|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 数学与信息科学学院 | | | | | | |  |
|  |  | | | | | | |  |
|  |  | | | | | | |  |
|  | **实验报告** | | | | | | |  |
|  |  | | | | | | |  |
|  |  | | | | | | |  |
|  | **课程名称：** | | **计算机操作系统** | | | | |  |
|  | **姓 名：** | | **任希恒** | | | | |  |
|  | **学 号：** | | **541910010217** | | | | |  |
|  | **专业班级：** | | **信息与计算科学专业19-02班** | | | | |  |
|  | **指导教师：** | | **时海亮** | | | | |  |
|  |  | |  | | | | |  |
|  |  | |  | | | | |  |
|  |  | **2021-2022** | | **学年第** | **2** | **学期** |  |  |

**实验2 线程的同步与互斥**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验日期：** | **2022年04月15日** | **实验类型：** | **设计型** | **实验成绩：** |  |

**一、实验目的**

通过本实验熟悉Windows系统提供的线程创建与撤销，熟悉Windows系统提供的线程互斥API，能够使用线程互斥API解决一些实际问题。

**二、实验内容**

1.主线程与子线程的同步；

2.对线程数限定的线程间同步；

3.基于互斥对象的线程互斥；

4.基于事件对象的线程互斥；

5.基于临界区对象的线程互斥；

**三、实验代码及结果**

## 1.第四题：基于互斥对象实现对火车票销售模拟系统；

第四题实验代码如下所示：

#include <windows.h>

#include <iostream.h>

int tickets=100;

HANDLE hEvent;

DWORD WINAPI Thread1Proc(LPVOID lpParameter);

DWORD WINAPI Thread2Proc(LPVOID lpParameter);

void main()

{

HANDLE hThread1;

HANDLE hThread2;

hThread1 = CreateThread(NULL,0,Thread1Proc,NULL,0,NULL);

hThread2 = CreateThread(NULL,0,Thread1Proc,NULL,0,NULL);

CloseHandle(hThread1);

CloseHandle(hThread2);

hEvent = CreateEvent (NULL,FALSE,FALSE,"tickets");

if (hEvent)

{

if (ERROR\_ALREADY\_EXISTS==GetLastError())

{

cout<<"only one instance can run !"<<endl;

return;

}

}

SetEvent(hEvent);

Sleep(5000);

CloseHandle(hEvent);

}

DWORD WINAPI Thread1Proc(LPVOID lpParameter)

{

while (TRUE)

{

WaitForSingleObject(hEvent,INFINITE);

if(tickets>0)

{

Sleep(1);

cout<<"Thread 1 sell ticket:"<<tickets--<< '\t';

}

else

break;

SetEvent(hEvent);

}

return 0;

}

DWORD WINAPI Thread2Proc(LPVOID lpParameter)

{

while (TRUE)

{

WaitForSingleObject(hEvent,INFINITE);

if(tickets>0)

{

Sleep(1);

cout<<"Thread 2 sell ticket:"<<tickets--<<endl;

}

else

break;

SetEvent(hEvent);

}

return 0;

