la funzione debba ritornare al chiamante.

Quindi una calling convention stabilisce come gli argomenti sono passati e i valori ritornati alle funzioni, specifica anche come i nomi delle funzioni vengono "decodificati".

La calling convention diventa molto importante quando ad esempio vogliamo linkare dei moduli C/C++ con del codice asm.

Indipentemente dalla calling convention scelta, succedono le seguenti cose:

1. Tutti gli argomenti sono estesi a 4 bytes (su win32) e messi nelle appropriate locazioni di memoria.
2. L'esecuzione del programma va all'indirizzo della funzione.
3. All'interno della funzione i registri ESI, EDI, EBX, and EBP vengono salvati sullo stack, e la parte di codice che si occupa di fare questo viene chiamato “prologo” e di solito viene generato dal compilatore.
4. Viene eseguito il codice opportuno della funzione e il valore di ritorno viene messo nel registro EAX.
5. I registri ESI, EDI, EBX, and EBP sono ripristinati, e Il codice che si occupa di fare questo viene chiamato "epilogo" e come il prologo, nella maggioranza dei casi viene generato dal compilatore.

6. Gli argomenti vengono rimossi dallo stack, operazione chiamata "stack cleanup" che potrebbe essere fatta sia all'interno della funzione chiamata (callee) che dal chiamante (caller) a seconda della calling convention usata.

Le principali calling convention sono:

Keyword	**Stack cleanup**	**Passaggio dei Parametri**
__cdecl	Caller	Pusha i parametri sullo stack in ordine inverso (da dx a sx), è la convenzione default per i programmi scritti in C/C++
__stdcall	Callee	Pusha i parametri sullo stack in ordine inverso (da dx a sx), è la convenzione usata di solito per chiamare funzioni Win32 API, infatti WINAPI non è altro che un #define WINAPI __stdcall

__fastcall	Callee	Parametri passati nei registri, poi pushati sullo stack, ossia convenzione implica che gli argomenti devono essere piazzati nei registri invece che sullo stack laddove possibile. Poichè si lavora sui registri è implicito che si vada più veloce (fast)
Thiscall	Callee	Parametri pushati sullo stack; il puntatore **this** salvato in ECX, è la calling convention default per le funzioni membro di classe C++
PASCAL	Callee	usata in Win16 . pusha i parametri da sx verso dx, al contrario della convenzione C
BASIC, FORTRAN	Callee	(che per l'assemblatore sono sinonimi a PASCAL ma mette i nomi in uppercase)

La funzione _beginthread ritorna un ntptr_t (long integer o __int64, dipende dalla piattaforma) un handle al thread creato o1L se vi è un errore, e in questo caso errno assume:

EAGAIN se si è raggiunto il limite massimo di threads

EINVAL se uno degli argomenti non è valido o se lo stack ha un valore non corretto.

Consideriamo il seguente semplice esempio in cui si crea un thread che conta finchè nel main non si preme il tasto return

```
#include <process.h> /* _beginthread, _endthread */
#include <stdio.h>
void StartPrimo_THread(void *b);
unsigned long c=0;
int fine=0;
void main()
{char s[2];
 int err=_beginthread( StartPrimo_THread, 0, NULL);
 if(err==-1) printf("errore ");
```

```
 else
  {printf("return per terminare ");
  getchar();
   fine=1;
   printf("%d",c);
  }
getchar();
}

void StartPrimo_THread(void *b)
{ while(fine==0) c++;
 _endthread();
}
```

Consideriamo adesso un'altro esempio più complesso che illustra la creazioni di thread multipli usando le funzioni: _beginthread _endthread.

Questo programma richiede alcune funzioni per la gestione della console. Per attivare la Multi-Threaded runtime library in

visual c++ 6.0 la si deve selezionare nel dialog box del compiler Project Settings.

