I Mutex

Gli oggetti Mutex, sono dei flag che coordinano l'esecuzione di operazioni mutualmente esclusive.

Solo un thread può detenere un Mutex, gli altri thread vengono sospesi su una Wait fino a che il Mutex non viene rilasciato.

Le funzioni per creare e gestire un oggetto mutex sono:

- CreateMutex
- OpenMutex
- ReleaseMutex

CreateMutex

La funzione CreateMutex crea un nuovo oggetto Mutex o ne apre uno esistente , e la sua sintassi è la seguente:

```
HANDLE CreateMutex(
LPSECURITY_ATTRIBUTES lpMutexAttributes,
// pointer to security attributes
BOOL bInitialOwner,
// flag che determina l'iniziale proprietario
LPCTSTR lpName // nome del mutex);
```

lpMutexAttributes

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

se è NULL, l'handle non può essere passato ad un processo figlio.

bInitialOwner

Specifica il proprietario iniziale dell'oggetto. Se questo valore è TRUE il thread chiamante ottiene la proprietà del mutex, altrimenti per ottenerla bisognerà effettuare una chiamata alla funzione ReleaseMutex.

lpName

nome del mutex

OpenMutex

La funzione OpenMutex apre un oggetto named mutex già esistente, e la sua sintassi è la seguente:

```
HANDLE OpenMutex(
DWORD dwDesiredAccess, // modalità di accesso al mutex
BOOL bInheritHandle, // inherit flag
LPCTSTR lpName // nome del mutex);
```

dwDesiredAccess

Specifica l'accesso richiesto per l'oggetto mutex è può essere una combinazione dei seguenti valori:

- MUTEX_ALL_ACCESS tutti i tipi di accesso
- SYNCHRONIZE abilita l'uso del mutex con una delle funzioni di wait per acquisire il mutex, o la funzione ReleaseMutex per rilasciarlo.

bInheritHandle Specifica se l'handle restituito può essere passato ad un processo figlio, se NULL, l'handle non può essere passato ad un processo figlio.

lpName Il nome del mutex

ReleaseMutex

La sintassi della funzione ReleaseMutex è la seguente

```
BOOL ReleaseMutex( HANDLE hMutex);
```

essa rilascia il parametro hmutex , restituisce TRUE in caso di sucesso, viceversa restituisce FALSE