}

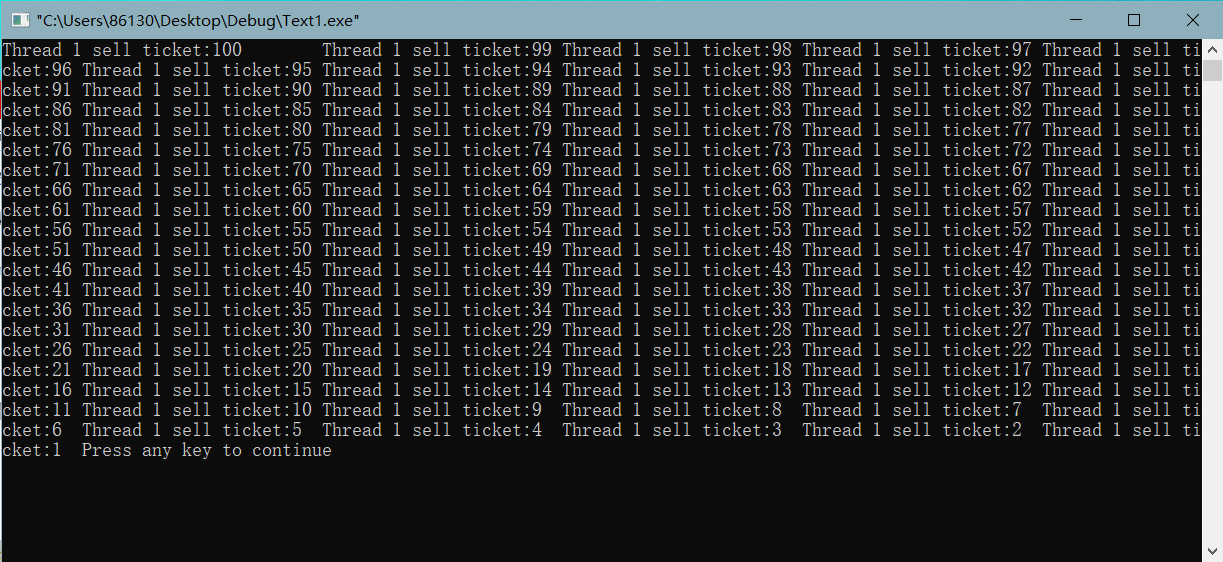
第四题实验结果如图所示：

图1

## 2.第六题：基于临界区对象实现对火车票销售模拟系统；

第六题实验代码如下所示：

#include <windows.h>

#include <iostream.h>

int tickets=100;

CRITICAL\_SECTION cs;

DWORD WINAPI Proc1Fun(LPVOID lpParameter);

DWORD WINAPI Proc2Fun(LPVOID lpParameter);

int main()

{

HANDLE hthread1;

HANDLE hthread2;

hthread1=CreateThread(NULL,0,Proc1Fun,NULL,0,NULL);

hthread2=CreateThread(NULL,0,Proc2Fun,NULL,0,NULL);

InitializeCriticalSection(&cs);

CloseHandle(hthread1);

CloseHandle(hthread2);

Sleep(4000);

DeleteCriticalSection(&cs);

return 0;

}

DWORD WINAPI Proc1Fun(LPVOID lpParameter)

{

while(TRUE)

{

EnterCriticalSection(&cs);

if(tickets>0)

{

Sleep(1);

cout<<"Thread1 sells ticket"<<tickets--<<endl;

}

else

break;

LeaveCriticalSection(&cs);

}

return 0;

}

DWORD WINAPI Proc2Fun(LPVOID lpParameter)

{

while(TRUE)

{

EnterCriticalSection(&cs);

if(tickets>0)

{

Sleep(1);

cout<<"Thread2 sells ticket"<<tickets--<<endl;

}

else

break;

LeaveCriticalSection(&cs);

}

return 0;

}

第六题实验结果如下所示：

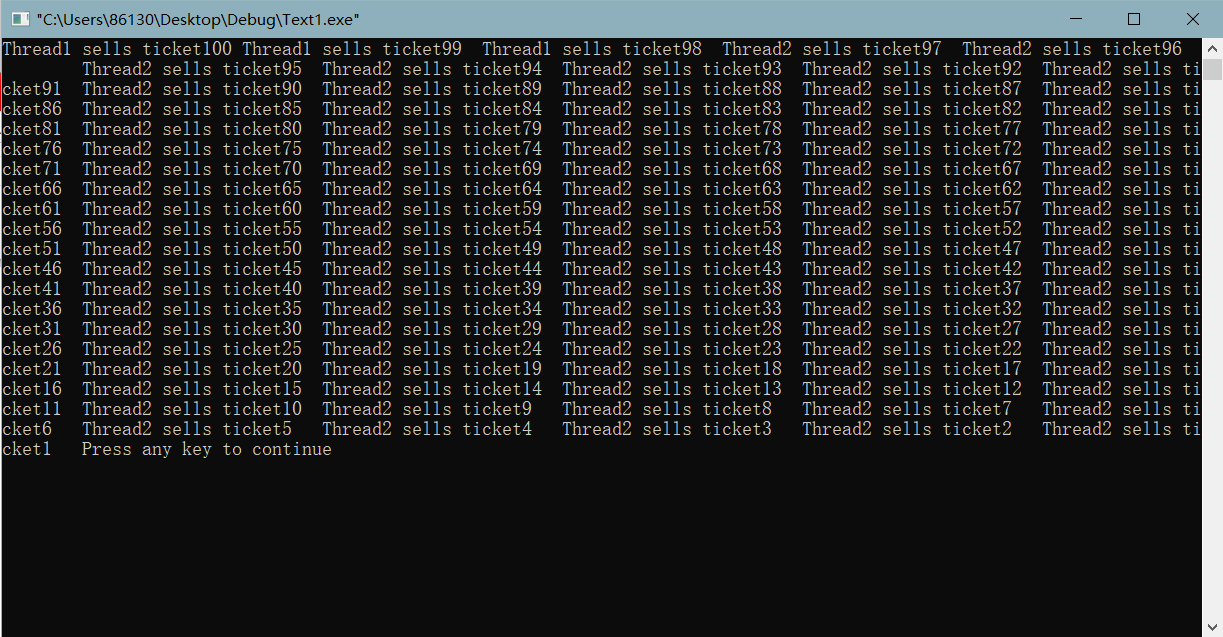


图2

**四、实验总结**

1、 临界区（CCriticalSection）

当多个线程访问一个独占性共享资源时，可以使用临界区对象。拥有临界区的线程可以访问被保护起来的资源或代码段，其他线程若想访问，则被挂起，直到拥有临界区的线程放弃临界区为止。具体应用方式：

