操作系统实验：线程的同步

原创

司马道 于 2020-04-23 09:27:21 发布 2133 收藏 19

文章标签： 操作系统 多线程 多进程

本文链接：https://blog.csdn.net/qq_44919455/article/details/105699776

实验二：线程的同步

一、实验目的

(1)了解线程的历史。
(2)理解线程同步的工作原理。
(3)掌握等待一个对象函数WaitForSingleObject()。
(4)掌握等待多个对象函数WaitForMultipleObjects()。

二、实验准备

(1).实验在windows XP，VC++6.0环境下进行。在这一步，安装了Windows XP虚拟机，学会了创建一个控制台工程文件。

(2).百度句柄的含义：句柄（handle），有多种意义，第一种解释：句柄是一种特殊的智能指针 。当一个应用程序要引用其他系统（如数据库、操作系统）所管理的内存块或对象时，就要使用句柄。句柄是Windows用来标志应用程序中建立的或是适用的对象的唯一整数，Windows大量使用了句柄来标识对象。

(3).信号量：信号量的使用主要是用来保护共享资源，使得资源在一个时刻只有一个进程（线程）所拥有。为了防止出现因多个程序同时访问一个共享资源而引发的一系列问题，我们需要一种方法，它可以通过生成并使用令牌来授权，在任一时刻只能有一个执行线程访问代码的临界区域。

(4).参照实验指导学习函数原型--WaitForSingleObject()

```
DWORD WaitForSingleObject（
HANDLE hHandle,              //对象句柄
DWORD dwMilliseconds         //等待时间
）；
```

参数说明：
<1> hHandle：等待对象的对象句柄。该对象句柄必须为SYNCHRONIZE（['sɪŋkrənaɪz]，同步）访问。
<2> dwMilliseconds：等待时间，单位为ms。若改值为0，函数在测试对象的状态后立即返回，若为INFINITE（无限的），函数一直等待下去，直到收到一个信号将其唤醒，如表2-1所示。　　返回值： 如果返回成功，其返回值说明是何种事件导致函数返回。

(5).参照实验指导学习函数原型--WaitForMultipleObjects()

```
DWORD  WaitForMultipleObject(
DWORD  nCount,               //句柄数组中的句柄数
XONST  HANDLE *lpHandles,      //指向对象句柄数组的指针
BOOL   fWaitAll,               //等待类型
DWORD  dwMilliseconds         //等待时间
);
```

参数说明：
<1> nCount:由指针*lpHandles指定的句柄数组中的句柄数，最大数是MAXIMUM_WAIT_OBJECTS。
<2> *lpHandles：指向对象句柄数组的指针。
<3> fWAitAll：等待类型。若存为true，当由lpHandles数组指定的所有对象被唤醒时函数返回；若为FALSE，当由lpHandles数组制定的某一个对象被唤醒时函数返回，且有返回值说明事由哪个对象引起的函数返回。
<4> dwMilliseconds ：等待时间。单位为ms。若该值为0，函数测试对象的状态后立即返回；若为INFINITE,函数一直等待下去，直到收到一

个信号将其唤醒。
返回值： 如果成功返回，其返回值说明是何种事件导致函数返回。

(6).参照实验指导学习函数原型--CreateSemaphore()

```
HANDLE  CreateSemaphore（
LPSECURITY_ATTRIBUTES  lpSemaphoreAttributes，//安全属性
LONG  lInitialCount，                          //信号量对象初始值
LONG  liMaximumCount，                       //信号量最大值
LPCTSTR  lpName       //信号量名
）；
```

参数说明：
<1> lpSemaphoreAttributes：指定安全属性，为null是，信号量得到一个默认的安全描述符。
<2> lInitialCount：指定信号量对象的初始值。该值必须大于等于0，小于等于lMaximumCount 。当其值大于0是，信号量被唤醒。当该函数释放了一个等待该信号量的线程时，lInitialCount值减1，当调用函数ReleaseSemaphore()时，按其指定的数量加一个值。
<3> lMaximumCount：指出该信号量的最大值，该值必须大于0.
<4> lpName：给出该信号量的名字。
返回值： 信号量创建成功，将返回该型号量的句柄。如果给出的信号量名是系统已经存在的信号量，将返回这个已经存在的信号量的句柄。如果失败，系统返回null，还可以调用函数GEtLastError（）查询失败的原因。

