实验 4 线程控制和信号量的使用

目标：

1、 教材 3.6.4 节讲述了使用 POSIX Pthreads 线程库控制线程的若干调用。本实验学习使用 Windows 中的相关调用创建以及等待线程结束的方法。

2、 学习使用 Windows 相关调用创建信号量，以及对信号量进行 P、V 操作的方法。使用信号量实现临界区的互斥访问。

例程 1：使用 Windows 相关调用创建。

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

DWORD Sum=0;              /*data is shared by the thread(s)*/

/*the thread runs in this separate function*/
DWORD WINAPI Summation(LPVOID Param){
    printf("Sub-thread Id is= %d\n", GetCurrentThreadId());
    DWORD Upper= *(DWORD*)Param;
    for(DWORD i=0;i<=Upper;i++)
        Sum+=i;
    return 0;
}

int main(int argc, char* argv[])
{
    DWORD ThreadId;
    HANDLE ThreadHandle;
    int Param;

    printf("Main-thread Id is= %d\n", GetCurrentThreadId());

    /**/
    if(argc!=2){
        fprintf(stderr,"An integer parameter is required\n");
        return -1;
    }
    Param=atoi(argv[1]);
    if(Param<0){
        fprintf(stderr, "An integer >=0 is required\n");
        return -1;
    }

    //Create the thread
```

```
    ThreadHandle=CreateThread(
                                NULL,           //default security attributes
                                0,              //default stack size
                                Summation,      //thread function
                                &Param,         //parameter to thread function
                                0,              //default creation flags
                                &ThreadId       //returns the thread identifiers
                                );
    if(ThreadHandle!=NULL){
        //now wait for the thread to finish
        //printf("ThreadId is=%d\n", ThreadId);
        WaitForSingleObject(ThreadHandle,INFINITE);

        //close the thread handle
        CloseHandle(ThreadHandle);
        printf("sum=%d\n", Sum);
    }
}
```

运行程序并回答下列问题：

（1） 子线程执行线程函数 Summation( )时，如何将参数 Param 传入该函数？一个线程函数的输入参数可以有任意数目，可以是任意类型，在设计入口线程函数时，是如何解决这一问题的？

（2） 如果由于设计不当，使得子线程不终止（即发生死循环），那么当主线程调用

WaitForSingleObject(ThreadHandle,INFINITE);

时，会发生什么现象？

（3） 如果在线程没有终止之前调用了 CloseHandle(ThreadHandle)，那么会产生什么现象？用程序运行结果来说明。

例程 2：使用 Windows 调用创建信号量，并用 P、V 操作实现临界区互斥。

```
/*
 * Use Win32 Semaphore to solve mutual exclusion
 * Sample code
 * Watch the results of using an Semaphore object.
 */
#include <windows.h>
#include <process.h>
#include <stdio.h>

#define THRDSNUM 6      /*The number of threads*/

HANDLE hSem;                /*Handle of semaphore*/
HANDLE hThrds[THRDSNUM];    /*Set of thread handles*/
DWORD WINAPI MyThread(LPVOID);      /*Prototype of thread function*/
```

```
DWORD WINAPI MyThread(LPVOID lpParameter)
{
    int* pNo = (int *)lpParameter;
    printf("    Thread #%d wait for critical region...\n", *pNo);

    /*Wait for semaphore available infinitely*/
    WaitForSingleObject(hSem,INFINITE);
    printf("        Thread #%d Enter critical region...\n", *pNo);

    printf("        Thread #%d will sleep 1 seconds\n", *pNo);
    Sleep(1000);
    printf("        Thread #%d Leave critical region...\n", *pNo);

    /*Release semaphore*/
    ReleaseSemaphore(hSem,1,NULL);    //releasing locks
    return ((DWORD)lpParameter);
}

int main()
{
    int ThrdNo[THRDSNUM];
    DWORD dw;
    DWORD rc;
    hSem = CreateSemaphore(NULL,    //安全属性,NULL 表示使用默认值
        1,                          //semaphore 初值
        6,                          //semaphore 的最大值
        "MySemaphore");             //semaphore 的名称

    /*Create a number of threads*/
    for(int i = 0; i<THRDSNUM; i++)
    {
        ThrdNo[i] = i+1;
        printf("Main thread will create the %d thread\n",i+1);
        hThrds[i]=CreateThread(NULL,
            0,
            MyThread,               /*Thread function*/
            &ThrdNo[i],             /*Parameter of thread function*/
            NULL,
            &dw);
    }

    /*Main thread wait for all of the threads terminate*/
    WaitForMultipleObjects(THRDSNUM,hThrds,TRUE,INFINITE);
```

```
    /*Close threads*/
    for(int i=0;i<THRDSNUM;i++)
    {
        CloseHandle(hThrds[i]);
        printf("Thread #%d terminated.\n", i+1);
    }
    //Sleep(100000);
    //asm("int3;");
    /*Close semaphore*/
    CloseHandle(hSem);

    return 0;
}
```

运行程序并回答下列问题：

（1） 观察程序运行结果，说明是否能够实现临界区的互斥访问？每次运行的结果是否一致？为什么？

（2） 例程中同一进程中的多个线程共享信号量句柄 hSem，那么用信号量能否实现进程的互斥？在多进程环境下，一个进程中的信号量句柄 hSem 不能被其它进程所共享，那么其它进程如何引用一个进程中创建的信号量？

（3） 一个进程中的子线程和信号量都属于该进程的资源。请设法让程序在 6 个线程和信号量没有关闭之前，暂停下来，然后用 Process Explorer 工具观察进程中所包含的这些资源，并获取线程的 ID 号。

（4） 主线程使用什么调用等待所有子线程的结束？