Esempio i due processi somma e sottrai che si contendono la risorsa num

```
#include <stdio.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include <ctype.h>
#define GetRandom( min , max )
 ((rand()%(int) ( ( ( max ) + 1)-(min)))+(min))
DWORD finito (LPDWORD lpdwParam)
{int *fine;
 fine=(int *) lpdwParam;
  _getch();
  *fine=1;
 return 0;
}
DWORD Somma (LPDWORD lpdwParam)
{int *num,fine=0;
 num=(int *)lpdwParam;
 HANDLE hMutex;
 Mutex=OpenMutex(EVENT_MODIFY_STATE|SYNCHRONIZE,
FALSE,"SommaMutex");
```

```
 printf ("Somma start\n");
 while(fine==0)
  {WaitForSingleObject(hMutex,INFINITE);
    num[0]+=num[1]; printf ("Somma %d \n",num[0]);
    ReleaseMutex(hMutex);
     Sleep(10);
   }return 0;
}
////////////////////////////////////////
DWORD Prodotto (LPDWORD lpdwParam)
{int *num,fine=0;
  num=(int *)lpdwParam;
  HANDLE hMutex;
hMutex=OpenMutex(EVENT_MODIFY_STATE|SYNCHRONIZE,
FALSE,"ProdottoMutex");
  printf ("Prodotto start\n");
  while(fine==0)
  {WaitForSingleObject(hMutex,INFINITE);
     num[0]*=num[2];
     printf ("Prodotto %d \n",num[0]);
     ReleaseMutex(hMutex);
     Sleep(10);
   } return 0;
```

```
}
////////////////////////////////////////
void main ()
{int i,fine=0;
 int numeri[3];
 HANDLE hMutexsomma, hMutexprodotto;
 HANDLE, hThreads[2], hfineThreads;
 DWORD dwThreadId0,dwThreadId1, dwThreadId2;
 hMutexsomma=CreateMutex(NULL,TRUE,"SommaMutex");
 hMutexprodotto=CreateMutex(NULL,TRUE,"ProdottoMutex")
 if((hMutexsomma==NULL)||(hMutexprodotto==NULL))
 fprintf(stderr,"Errore CreateMutex\n");
 else
  {hfineThreads=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)
finito, (LPVOID) &fine,0, &dwThreadId0);
    if (hfineThreads==NULL)
      fprintf(stderr,"Errore CreateThread finito error\n");
   else
    {hThreads[0]=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)
Somma, (LPVOID)numeri,CREATE_SUSPENDED, &dwThreadId1);
     if (hThreads[0]==NULL)
       fprintf(stderr,"Errore CreateThread Somma error\n");
     else
```

```
      {hThreads[1]=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)
Prodotto, (LPVOID) numeri, CREATE_SUSPENDED, &dwThreadId2);
       if (hThreads[1]==NULL)
        fprintf(stderr,"Errore CreateThread Prodotto error\n");
       else
        {i=1;
         while(fine==0)
          {srand( (unsigned)time( NULL ) );
           WaitForSingleObject(hMutexsomma,INFINITE);
         /* aspetta che il mutexsomma sia libero inizialmente il
mutex è libero*/
           WaitForSingleObject(hMutexprodotto,INFINITE);
         /* aspetta che il mutexprodotto sia libero inizialmente
il mutex è libero*/
           numeri[0]=GetRandom( 1, 100 );
           printf("primo numero della terna Numero %d
e':%d\n",i,numeri[0]);
           numeri[1]=GetRandom( 1, 100 );
           printf("secondo numero della terna Numero %d
e':%d\n",i,numeri[1]);
           numeri[2]=GetRandom( 1, 100 );
          printf("terzo numero della terna Numero %d
e':%d\n",i++,numeri[2]);
```

```
            ResumeThread( hThreads[0]);
            ReleaseMutex(hMutexsomma);
            Sleep(5);
           WaitForSingleObject(hMutexsomma,INFINITE);
           /* aspetta che il mutexsomma sia libero*/
           SuspendThread( hThreads[0]);
           ReleaseMutex(hMutexprodotto);
           ResumeThread( hThreads[1]);
           Sleep(5);
           WaitForSingleObject(hMutexprodotto,INFINITE);
           /* aspetta che il mutexsomma sia libero*/
           SuspendThread( hThreads[1]);
           ReleaseMutex(hMutexsomma);
          }
       TerminateThread(hThreads[0],0);
       TerminateThread(hThreads[0],1);
       }
    }
  }
 }
}
```