Nell'esempio si creano tanti threads che visualizzano nella console una lettera scelta e posizionata a caso, muovendola nello schermo

```
#include <windows.h>
#include <process.h> /* _beginthread, _endthread */
#include <stddef.h>
#include <stdlib.h>
#include <conio.h>
void Bounce( void *ch );
void CheckKey( void *dummy );
/* funzione random tra min and max. */
#define GetRandom( min, max ) ((rand() % (int)(((max) + 1) - (min))) +
(min))
BOOL repeat = TRUE; /*BOOL tipo di dato booleano di Window*/
HANDLE hStdOut; /* Handle per la  console window */
CONSOLE_SCREEN_BUFFER_INFO csbi;
/* Console information structure */

void main()
```

```
{CHAR ch = 'A';
  hStdOut = GetStdHandle( STD_OUTPUT_HANDLE );
  /*prede le informazioni delle colonne e delle righe della console di
testo */
  GetConsoleScreenBufferInfo( hStdOut, &csbi );
  _beginthread( CheckKey, 0, NULL );
  /* Loop finchè  CheckKey temina. */
   while( repeat )
   {_beginthread( Bounce, 0, (void *) (ch++) );
     Sleep( 1000L );
   }
}
/*funzione CheckKey  */
void CheckKey( void *dummy )
 {_getch();
  repeat = 0;
 }
/* Bounce - Thread che crea e muove una lettera nello schermo
Il parametro ch è la lettera da visualizzare */
void Bounce( void *ch )
 {char blankcell = 0x20;/*carattere vuoto */
   char blockcell = (char) ch;
   BOOL first = TRUE;
   COORD oldcoord, newcoord;/*cordiante della console*/
   DWORD result;
   /* Numero random generato a partire da _threadid  */
```

```
   srand( _threadid );
   newcoord.X = GetRandom( 0, csbi.dwSize.X - 1 );
   newcoord.Y = GetRandom( 0, csbi.dwSize.Y - 1 );
   while( repeat )
    {Sleep( 100L );
      if( first ) first = FALSE;
     else
    WriteConsoleOutputCharacter( hStdOut, &blankcell, 1, oldcoord,
&result );
     WriteConsoleOutputCharacter( hStdOut, &blockcell, 1, newcoord,
&result );
     oldcoord.X = newcoord.X;oldcoord.Y = newcoord.Y;
     newcoord.X += GetRandom( -1, 1 );
     newcoord.Y += GetRandom( -1, 1 );
 /* Correzioni nel caso che si vada fuori dallo schermo  */
     if( newcoord.X < 0 ) newcoord.X = 1;
     if( newcoord.X == csbi.dwSize.X )
        newcoord.X = csbi.dwSize.X - 2;
     if( newcoord.Y < 0 ) newcoord.Y = 1;
     if( newcoord.Y == csbi.dwSize.Y )
        newcoord.Y = csbi.dwSize.Y - 2;
     }
  _endthread();
}
```

_beginthreadex

La funzione _beginthread non da la possibilità di controllare l'andamento del thread né di utilizzare le funzioni di attesa di terminazione come waitforsingleobject. Per qusro motivo dalla versione 2003 è possibile utilizzare una versione molto più completa di beginthread , la funzione beginthreadex che ha la seguente sintassi

```
uintptr_t _beginthreadex(
void *security,
unsigned stack_size,
void( __cdecl *start_address ) ( void * ),
   void *arglist,
   unsigned initflag,
   unsigned *thrdaddr
);
```

con le variabili di input rispettivamente:

1. void **security* Puntatore a una struttura tipo SECURITY_ATTRIBUTES che determina se l'handle può essere ereditato da un processo figlioc. Se NULL, l'handle non può esserlo, e deve essere NULL per applicazioni Windows 95.
2. unsigned stack_size che è la grandezza dello stack di memoria
3. (__cdecl *start_address)(void *) che rappresenta la funzione chiamata,
4. void *arglist che rappresenta gli argomenti da passare alla funzione.
5. Initflag che rappresentalo stato iniziale del thread (0 running o CREATE_SUSPENDED)
6. Thrdaddr thread identifier.

Consideriamo il seguente semplice esempio in cui si usa handle restituito da _beginthreadex con la funzione waitforsingleobject.

```
#include <windows.h>
#include <stdio.h>
#include <process.h>
unsigned Counter;
unsigned __stdcall SecondThreadFunc( void* pArguments )
{printf( "In second thread...\n" );
 while ( Counter < 1000000 )Counter++;
 _endthreadex( 0 );
 return 0;
}

int main()
{HANDLE hThread;
 unsigned threadID;
 printf( "Creating second thread...\n" );
 // Create the second thread.
 hThread = (HANDLE)_beginthreadex( NULL, 0,
&SecondThreadFunc, NULL, 0, &threadID );
```

```
    // Wait until second thread terminates. If you comment
out the line
    // below, Counter will not be correct because the
thread has not
    // terminated, and Counter most likely has not been
incremented to
    // 1000000 yet.
    WaitForSingleObject( hThread, INFINITE );
    printf( "Counter should be 1000000; it is-> %d\n",
Counter );
    // Destroy the thread object.
    CloseHandle( hThread );
   getchar();
}
```

CreateThread

In ambiente windows per creare un thread si utilizzano le librerie SDK, ed in c++ si utilizza la funzione CreateThread la cui sintassi è:

```
HANDLE CreateThread(
LPSECURITY_ATTRIBUTES lpThreadAttributes,
// pointer to security attributes
DWORD dwStackSize,
// initial thread stack size
LPTHREAD_START_ROUTINE lpStartAddress,
// pointer to thread function
 LPVOID lpParameter,
 DWORD dwCreationFlags, // creation flags
 LPDWORD lpThreadId // pointer to receive thread ID
 );
```

che è molto simile a _beginthreadex

Se tutto va bene la funzione restituisce un puntatore Handle al thread, viceversa restituisce NULL.