定义临界区对象CcriticalSection g\_CriticalSection;

在访问共享资源（代码或变量）之前，先获得临界区对g\_CriticalSection.Lock

访问共享资源后，则放弃临界区对象，g\_CriticalSection.Unlock（）；

2、 事件（CEvent）

事件机制，则允许一个线程在处理完一个任务后，主动唤醒另外一个线程执行任务。比如在某些网络应用程序中，一个线程如A负责侦听通信端口，另外一个线程B负责更新用户数据，利用事件机制，则线程A可以通知线程B何时更新用户数据。每个Cevent对象可以有两种状态：有信号状态和无信号状态。Cevent类对象有两种类型：人工事件和自动事件。

自动事件对象，在被至少一个线程释放后自动返回到无信号状态；

人工事件对象，获得信号后，释放可利用线程，但直到调用成员函数ReSet()才将其设置为无信号状态。在创建Cevent对象时，默认创建的是自动事件。

CEvent(BOOL bInitiallyOwn=FALSE,

BOOL bManualReset=FALSE,

LPCTSTR lpszName=NULL,

LPSECURITY\_ATTRIBUTES lpsaAttribute=NULL);

bInitiallyOwn:指定事件对象初始化状态，TRUE为有信号，FALSE为无信号；

bManualReset：指定要创建的事件是属于人工事件还是自动事件。TRUE为人工事件，FALSE为自动事件；后两个参数一般设为NULL，在此不作过多说明。

2、BOOL CEvent：：SetEvent();

将Cevent类对象的状态设置为有信号状态。如果事件是人工事件，则Cevent类对象保持为有信号状态，直到调用成员函数ResetEvent()将其重新设为无信号状态时为止。如果为自动事件，则在SetEvent（）后将事件设置为有信号状态，由系统自动重置为无信号状态。

3、BOOL CEvent：：ResetEvent();

将事件的状态设置为无信号状态，并保持该状态直至SetEvent（）被调用为止。由于自动事件是由系统自动重置，故自动事件不需要调用该函数。一般通过调用WaitForSingleObject（）函数来监视事件状态。

4、 互斥量（CMutex）

互斥对象和临界区对象非常相似，只是其允许在进程间使用，而临界区只限制与同一进程的各个线程之间使用，但是更节省资源，更有效率。

5、 信号量（CSemphore）

当需要一个计数器来限制可以使用某共享资源的线程数目时，可以使用“信号量”对象。CSemaphore类对象保存了对当前访问某一个指定资源的线程的计数值，该计数值是当前还可以使用该资源的线程数目。如果这个计数达到了零，则所有对这个CSemaphore类对象所控制的资源的访问尝试都被放入到一个队列中等待，直到超时或计数值不为零为止。

CSemaphore 类的构造函数原型及参数说明如下：

CSemaphore(

LONG lInitialCount = 1,

LONG lMaxCount = 1,

LPCTSTR pstrName = NULL,

LPSECURITY\_ATTRIBUTES lpsaAttributes = NULL

);

lInitialCount:信号量对象的初始计数值，即可访问线程数目的初始值；

lMaxCount：信号量对象计数值的最大值，该参数决定了同一时刻可访问由信号量保护的资源的线程最大数目；后两个参数在同一进程中使用一般为NULL，不作过多讨论；

一般是将当前可用资源计数设置为最大资源计数，每增加一个线程对共享资源的访问，当前可用资源计数就减1，只要当前可用资源计数大于0，就可以发出信号量信号。如果为0，则放入一个队列中等待。线程在处理完共享资源后，应在离开的同时通过ReleaseSemaphore（）函数将当前可用资源数加1。

BOOL ReleaseSemaphore( HANDLE hSemaphore, // hSemaphore:信号量句柄

　　　　　 LONG lReleaseCount, // lReleaseCount：信号量计数值

　　　　　　　　　　　　　LPLONG lpPreviousCount // 参数一般为NULL);