实验6：并发程序设计

**【实验目的】**

1. 掌握Windows进程的创建和销毁等控制方法；
2. 掌握Windows线程的创建方法；
3. 掌握利用windows信号机制实现信号量，控制线程同步的基本方法

**【实验要求及内容】**

1. 编写代码，在程序中创建和销毁一个notepad进程；
2. 编写代码，在程序中创建两个线程，分别给与不同参数，线程不断打印这个参数以显示自己的存在。
3. 利用Windows的信号量机制实现简单的P，V操作同步控制。

实现如下功能：

线程1：主要功能是循环打印主线程赋给的参数

线程2：每打印20个数后，需要等待线程3（Thread3）打印30个数，释放一次信号，才能继续

线程3：在每打印30个数以后，释放信号量

1. 用两个线程模拟实现生产者、消费者模型。（选做）

**【相关函数说明】**

1）创建进程

Windows系统中创建线程的函数是CreateProcess。请查阅相关资料了解这个函数。在本实验中，我们采用最简单的方法使用这个函数，以了解Windows进程创建和控制的基本方法。

BOOL CreateProcess(

LPCTSTR lpApplicationName,

LPTSTR lpCommandLine,

LPSECURITY\_ATTRIBUTES lpProcessAttributes,

LPSECURITY\_ATTRIBUTES lpThreadAttributes,

BOOL bInheritHandles,

DWORD dwCreationFlags,

LPVOID lpEnvironment,

LPCTSTR lpCurrentDirectory,

LPSTARTUPINFO lpStartupInfo,

LPPROCESS\_INFORMATION lpProcessInformation );

重要参数：

* lpApplicationName：

这是一个字符串，是被启动进程的程序的路径，比如：

“C:\\Program Files\\Microsoft Office\\root\\Office16\\WINWORD.exe”

注意在Windows下，路径间隔需要使用双斜杠

* lpCommandLine

这也是一个字符串，是被启动进程的程序命令行参数，就是在C语言中学过的main函数的参数表。

本实验建议使用notepad作为被创建的进程，这个程序是Windows系统的常用应用，不需要指明路径就能运行，所以我们让lpApplicationName为NULL，lpCommandLine 为”notepad”，就可以启动这个进程了。

* lpStartupInfo

输出参数，是个结构体变量（STARTUPINFO），需要先声明变量，然后传入指针，在传入前，最好把变量内容初始化为0，然后将其cb成员填写为结构体的大小。如下：

STARTUPINFO si;

ZeroMemory( &si, sizeof(si) );

si.cb = sizeof(si);

感兴趣的的同学可以在调试状态下，执行了创建函数以后，看看这个结构体里都有些什么内容。

* lpProcessInformation

输出参数，是个结构体（PROCESS\_INFORMATION），需要先声明变量，然后传入指针，传入前，将结构体内容初始化为0。如下：

PROCESS\_INFORMATION pi;

ZeroMemory( &pi, sizeof(pi) );

这个变量中的hProcess成员是进程的句柄（相当于指针），在进程的各控制函数中是关键核心变量，请注意。

这个变量中的hThread成员是进程的主线程句柄，在线程的控制函数中是关键核心变量，请注意。

除了上述四个参数外，CreateProcess所需的其他参数均可以填写为0或NULL。

2）创建线程

HANDLE CreateThread(

LPSECURITY\_ATTRIBUTES lpThreadAttributes,//SD

SIZE\_T dwStackSize,//initialstacksize

LPTHREAD\_START\_ROUTINE lpStartAddress,//threadfunction

LPVOID lpParameter,//threadargument

DWORD dwCreationFlags,//creationoption

LPDWORD lpThreadId//threadidentifier )

创建线程也需要诸多参数，比较复杂，本实验实现最简单的创建方法。

与进程创建执行的是已经编译好的程序不同，线程的代码是本程序的部分，比如某个子函数。但是线程与在程序中调用该子函数是有区别的。本实验通过创建两个线程，观察和记录他们的执行情况，从而对线程的动态性有足够认识。

最简单的线程创建需要使用以下三个参数：

* lpStartAddress

这是一个函数指针类型，指向线程程序体，类似于进程的main函数，是需要线程执行的内容。一般填写子函数的名字就可以了。

* lpParameter

输入参数，当线程开始执行时，如果需要向线程传递初始化参数，可以使用这个指针。

如在本实验中，因为子线程都是相同的函数，为了能区分他们，主线程通过dwThrdParam参数向两个子线程传递了不同的值，供他们打印，实验者就可以观察识别出是哪个线程在执行了。

* lpThreadId

输出参数，让主线程得到子线程的标识符，这样当主线程需要强制关闭或控制子线程时，就可以利用这个标识符。当然一般情况下，主线程都会放任让子线程自行执行到结束。

这是指针类型的输出参数，需要先申明一个线程标识符变量（类型：DWORD），然后取其地址输入。

* 返回值

CreateThread函数的返回值是被创建线程的句柄，其类型为HANDLE；在大部分的线程控制函数中使用。线程句柄和标示符都可以用来控制线程，不同的只是应用场景。根据需要选用。

3）其他控制函数

* + **TerminateProcess()的基本使用方法：(销毁进程)**

参数：

hProcess：需要销毁的进程句柄

UINT uExitCode：进程终止码

返回值：

如果函数执行成功，返回非零值；如果函数执行失败，返回零。

代码示例：

TerminateProcess(pi.hProcess, 0);