(7).参照实验指导学习函数原型--OpenSemaphore()

```
HANDLE  OpenSemaphore（
DWORD  dwDesidedAccess，                //访问标志
BOOL  bInheritHandle，                    //继承标志
LPCTSTR  lpNme                            //信号量名
）；
```

参数说明：
<1> dwDesiredAccess：指出打开后要对信号量进行何种访问。
<2> bInheritHandle：指出返回的的信号量句柄是否可以继承。
<3> lpName：给出信号量的名字
返回值： 信号量打开成功，将返回信号量的句柄；如果失败，系统返回null，可以调用函数GetLastError（）查询失败的原因。

(8).参照实验指导学习函数原型--ReleaseSemaphore()

```
BOOL ReleaseSemaphore（
HANDLE  hSemaphore，           //信号量对象句柄
LONG lReleaseCount，             //信号量要增加数值
LPLONG lpPreiousCount            //信号量要增加数值地址
）；
```

参数说明：
<1> hSemaphore：创建或打开信号量时给出的信号量对象句柄。Windows NT中建议使用SEMAPHORE_MODIFY_STARTE访问属性打开该信号量。
<2> lReleaseCount：信号量要增加数值。该值必须大于0。如果增加该值后大于信号创建时给出的lMaximumCount值，则增加操作失效，函数返回FALSE。
<3> lpPreiousCount ：接收信号量的一个32位的一个变量。若不需要接受该值，可以指定为null。
返回值： 如果成功，将返回一个非0值；如果失败，系统返回一个0，可以调用一个GetLastError

(9).线程同步的原理：当一个线程访问临界区时，判断是否有其他线程进入临界区，如果没有它才可以进入临界区，如果已经有线程进入临界区它会被同步挂起，直到进入的线程离开。

三、实验内容

（一）实验内容

实验1：等待一个对象 掌握等待一个对象函数。模仿教学视频中的程序，创建一个线程，在该线程完成后，主线程结束。完成制作，开始使用香锅。

实验2：等待多个对象 掌握等待多个对象函数。模仿教学视频中的程序，创建三个线程，在三个线程完成后，主线程结束。在米饭，什锦菇，麻辣香锅结束后，开始食用。

（二）主要代码：

实验一主要代码：

```
#include "stdafx.h"
#include "02.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/
// The one and only application object
CWinApp theApp;
using namespace std;

static HANDLE h1;                              //线程句柄
static HANDLE hHandle1=NULL;                   //信号量句柄

void func()
{
 printf("开始制作!预计时间5秒。\n");
 Sleep(5000);
 printf("制作完成!\n");

 BOOL rc;
 DWORD err;

 rc=ReleaseSemaphore(hHandle1,1,NULL);
 err=GetLastError();
 printf("ReleaseSemaphore err = %d\n",err);
 if (rc==0)  printf("Semaphore realese Fail!\n");
 else printf("Semaphore realese Success! rc=%d\n",rc);
};

int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
 int nRetCode = 0;
 DWORD dRes,err;

 hHandle1=CreateSemaphore(NULL,0,1,"SemaphoreName1");//创建一个信号量
 if (hHandle1==NULL) printf("Semaphore Create!\n");
 else printf("Semaphore Create Success!\n");

 hHandle1=OpenSemaphore(SYNCHRONIZE|SEMAPHORE_MODIFY_STATE,
  NULL,
  "SemaphoreName1");                                //打开信号量
 if (hHandle1==NULL)  printf("Semaphore Open Fail!\n");
 else printf("Semaphore Open Success!\n");

 HANDLE h1=NULL;
 DWORD dwThreadID1=NULL;

 h1=CreateThread((LPSECURITY_ATTRIBUTES)NULL,
  0,
  (LPTHREAD_START_ROUTINE)func,
  (LPVOID)NULL,
  0,&dwThreadID1);                                  //创建子线程
```

```
 dRes=WaitForSingleObject(hHandle1,INFINITE);        //主线程等待子线程结束

 err=GetLastError();
 if (err==0)   printf("制作完成请开动吧!\n");
 else  printf("WaitforSingleob=%d\n",err);

 return nRetCode;
}
```