Esempio compito marzo 2012 già visto in linux

```
#include <stdio.h>
#include <stdlib.h>  //x srand-rand-exit-system-malloc
#include <windows.h> //x WIN32 Api
#include <conio.h>   //getch-putch [I/O basso livello]
#include <time.h>    //x time (seme della funzione srand)


/*per la generazione dei numeri casuali*/
#define Randomize(x) srand((unsigned)time(NULL)*(x))  //x inizializzare il seme
//il sema dipende anche da un altro valore, in modo da ottenere semi diversi
con maggiore probabilità
#define GetRandom( min, max ) ((rand() % (int)(((max) + 1) - (min))) + (min))

#define n 10 //numero giocatori
#define k 5 //numero tiri di ogni giocatore

/*definizione prototipi*/
DWORD mythread (LPDWORD lpdwParam);
void printfp(char *stringa); //prototipo printf protetta da mutex

HANDLE giocatore[n]; //puntatori handle al Thread e al mutex della printfp
HANDLE palleVerdi[2]; //puntatore handle palle Verdi
HANDLE palleRosse[2]; //puntatore handle palle Verdi
HANDLE palleNere[3]; //puntatore handle palle Verdi
HANDLE pista, punt, Mut; //puntatore handle per i mutex di protezione pista-
vettore punteggi-printfp
```

```
int punteggio[n];  //vettore che conterrà il punteggio di ogni giocatore

//MAIN
int main ()
{

    int i,num_casuale,fine=0,aux[n],j,max=0,errore;
    int nome[n];
    HANDLE hfineThreads;        //puntatori handle al thread finito
   errore=0;
  //creazione vettore di semafori per le palle
    for(i=0;i<2;i++)
    {if( (palleVerdi[i]=CreateMutex(NULL,FALSE,NULL))==NULL) errore=1; //Mutex
per le 2 palle verdi
     if( (palleRosse[i]=CreateMutex(NULL,FALSE,NULL))==NULL) errore=2; //Mutex
per le 2 palle rosse
     if( (palleNere[i]=CreateMutex(NULL,FALSE,NULL))==NULL) errore=3; //Mutex
per le 2 palle nere
     }
    if( (palleNere[2]=CreateMutex(NULL,FALSE,NULL))==NULL) errore=3;
//Ulteriore  palla nera


    if(errore>0)
    {printf("Errore Creazione Mutex Palle!!!\n");exit(0);}

  //creazione mutex per protezione printfp-pista-vettore punteggi
```

```
  Mut=CreateMutex(NULL,FALSE,NULL);
  pista=CreateMutex(NULL,FALSE,NULL);
  punt=CreateMutex(NULL,FALSE,NULL);
  if (Mut==NULL || pista==NULL || punt==NULL)
  {printf("Errore Creazione Mutex!!!\n");exit(0);}

  //inizializzo i vettori
  for(i=0;i<n;i++)
  {   nome[i]=i;
      punteggio[i]=0;
      aux[i]=0; //vettore ausiliario per avere la classifica finale
  }

  printf("Alla fine dell'esecuzione, premere INVIO per avere la classifica
finale!!!\n");
    //creazione thread
    for(i=0;i<n;i++)
    {

    giocatore[i]=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)mythread,(LPVOID)&
nome[i],0,NULL);
        if (giocatore[i]==NULL)
           printf("Errore Creazione Thread!!!\n");

    }
    /*ciclo di ATTESA ATTIVA, si aspetta che fine venga impostato ad 1 tramite
input da tastiera*/
  WaitForMultipleObjects(n,giocatore,TRUE,INFINITE);
```

```
  for(i=0;i<n;i++)
    aux[i]=punteggio[i]; //copio i valori su un vettore ausiliario che serve
per stilare alla fine la classifica

  //Stampo i Risultati finali
  for(i=0;i<n;i++)
    printf("Il giocatore %d ha totalizzato %d punti!\n",i+1,punteggio[i]);

  //chiudo le risorse utilizzate
  CloseHandle(Mut);
  CloseHandle(pista);
  CloseHandle(punt);
  for(i=0;i<2;i++)

{CloseHandle(palleRosse[i]);CloseHandle(palleVerdi[i]);CloseHandle(palleNere[i]
);
  } CloseHandle(palleNere[2]);
  //termino i thread

  system("PAUSE");
    return 0;

}


//funzione chiamata dal thread che esegue le operazioni desiderate
DWORD mythread (LPDWORD lpdwParam)
```

```
{   char buff[255];
    long rish;
    int num,p,j,ris;
    num=(int) *lpdwParam;      /*casting variabile passata come quarto
elemento*/
    Randomize(num);
  p=GetRandom(0,2); //scelgo random il colore della palla del giocatore
(0=verde,1=rossa,2=nera)
  sprintf(buff,"Sono il giocatore %d e ho scelto la palla %d!\n",num+1,p);
  printfp(buff);
  for(j=0;j<k;j++) //ciclo per simulare i k tiri del giocatore
  {switch(p)
    {case 0:rish=WaitForMultipleObjects(2,palleVerdi,FALSE,INFINITE);break;//il
giocatore cerca si acquisire la palla del colore che vuole
     case 1:rish=WaitForMultipleObjects(2,palleRosse,FALSE,INFINITE);break;
     case 2:rish=WaitForMultipleObjects(3,palleNere,FALSE,INFINITE);break;
    }
    WaitForSingleObject(pista,INFINITE);  //il giocatore aspetta che la pista
sia libera per il tiro
    ris=GetRandom(0,10); //risultato del tiro effettuato dal giocatore
    sprintf(buff,"Il giocatore %d nel tiro %d ha ottenuto un punteggio
di:%d!\n",num+1,j,ris);
    printfp(buff);
    WaitForSingleObject(punt,INFINITE);
    punteggio[num]=punteggio[num]+ris;
    ReleaseMutex(punt);
    ReleaseMutex(pista);
    switch(p)
```

```
    {case 0:ReleaseMutex(palleVerdi[rish-WAIT_OBJECT_0]);break;
     case 1:ReleaseMutex(palleRosse[ris-WAIT_OBJECT_0]);break;
     case 2:ReleaseMutex(palleNere[ris-WAIT_OBJECT_0]);break;
    }
    sprintf(buff,"Il giocatore %d ha lasciato la pista e la
palla!\n",num+1,'0',0);
    printfp(buff);Sleep(50);
  }
    ExitThread(TRUE);
}



//printf protetta
void printfp(char *stringa)
{
    WaitForSingleObject(Mut,INFINITE);
    printf(stringa);
    ReleaseMutex(Mut);


}
```