* **SuspendThread()的基本使用方法：（挂起一个进程的主线程）**

参数：

hThread：需要挂起的线程句柄

返回值：

如果函数运行成功，返回线程已经被挂起的次数；如果函数运行失败，返回-1；

代码示例：

if (SuspendThread(pi.hThread) >= 0 )

* **ResumeThread()的基本使用方法：（激活一个进程的主线程）**

参数：

hThread：需要激活的线程句柄

返回值：

如果函数执行成功，返回非零值；如果函数执行失败，返回零。

代码示例：

ResumeThread(pi.hThread)

4）线程同步控制

Windows系统实现进/线程控制的机制有4种：临界区、互斥量、事件、信号量。简单说明如下：

//临界区：

CriticalSection:

CRITICAL\_SECTION g\_csA;

InitializeCriticalSection(&g\_csA);

EnterCriticalSection(&g\_csA);

LeaveCriticalSection(&g\_csA);

DeleteCriticalSection(&g\_csA);

//互斥对象:

HANDLE hMutex;

hMutex=CreateMutex(NULL,TRUE,LPCTSTR("tickets"));

WaitForSingleObject(hMutex,INFINITE);

ReleaseMutex(hMutex);

CloseHandle(hMutex);

//事件对象

HANDLE g\_hEvent;

g\_hEvent=CreateEvent(NULL,FALSE,FALSE,LPCTSTR("tickets"));

OpenEvent(),打开一个事件——用的少

ResetEvent(g\_hEvent);

SetEvent(g\_hEvent);回置事件

WaitForSingleObject(g\_hEvent,IFINITE)

WaitForMultipleObjects()

//信号量

CreateSemaphore() 创建一个信号量

OpenSemaphore() 打开一个信号量

ReleaseSemaphore() 释放信号量

WaitForSingleObject() 等待信号量

//信号量示例：

HANDLE global\_Semephore;

global\_Semephore= CreateSemaphore(NULL, 1, 1, NULL);

WaitForSingleObject(global\_Semephore, INFINITE);

ReleaseSemaphore(global\_Semephore, 1, NULL);

**【部分实验源码】**

#include <iostream>

#include <process.h>

#include <windows.h>

using namespace std;

DWORD WINAPI Thread1( LPVOID lpParam );

DWORD WINAPI Thread2( LPVOID lpParam );

DWORD WINAPI Thread3( LPVOID lpParam );

bool isStop = false;

int main() {

STARTUPINFO si;

PROCESS\_INFORMATION pi;

TCHAR szCommandLine[] = TEXT("notepad");

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi));

int x;

while (true) {

cout << "请输入要选择的操作：\n0:退出\n1:创建进程\n2:挂起进程\n3:激活进程\n4:销毁进程\n5:启动两个一样的线程\n6:启动两个不同线程\n7:停止线程\n";

cin >> x;

switch (x) {

case 0:

return 0;

case 1:

if (!CreateProcess( )){ //创建进程，参数2使用szCommandLine，启动notepad软件

fprintf(stderr, "Createprocess Failed \n");

return 0;

}

break;

case 2:

if (SuspendThread(pi.hThread) >= 0 )

cout << "挂起进程成功" << endl;

else

cout << "挂起失败" << endl;

break;

case 3:

if ( ) //调用激活线程的函数

cout << "激活进程成功" << endl;

else

cout << "激活失败" << endl;

break;

case 4:

if ( ) //调用终止线程的函数

cout << "销毁进程成功" << endl;

else

cout << "销毁失败" << endl;

break;

case 5:

HANDLE hThread1, hThread2;

DWORD dwThreadId1, dwThreadId2;

//创建两个相同的子线程，启动并观察,一个传入整数11，一个传入整数22

hThread1 = CreateThread(

NULL, // no security attributes

0, // use default stack size

, //需要填写Thread1

(void \*)11, // argument to thread function

0, // use default creation flags

&dwThreadId1 ); // returns the thread identifier

hThread2 = CreateThread(

NULL, // no security attributes

0, // use default stack size

, //需要填写Thread1

(void \*)22, // argument to thread function

0, // use default creation flags

&dwThreadId2 ); // returns the thread identifier

break;

case 6:

//创建两个不同的子线程，启动并观察同步控制过程

HANDLE hThread3,hThread4;

DWORD dwThreadId3,dwThreadId4;

//创建信号量

HANDLE hSem;

hSem = ; //创建一个信号量，初值为0，最大为1

//创建两个不同的子线程，启动并观察,信号量句柄hSem作为线程参数传递给线程

hThread3 = CreateThread(

NULL, // no security attributes

0, // use default stack size

, //需要填写Thread2

(LPVOID)hSem, // argument to thread function

0, // use default creation flags

&dwThreadId3 ); // returns the thread identifier

hThread4 = CreateThread(

NULL, // no security attributes

0, // use default stack size

, // 需要填写Thread3

(LPVOID)hSem, // argument to thread function

0, // use default creation flags

&dwThreadId4 ); // returns the thread identifier

break;

case 7:

isStop = true;

break;

default:

cout << "选项不正确" << endl;

}

return 0;

}

DWORD WINAPI Thread1( LPVOID lpParam )

{

int count;

count = (long long)lpParam;

while(!isStop){

printf("%d ",count);

//Sleep(10);

}

return 0;

}

// 补充Thread2，每打印20个，申请信号量

// 补充Thread3，每打印30个，释放信号量