实验二主要代码：

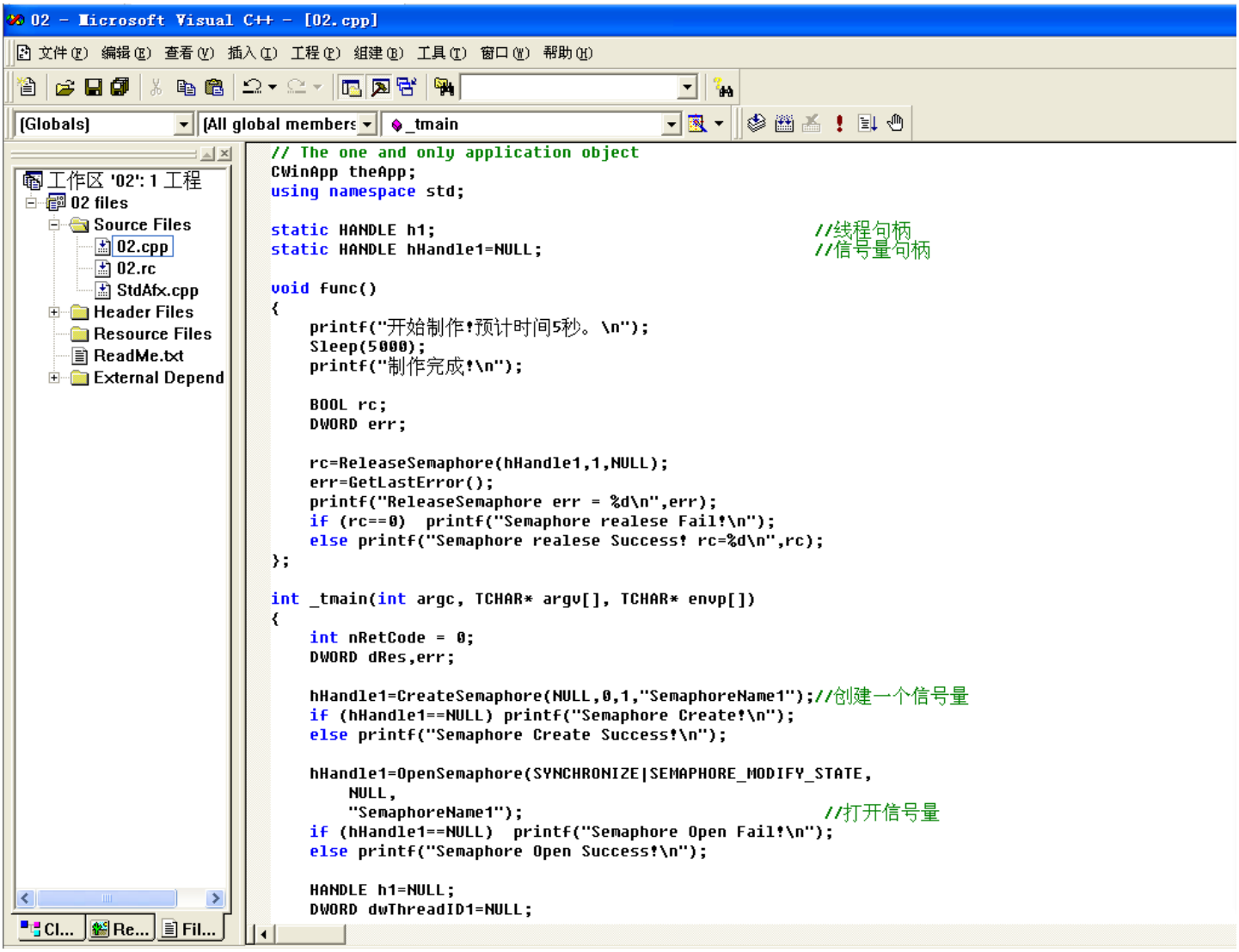
```
#include "stdafx.h"
#include "022.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/
// The one and only application object
CWinApp theApp;
using namespace std;

void func(int meal_code)
{
 if(meal_code==0){
  Sleep(5000);
  printf("麻辣香锅制作完成!\n");
 }
 else if(meal_code==1){
  Sleep(3000);
  printf("深津沽制作完成!\n");
 }
 else if(meal_code==2){
  Sleep(6000);
  printf("米饭制作完成!\n");
 }
};

int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
 int nRetCode = 0;
 DWORD dRes,err;

 HANDLE handle1=NULL;
 HANDLE handle2=NULL;
 HANDLE handle3=NULL;

 DWORD ThreadID1=NULL;
 DWORD ThreadID2=NULL;
 DWORD ThreadID3=NULL;

 int a=0;
 int b=1;
 int c=2;

 handle1=CreateThread((LPSECURITY_ATTRIBUTES)NULL,
  0,
```

