Sincronizzazione 9 SEMAFORI WINDOWS

martedì 25 novembre 2025 15:55

Vediamo ora i semafori in generale in windows:

1. **Creazione**

```
HANDLE CreateSemaphore(LPSECURITY_ATTRIBUTES
                         lpSemaphoreAttributes,
                       LONG lInitialCount,
                       LONG lMaximumCount,
                       LPCTSTR lpNamE)
```

Descrizione
• invoca la creazione di un semaforo

a. **Restituzione**
• handle al semaforo in caso di successo, NULL in caso di fallimento

Parametri
• lpSemaphoreAttributes: puntatore a una struttura SECURITY_ATTRIBUTES
• lInitialCount: valore iniziale del semaforo
• lMaximumCount: massimo valore che il semaforo puo' assumere
• lpName: nome del semaforo

b. **È unico distributore di gettoni**

2. **Operazioni su semaforo**

```
HANDLE OpenSemaphore(LDWORD dwDesiredAccess,
                     BOOL bInheritHandle,
                     LPCTSTR lpName)
```

Accesso al semaforo

```
DWORD WaitForSingleObject(HANDLE hHandle,
                          DWORD dwMilliseconds)
```

a. **Rilascio del semaforo**

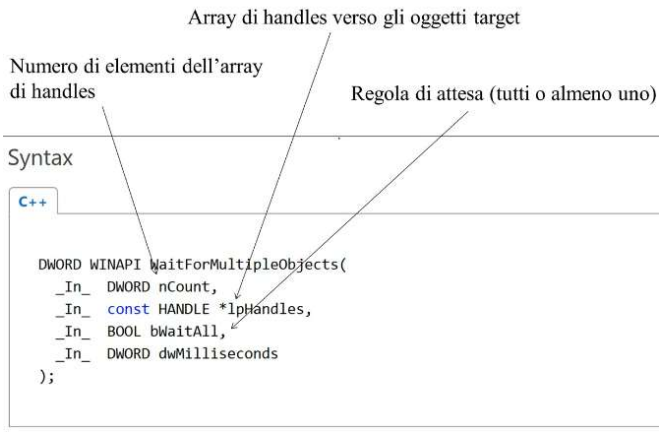
```
BOOL ReleaseSemaphore(HANDLE hSemaphore,
                      LONG lReleaseCount,
                      LPLONG lpPreviousCount)
```

Unita' di rilascio

Puntatore all'area di memoria dove scrivere il vecchio valore del semaforo

3. **Sincronizzazione su oggetti multipli**

a.



Vediamo un esempio:

```c
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>



#define SIZE (100)
#define END (1000000)

#define AUDIT if(0)

long v[SIZE];

long counter = 0;

HANDLE sem_free_slot[SIZE];

HANDLE sem_available_item[SIZE];
```

```
DWORD producer(void){

        long data = 0;
        long my_index = 0;
        long dummy;

        printf("ready to produce\n");
        fflush(stdout);

retry:
        WaitForSingleObject(sem_free_slot[my_index], INFINITE);
        v[my_index] = data;
        ReleaseSemaphore(sem_available_item[my_index],1,&dummy);
        my_index = (my_index + 1) % SIZE;
        data++;
        counter++;
        goto retry;
        return 0;

}
```

```
WORD consumer(void){

       long data = 0;
       long my_index = 0;
       long value;
       long dummy;

       printf("ready to consume\n");
       fflush(stdout);

retry:
       WaitForSingleObject(sem_available_item[my_index], INFINITE);

       value = v[my_index];

       AUDIT
       printf("consumer got value %d\n", value);

       if (value != data){
                       printf("consumer: synch protocol broken at expected value: %d
                       - real is %d!!\n", data, value);
                       ExitProcess(-1);
       };

       if (value == END){
                       printf("ending condition met - last read value is %d\n", value);
                       ExitProcess(0);
       }

       ReleaseSemaphore(sem_free_slot[my_index],1,&dummy);

       my_index = (my_index + 1) % SIZE;

       data++;
       counter--;

goto retry;

}
```

```
int main(int argc, char *argv[]) {

        HANDLE hProducerThread;
        HANDLE hConsumerThread;

        DWORD hid;
        DWORD exit_code;
        int i;
        char buffer[128];//mutex names

        for (i = 0; i < SIZE; i++){
                sprintf(buffer, "slots%d", i);
                sem_free_slot[i] = CreateSemaphore(NULL, 1,1, buffer);
                sprintf(buffer, "items%d", i);
                sem_available_item[i] = CreateSemaphore(NULL, 0,1, buffer);

        }

        hConsumerThread = CreateThread(NULL,
                0,
                (LPTHREAD_START_ROUTINE)consumer,
                NULL,
                NORMAL_PRIORITY_CLASS,
                &hid);

        hProducerThread = CreateThread(NULL,
                0,
                (LPTHREAD_START_ROUTINE)producer,
                NULL,
                NORMAL_PRIORITY_CLASS,
                &hid);


        WaitForSingleObject(hConsumerThread, INFINITE);
        WaitForSingleObject(hProducerThread, INFINITE);

}
```