Sincronizzazione threads in Window

Se più threads si contendono una risorsa, deve esistere un meccanismo che consenta di sincronizzare i thread, e che quindi preveda che un thread, che aspetta il rilascio di una risorsa da parte di un altro thread, vada in uno stato di attesa o wait.

Gli oggetti di sincronizzazione hanno lo scopo quindi di consentire al programmatore di attivare tale meccanismo, sotto windows in C/C++ questi oggetti sono:

- Critical Section
- Mutex
- Semaphore
- Event

Critical Section

Un oggetto critical section è un oggetto di sincronizzazione utilizzati solo ed esclusivamente dai trheads di un unico processo. Un oggetto CS allocato da un processo dichiarando delle variabili di tipo CRITICAL_SECTION, può appartenere ad un solo un thread alla volta.

Prima di utilizzare le variabili di tipo CRITICAL_SECTION devono essere inizializzate attraverso la funzione InitializeCriticalSection.

Un thread usa la fuzione EnterCriticalSection per richiedere la proprietà esclusiva della critical section, e la funzione LeaveCriticalSection per rilasciarla.

Se la CS è posseduta da un'altro thread, la funzione EnterCriticalSection aspetta indefinitamente il rilascio della CS, quindi si deve stare attenti a possibili DeadLock.

Nel seguente esempio due thread accedono a due Critical Section incrociate causando un DeadLock.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
CRITICAL_SECTION cs1,cs2;
long WINAPI ThreadF1(long);
long WINAPI ThreadF2(long);
void main()
 {InitializeCriticalSection(&cs1);
   InitializeCriticalSection(&cs2);
   DWORD iThreadID1,iThreadID2;
   HANDLE thread1,thread2;
   thread1=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)ThreadF1,
NULL,0,&iThreadID1);
```

```
   thread2=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)ThreadF2,
NULL,0,&iThreadID2);
  getchar();
  TerminateThread(thread1,0);
  TerminateThread(thread2,0);
}

long WINAPI ThreadF1(long lParam)
{while(TRUE)
  {EnterCriticalSection(&cs1);
    printf("\nThread1 entra nella CS 1 ma non nella 2.");
    EnterCriticalSection(&cs2);
    printf("\nThread1 entra nella CS 1 and 2!");
    LeaveCriticalSection(&cs2);
    printf("\nThread1 lascia la CS 2 ma non la 1.");
    LeaveCriticalSection(&cs1);
    printf("\nThread1 lascia entrambe le CS...");
    Sleep(50);
  }
 return(0);
}

long WINAPI ThreadF2(long lParam)
```

```
{while(TRUE)
  {EnterCriticalSection(&cs2);
    printf("\nThread2 entra nella CS 2 ma non nella 1.");
    EnterCriticalSection(&cs1);
    printf("\nThread2 entra nella CS2 and 1!");
    LeaveCriticalSection(&cs1);
    printf("\nThread2 lascia la CS 1 ma non la 2.");
    LeaveCriticalSection(&cs2);
    printf("\nThread2 lascia entrambi le CS...");
    Sleep(50);
}
return(0);

}
```

Come esercitazione vi propongo di reparare un programma che simuli Il problema dei filosofi a tavola con le CS.

Si ricorda che per mangiare i 5 filosofi devono avere disponibili sia le bacchette alla loro destra che alla loro sinistra. Si generino,

quindi, cinque thread che rappresentano i 5 filosofi, e 5 Critical Section che rappresentano le 5 bacchette.

Alla fine della simulazione si vuole sapere quante volte hanno mangiato i 5 filosofi.

Funzioni di Wait

Prima di parlare dei restanti oggetti di sincronizzazione Mutex, Semaphore e Event tutti oggetti di sincronizzazione utilizzati per sospendere un thread fino a che uno o più di questi oggetti non diventano disponibili o "segnalati".

Le seguenti funzioni di Wait sono utilizzate a tale

- WaitForSingleObject
- WaitForMultipleObjects
- MsgWaitForMultipleObjects

WaitForSingleObject

La funzione WaitForSingleObject controlla un singolo oggetto, e la sua sintassi è:

```
DWORD WaitForSingleObject(
HANDLE h, // handle all'oggeto di cui si aspetta il rilascio
DWORD Milliseconds // time-out  in millisecondi
);
```

Con h handle dell'oggetto, se l'oggetto è disponibile o segnalato (signaled), la funzione termina ed il processo esecutivo procede. Viceversa se l'oggetto non è disponibile (not signaled) il processo va in stato di wait fino a quando lo stato dell'oggetto diventa signaled o l'intervallo di time-out è raggiunto.

Milliseconds parametro DWORD che indica i millisecondi del time-out o INFINITE

La funzione ritorna i seguenti valori:

WAIT_OBJECT_0 se lo stato dell'oggetto è diventato signaled

WAIT_TIMEOUT se lo stato dell'oggetto non è diventato signaled nell'intervallo di tempo definito nel secondo parametro

WAIT_ABANDONED se l'oggetto non è diventato signaled ed il thread in cui l'oggetto è stato dichiarato è terminato.

WaitForMultipleObjects

La funzione WaitForMultipleObjects controlla più oggetti di sincronizzazione, e la sua sintassi è:

```
DWORD WaitForMultipleObjects(
 DWORD nCount, // numero di oggetti da controllare
 CONST HANDLE *lpHandles, // vettore di oggetti handle
 BOOL fWaitAll, // wait flag
 DWORD Milliseconds // time-out  in millisecondi
);
```

Con nCount parametro che indica il numero di oggetti che si vogliono monitorare.

lpHandles puntatore al vettore di handle degli oggetti da monitorare.

fWaitAll parametro booleano che:

- se TRUE la funzione aspetta finché lo stato di tutti gli oggetti diventa signaled
- se FALSE, la funzione aspetta che uno solo degli oggetti divenga signaled, e in questo caso il valore di ritorno indica l'oggetto il cui stato è cambiato.

Milliseconds parametro DWORD che indica i millisecondi del time-out o INFINITE

La funzione ritorna i seguenti valori

WAIT_OBJECT_n con n= 0..Count-1 se lo stato dell'oggetto è diventato signaled.

WAIT_TIMEOUT se lo stato dell'oggetto/i non è diventato signaled nell'intervallo di tempo definito nel secondo parametro.

WAIT_ABANDONED_n con n= 0..Count-1 se l'oggetto non è diventato signaled ed il thread in cui l'oggetto è stato dichiarato è stato terminato.

Quindi l'indice dell'oggetto diventato segnalato sarà i=WAIT_OBJECT_n-WAIT_OBJECT_0.

MsgWaitForMultipleObjects

La funzione MsgWaitForMultipleObjects è molto simile alla funzione WaitForMultipleObjects eccetto al fatto che ha un ulteriore parametro con cui specificare un evento da attendere.

L'evento può essere l'input da tastiera (QS_KEY) , o un input da mouse (QS_MOUSEBUTTON o QS_MOUSEMOVE) ect.

la sua sintassi molto simile a WaitForMultipleObjects è:

```
DWORD MsgWaitForMultipleObjects(
  DWORD nCount, // numero di oggetti da controllare
  LPHANDLE pHandles, // vettore di oggetti handle
  BOOL fWaitAll, // wait flag
  DWORD dwMilliseconds, // time-out in millisecondi
  DWORD dwWakeMask // tipo di input da aspettare
  );
```

Quindi se il parametro fWaitAll booleano è:

- TRUE la funzione aspetta finché lo stato di tutti gli oggetti diventa signaled e si e verificato uno degli eventi specificati nella variabile dwWakeMask
- se FALSE, la funzione aspetta che uno solo degli oggetti divenga signaled, o si e verificato uno degli eventi specificati nella variabile dwWakeMask e in questo caso il valore di ritorno indica l'oggetto il cui stato è cambiato.

La variabile dwWakeMask può essere la combinazione in OR (|) di uno dei seguenti parametri:

QS_ALLEVENTS (0x04BF)

Un input WM_TIMER, WM_PAINT, WM_HOTKEY, o un qualsiasi messaggio postato nella coda, equivalente alla combinazione QS_INPUT | QS_POSTMESSAGE | QS_TIMER | QS_PAINT | QS_HOTKEY

QS_ALLINPUT (0x04FF)

Qualsiasi messaggio presente nella coda, equivalente alla combinazione QS_INPUT | QS_POSTMESSAGE | QS_TIMER | QS_PAINT | QS_HOTKEY | QS_SENDMESSAGE

QS_ALLPOSTMESSAGE (0x0100)

Qualsiasi messaggio postato nella coda

QS_HOTKEY (0x0080)

Un input di tipo WM_HOTKEY

QS_INPUT (0x407)

Un messaggio di input postato nella coda, equivalente alla combinazione QS_MOUSE | QS_KEY | QS_RAWINPUT

QS_KEY (0x0001)

Un WM_KEYUP, WM_KEYDOW, WM_SYSKEYUP, WM_SYSKEYDOWN messaggio di input postato nella coda

QS_MOUSE (0x0006)

Un WM_MOUSEMOVE, WM_RBUTTONDBLCLK, WM_RBUTTONUP, WM_RBUTTONDOWN, WM_LBUTTONDBLCLK WM_LBUTTONUP, WM_LBUTTONDOWN, messaggio di input postato nella coda, equivalente alla combinazione QS_MOUSEMOVE | QS_MOUSEBUTTON

QS_MOUSEBUTTON(0x0004)

Un WM_RBUTTONDBLCLK, WM_RBUTTONUP, WM_RBUTTONDOWN, WM_LBUTTONDBLCLK WM_LBUTTONUP, WM_LBUTTONDOWN, messaggio di input postato nella coda

QS_MOUSEMOVE(0x0002)

Un WM_MOUSEMOVE, messaggio di input postato nella coda

QS_PAINT(0x0020)

Un WM_PAINT, messaggio di input postato nella coda

QS_POSTMESSAGE(0x0008)

Un messaggio postato nella coda

QS_RAWINPUT(0x0400)

Un messaggio da un qualsiasi device

QS_SENDMESSAGE(0X0040)

Un messaggio inviato da un altro thread

QS_SENDMESSAGE(0X0040)

Un messaggio tipo WM_TIMER