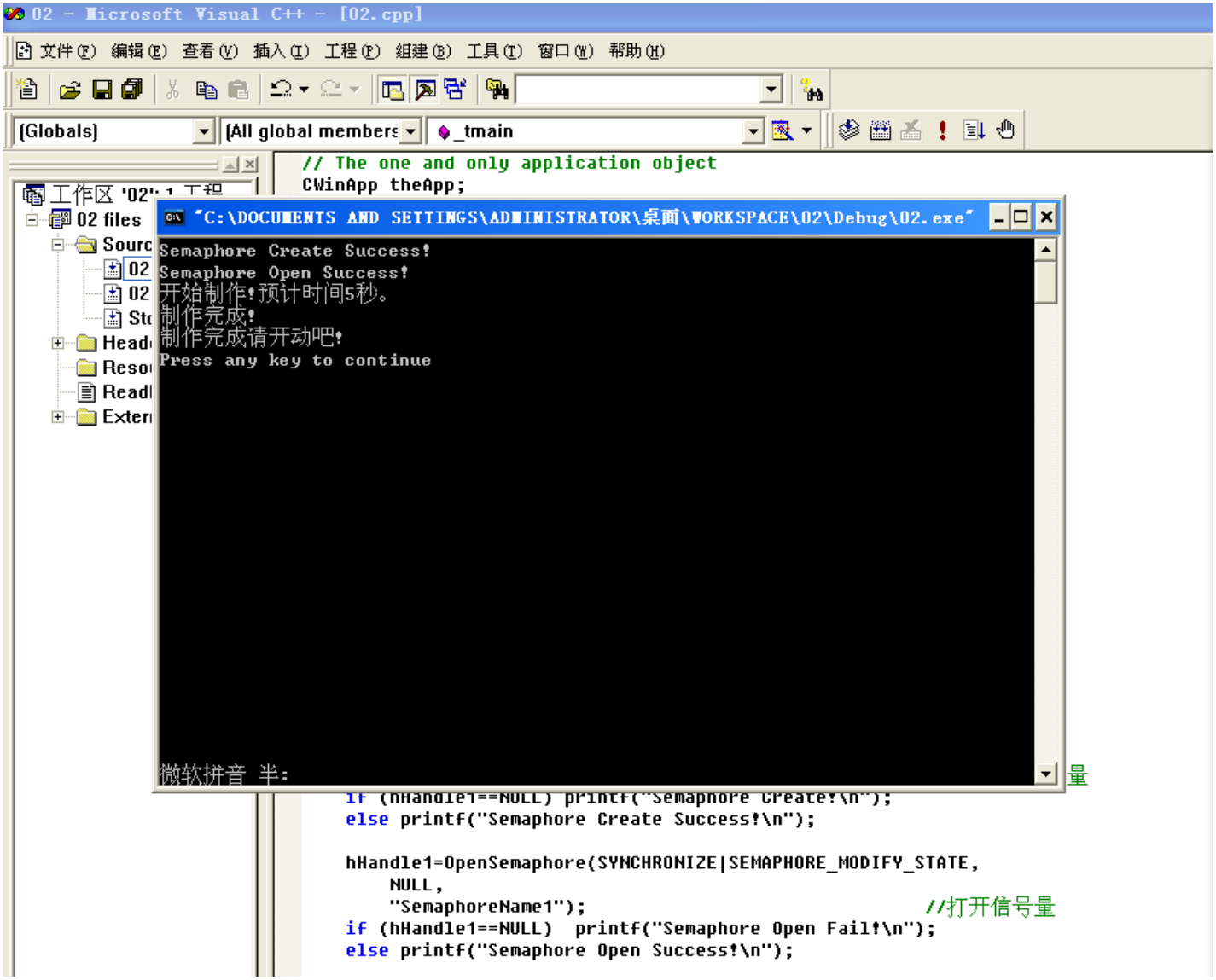
```
 (LPTHREAD_START_ROUTINE)func,
 (LPVOID)a,
 0,&ThreadID1);                                  //创建子线程
handle2=CreateThread((LPSECURITY_ATTRIBUTES)NULL,
 0,
 (LPTHREAD_START_ROUTINE)func,
 (LPVOID)b,
 0,&ThreadID2);                                  //创建子线程
handle3=CreateThread((LPSECURITY_ATTRIBUTES)NULL,
 0,
 (LPTHREAD_START_ROUTINE)func,
 (LPVOID)c,
 0,&ThreadID3);                                  //创建子线程

HANDLE hHandles[3];
hHandles[0]=handle1;
hHandles[1]=handle2;
hHandles[2]=handle3;

dRes=WaitForMultipleObjects(3,hHandles,1,INFINITE);//主线程等待子线程结束

err=GetLastError();
if (err==0)   printf("制作完成请开动吧!\n");
else  printf("WaitforSingleob=%d\n",err);

return nRetCode;
}
```