Semafori

I Mutex sono dei flag che coordinano l'esecuzione di operazioni mutualmente esclusive è assume solo i valori 0 e 1. Nel caso che si debba acquisisce una di serie di risorse identiche, in questi casi il Mutex non è più efficace e si utilizzano i Semafori che sono dei Mutex a più valori.

Le funzioni per creare e gestire un oggetto semaphore sono:

- CreateSemaphore
- OpenSemaphore
- ReleaseSemaphore

CreateSemaphore

La funzione CreateSemaphore crea un nuovo oggetto Semaphore o ne apre uno esistente, e la sua sintassi:

```
HANDLE CreateSemaphore(
LPSECURITY_ATTRIBUTES lpMutexAttributes,
// pointer to security attributes
LONG lInitialCount, //valore iniziale
LONG lMaximumCount, //valore massimo
LPCTSTR lpName // nome del mutex);
```

lpMutexAttributes

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

lInitialCount

Specifica il valore iniziale de semaforo. Questo valore deve essere >=0 e <=*lMaximumCount*. Lo stato del semaforo è attivo quando il proprio valore è maggiore di zero, non attivo quando è zero. Il valore è decrementato di uno ogni volta che una funzione di wait di un thread sul semaforo lo acquisisce. Il proprio valore è incrementato di uno specifico valore chiamando la funzione ReleaseSemaphore

lMaximumCount

Specifica il valore massimo del semaforo . Il suo valore deve essere maggiore di zero.

lpName

Il nome del semaforo

OpenSemaphore

La funzione OpenSemaphore apre un oggetto named semaphore già esistente, e la sua sintassi è la seguente:

```
HANDLE OpenSemaphore(
DWORD dwDesiredAccess, // accesso richiesto
BOOL bInheritHandle,
LPCTSTR lpName // Nome);
```

La funzione OpenSemaphore ritorna un handle di un oggetto Semaphore esistente, i parametri sono:

dwDesiredAccess

Specifica l'accesso richiesto per l'oggetto Semaphore è può essere una combinazione dei seguenti valori:

- SEMAPHORE_ALL_ACCESS tutti i flag

- SEMAPHORE_MODIFY_STATE Abilita l'uso della funzioni ReleaseSemaphore per modificare i valore del semaforo.
- SYNCHRONIZE abilita l'uso del semaforo in una delle wait functions

bInheritHandle

Specifica se l'handle restituito può essere passato ad un processo figlio,se è NULL, l'handle non può essere passato ad un processo figlio.

lpName

Il nome del semaforo

ReleaseSemaphore

La funzione ReleaseSemaphore incrementa il valore di un semaforo di una specifica quantità, la sua sintassi è:

```
BOOL ReleaseSemaphore(
HANDLE hSemaphore, // handle del semaforo
LONG lReleaseCount, // incremento
LPLONG lpPreviousCount // valore precedente
);
```

hSemaphore l'handle del semaforo

lReleaseCount di quanto incrementare il semaforo

lpPreviousCount Puntatore che riceverà il valore precedente

Esempio gestione di un parcheggio di auto di 30 posti complessivi distribuiti in 3 piani di 10 posti ciascuno:

```
#include <windows.h>
#include <time.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
void Car(LPDWORD ch);
#define PST 10
#define N 3
#define MAXTHREAD 30
HANDLE THRCAR[MAXTHREAD];//l'array di Thread
HANDLE hfineThreads;
HANDLE hparcheggio[N];//l'array di semafori
HANDLE  Mut; //puntatore handle per i mutex di protezione printfp
#define GetRandom( min, max ) ((rand() % (int)(((max) + 1) -
(min))) + (min))
void printfp(char *stringa); //prototipo printf protetta da mutex
DWORD finito (LPDWORD lpdwParam)
{int *fine;
 fine=(int *) lpdwParam;
```

```
 _getch();
 *fine=1;
return 0;
}
void main()
{DWORD ris;
 int i,ii,fine=0;
 unsigned long car=0;
 Mut=CreateMutex(NULL,FALSE,NULL); //mutex per protezione printfp
for(i=0;i<N;i++)
   hparcheggio[i]=CreateSemaphore(NULL,PST,PST,NULL);
 printf(" Digita un tasto per Uscire \n");
 hfineThreads=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)finito,
(LPVOID) &fine,0, NULL);
//il ciclo do crea un thread per ogni automobile che entra
 i=0;
do
  {if(i==MAXTHREAD)
   {ris=WaitForMultipleObjects(MAXTHREAD,THRCAR,false, INFINITE);
    ii=ris-WAIT_OBJECT_0;}
  else i++;
  ii=i;
  THRCAR[ii]=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)Car,
```

```
(LPVOID) &car ,0, NULL);
  Sleep(GetRandom(200,1000));
  car++;
  }while (fine==0);
WaitForMultipleObjects(MAXTHREAD,THRCAR,true, INFINITE);
for ( i=0;i<N;i++)
   CloseHandle(hparcheggio[i]);
system("PAUSE");
}
////////////////
void Car(LPDWORD ch)
{unsigned long car,*Pcar;
  DWORD numpno,ris;
   long numpst;
  char buffer[80];
  Pcar=(unsigned long *)ch;
  car=*Pcar;
  ris=WaitForMultipleObjects(N,hparcheggio,false, INFINITE);
  numpno=ris-WAIT_OBJECT_0+1;
  sprintf(buffer,"la macchina %d parcheggia al
piano%d\n",car,numpno);
  printfp(buffer);
  srand( (unsigned)time( NULL ) );
```

```
  int a=GetRandom(5000,10000);
  Sleep(a);//specificare un tempo di parcheggio random
  ReleaseSemaphore(hparcheggio[numpno-1],1,&numpst);
  sprintf(buffer,"la macchina %d parcheggiata al piano %d posti
Liberi al pinao %d\n" ,car, numpno , numpst+1);
  printfp(buffer);
 }
void printfp(char *stringa)
{
 WaitForSingleObject(Mut,INFINITE);
 printf(stringa);
 ReleaseMutex(Mut);
}
```