Le variabili sono rispettivamente:

lpThreadAttributes Puntatore ad una struttura di tipo SECURITY_ATTRIBUTES che determina se l'handle restituito può essere passato ad un processo figlio,se NULL, l'handle non può essere passato ad un processo figlio.

dwStackSize un DWORD che rappresenta la grandezza dello stack di memoria,

lpStartAddress Un puntatore alla funzione chiamata

lpParameter Un puntatore a VOID ai parametri della funzione

dwCreationFlags Un flag addizionale che specifica come il trhead si comporterà inizialmante, se il suo valore è CREATE_SUSPENDED, il thread è creato in uno stato di attesa, è si attiverà solo quando ci sarà una chiamata alla funzione ResumeThread. Se è zero, il thread si attiverà immediatamente dopo la sua creazione.

lpThreadId Il puntatore che riceverà ID del Thread

TerminateThread

La funzione TerminateThread ha la seguente sintassi:

```
BOOL TerminateThread(
HANDLE hThread,// handle to the thread
DWORD dwExitCode // exit code for the thread
);
```

variabili sono rispettivamente:

HANDLE hThread handle del thread

DWORD dwExitCode Specifica il codice di uscita del thread. Si utilizza la funzione GetExitCodeThread per ritrovare il valore di uscita.

Priorità di un thread

Le funzioni che regolano le priorità di un thread sono:

SetThreadPriority

GetThreadPriority

La funzione SetThreadPriority assegna una priorità al thread ed ha la seguente sintassi:

```
BOOL SetThreadPriority(
  HANDLE hThread, // handle to the thread
  int nPriority   // thread priority level
);
```

I parametri sono:

hThread Handle del thread

nPriority Specifica il valore della priorità che può assumere uno dei seguenti valori:

THREAD_PRIORITY_ABOVE_NORMAL 1 sopra la priorità normale

THREAD_PRIORITY_BELOW_NORMAL 1 sotto la priorità normale

THREAD_PRIORITY_HIGHEST 2 sopra la priorità normale

THREAD_PRIORITY_IDLE

indica un livello di priorità base di 1 per i processi IDLE_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS, o HIGH_PRIORITY_CLASS e livello di priorità

base di 16 per i processi REALTIME_PRIORITY_CLASS

THREAD_PRIORITY_LOWEST 2 sotto la priorità normale

THREAD_PRIORITY_NORMAL priorità normale

THREAD_PRIORITY_TIME_CRITICAL

indica un livello di priorità base di 15 per i processi IDLE_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS, o HIGH_PRIORITY_CLASS e livello di priorità base di 31 per i processi REALTIME_PRIORITY_CLASS

GetThreadPriority funzione che trova la priorità di thread ed ha la seguente sintassi

```
int GetThreadPriority(
  HANDLE hThread, // handle to  thread
  );
```

Sospensione e risveglio di un thread

Le funzioni che regolano la sospensione e risveglio di un thread sono

SuspendThread

ResumeThread

Ogni thread ha un contatore delle sospensioni inizialmente posto a zero, la funzioni SuspendThread incrementa di uno il contatore, il thread è sospeso fino a quando il contatore non ritorna zero.

La funzione ResumeThread viceversa diminuisce il contatore e se esso è zero il thread sarà riattivato.

Le funzioni se hanno sucesso, ritornano il valore precedente del contatore, viceversa se la funzione fallisce ritorna il valore (DWORD) -1 e per avere ulteriori informazioni sull'errore si deve richiamare la funzione GetLastError.

La sintassi delle funzioni è la seguente

```
DWORD SuspendThread(
  HANDLE hThread, // handle to the thread
 );
DWORD ResumeThread(
  HANDLE hThread, // handle to the thread
 );
```

Diamo ora un esempio di utilizzo delle funzioni appena descritte.

Il programma sottostante genera due processi identici che stampano i valori da 1 a 1000 con priorità distinte.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
DWORD finito (LPDWORD lpdwParam)
{int *fine; fine=(int *) lpdwParam;
  _getch();
  *fine=1;
  return 0;
}

DWORD mythread (LPDWORD lpdwParam)
{int *proc;
  proc=(int *) lpdwParam;
  while(1)
   {for(int i=0;i<1000;i++)
     printf("sono il thread %d: %d\n",*proc,i);
   }
}
```

```
void main ()
{int i,j,fine=0;
 HANDLE hThreadT1,hThreadT2,hfineThreads;
 DWORD dwThreadId1,dwThreadId2,dwThreadIdFine;
 hfineThreads=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)
finito,(LPVOID) &fine,0, &dwThreadIdFine);
  if (hfineThreads!=NULL)
    {i=1;
      hThreadT1=CreateThread(NULL,0,
(LPTHREAD_START_ROUTINE)mythread,(LPVOID)&i,CREATE_SUSPENDED ,
&dwThreadId1);
     SetThreadPriority(hThreadT1,THREAD_PRIORITY_LOWEST);
     j=2;
     hThreadT2=CreateThread(NULL,0,
(LPTHREAD_START_ROUTINE)mythread,(LPVOID) &j,CREATE_SUSPENDED ,
&dwThreadId2);
    SetThreadPriority(hThreadT2,THREAD_PRIORITY_HIGHEST);
    ResumeThread(hThreadT2);
    ResumeThread(hThreadT1);
    WaitForSingleObject( hfineThreads, INFINITE );
    TerminateThread(hThreadT1,0);
    TerminateThread(hThreadT2,0);
   getchar();}}
```