四、实验结果与总结

实验一总结：

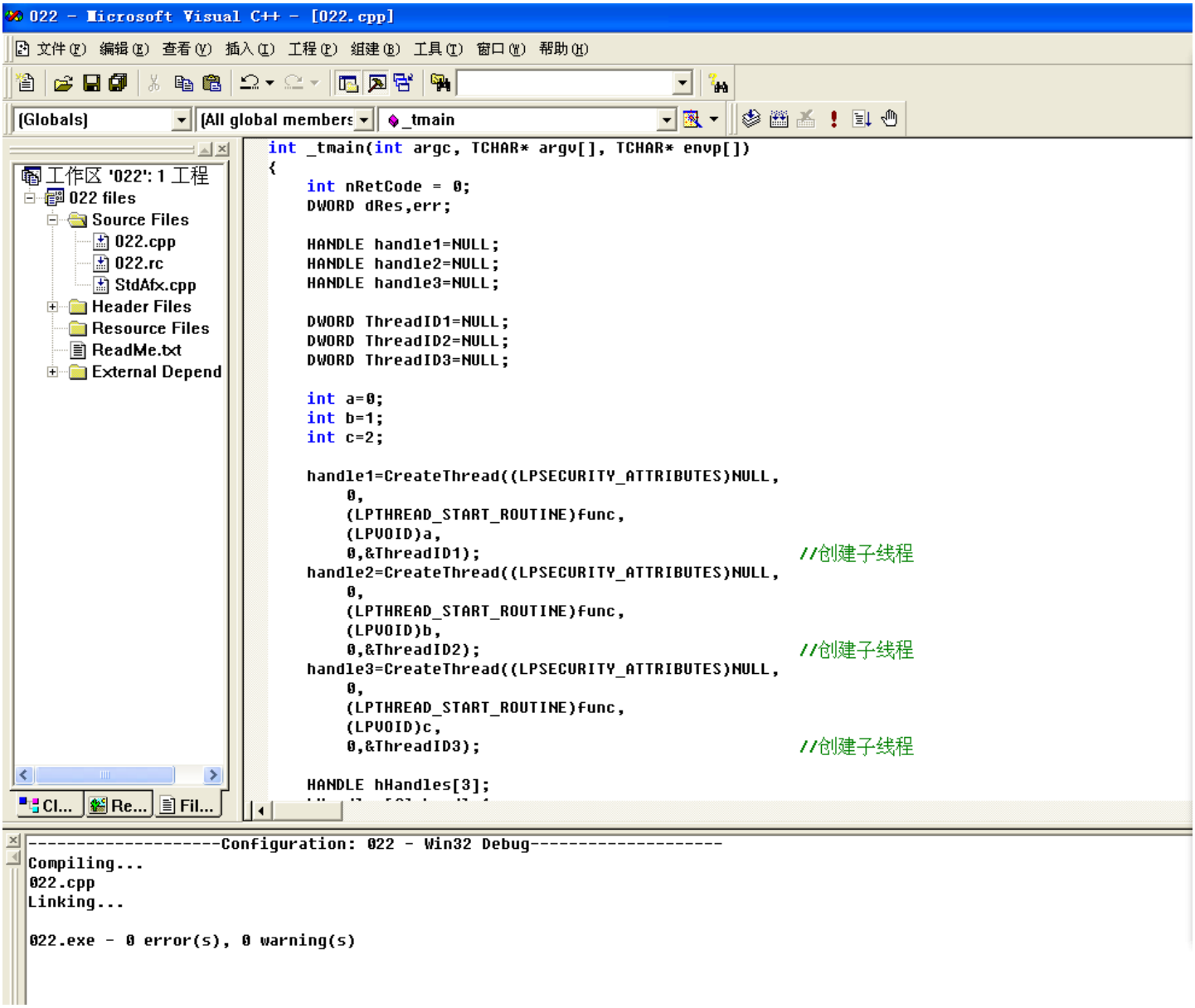
完成了主子两个线程之间的同步，子线程先执行。在主线程中使用系统调用CreateThread（）创建一个子线程。主线程创建一个子线程后进入阻塞状态，直到子线程运行完毕后唤醒主线程。实验成功进行.


02 - Microsoft Visual C++ - [02.cpp]
文件(F) 编辑(E) 查看(V) 插入(I) 工程(P) 组建(B) 工具(T) 窗口(W) 帮助(H)
(Globals)
(All global members
_tmain
工作区 '02': 1 工程
02 files
Source Files
02.cpp
02.rc
StdAfx.cpp
Header Files
Resource Files
ReadMe.txt
External Depend
Cl...
Re...
Fil...
// The one and only application object
CWinApp theApp;
using namespace std;
static HANDLE h1; //线程句柄
static HANDLE hHandle1=NULL; //信号量句柄
void func()
{
printf("开始制作!预计时间5秒。\n");
Sleep(5000);
printf("制作完成!\n");
BOOL rc;
DWORD err;
rc=ReleaseSemaphore(hHandle1,1,NULL);
err=GetLastError();
printf("ReleaseSemaphore err = %d\n",err);
if (rc==0) printf("Semaphore realese Fail!\n");
else printf("Semaphore realese Success! rc=%d\n",rc);
};
int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
int nRetCode = 0;
DWORD dRes,err;
hHandle1=CreateSemaphore(NULL,0,1,"SemaphoreName1");//创建一个信号量
if (hHandle1==NULL) printf("Semaphore Create!\n");
else printf("Semaphore Create Success!\n");
hHandle1=OpenSemaphore(SYNCHRONIZE|SEMAPHORE_MODIFY_STATE,
NULL,
"SemaphoreName1"); //打开信号量
if (hHandle1==NULL) printf("Semaphore Open Fail!\n");
else printf("Semaphore Open Success!\n");
HANDLE h1=NULL;
DWORD dwThreadID1=NULL;



实验二总结：

学会了正确使用等待对象、WaitForSingleObject（）或WaitForMultipleObject（0及信号量对象CreateSemaphore（）、OpenSemaphore（）、ReleaseSemaphore（）等系统调用，进一步理解了线程的同步。


022 - Microsoft Visual C++ - [022.cpp]
文件(F) 编辑(E) 查看(V) 插入(I) 工程(P) 组建(B) 工具(T) 窗口(W) 帮助(H)
(Globals)
(All global members
_tmain
工作区 '022': 1 工程
022 files
Source Files
022.cpp
022.rc
StdAfx.cpp
Header Files
Resource Files
ReadMe.txt
External Depend
Cl...
Re...
Fil...
int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
int nRetCode = 0;
DWORD dRes,err;
HANDLE handle1=NULL;
HANDLE handle2=NULL;
HANDLE handle3=NULL;
DWORD ThreadID1=NULL;
DWORD ThreadID2=NULL;
DWORD ThreadID3=NULL;
int a=0;
int b=1;
int c=2;
handle1=CreateThread((LPSECURITY_ATTRIBUTES)NULL,
0,
(LPTHREAD_START_ROUTINE)func,
(LPVOID)a,
0,&ThreadID1); //创建子线程
handle2=CreateThread((LPSECURITY_ATTRIBUTES)NULL,
0,
(LPTHREAD_START_ROUTINE)func,
(LPVOID)b,
0,&ThreadID2); //创建子线程
handle3=CreateThread((LPSECURITY_ATTRIBUTES)NULL,
0,
(LPTHREAD_START_ROUTINE)func,
(LPVOID)c,
0,&ThreadID3); //创建子线程
HANDLE hHandles[3];
--------------------Configuration: 022 - Win32 Debug--------------------
Compiling...
022.cpp
Linking...
022.exe - 0 error(s), 0 warning(s)