# 实验四 线程的互斥

# 专业 软件工程2 姓名 毛济洲 学号 20182344050

## 一．实验目的

1、进一步掌握windows系统环境下线程的创建与撤销。

2、熟悉线程互斥API

3、使用线程互斥API解决实际问题。

## 二．实验内容

能正确使用临界区对象，临界区对象（CriticalSection）包括初始化临界区（InitializeCriticalSection（））、进入临界区（EnterCriticalSection()）、退出临界区（LeaveCriticalSection（））及删除临界区(DeleteCriticalSection())，进一步理解线程的互斥。

## 三．实验步骤

**（1）       任务分析：**

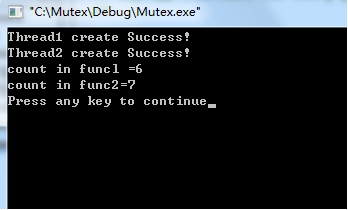
能正确使用临界区对象,包括初始化临界区InitializeCriticalSection()、进入临界区EnterCriticalSection()、退出临界区LeaveCriticalSection()及删除临界区Delete

**（2）       程序设计：**

编写c程序，在主线程中使用InitializeCriticalSection（）初始化临界区，然后建立两个子线程，在两个子线程中使用全局变量count前、后分别使用EnterCriticalSection（）进入临界区及使用LeaveCriticalSection（）退出临界区，等两个子线程运行完毕，主线程使用DeleteCriticalSection（）删除临界区并撤销线程

（3）       **程序结果：**打印程序的输出结果（表明对应的数据）或主要界面图。

  开始时的界面：（如图所示）



## 四．实验总结

1、进一步巩固和掌握第一二次实验线程的创建于撤销和线程的同步的一些基本函数的运用方法和各自的功能的基础上了解线程的互斥。

2、对初始化临界区InitializecritiCalSection（）、进入临界区EnterCriticalSection（）、退出临

界区LeaveCriticalSection（）及删除临界区DeleteCriticalsection（）等命令的使用方法以及他们各自的功能有所了解。

3、对线程互斥的源代码有所了解。

## 五．附录源代码

#include "stdafx.h"

#include "Mutex.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 int count=5;

static HANDLE h1;

static HANDLE h2;

LPCRITICAL\_SECTION hCriticalSection;

CRITICAL\_SECTION Critical;

void func1();

void func2();

int \_tmain(int argc, TCHAR\* argv[], TCHAR\* envp[])

{

int nRetCode = 0;

DWORD dwThreadID1,dwThreadID2;

hCriticalSection=&Critical;

InitializeCriticalSection(hCriticalSection);

h1=CreateThread((LPSECURITY\_ATTRIBUTES)NULL,

0,

(LPTHREAD\_START\_ROUTINE)func1,

(LPVOID)NULL,

0,&dwThreadID1);

if(h1==NULL)printf("Thread1 create Fail!\n");

else printf("Thread1 create Success!\n");

h2=CreateThread((LPSECURITY\_ATTRIBUTES)NULL,

0,

(LPTHREAD\_START\_ROUTINE)func2,

(LPVOID)NULL,

0,&dwThreadID2);

if(h2==NULL)printf("Thread2 create Fail!\n");

else printf("Thread2 create Success!\n");

Sleep(1000);

CloseHandle(h1);

CloseHandle(h2);

DeleteCriticalSection(hCriticalSection);

ExitThread(0);

return nRetCode;

}

void func2()

{

int r2;

EnterCriticalSection(hCriticalSection);

r2=count;

\_sleep(100);

r2=r2+1;

count=r2;

printf("count in func2=%d\n",count);

LeaveCriticalSection(hCriticalSection);

}

void func1()

{

int r1;

EnterCriticalSection(hCriticalSection);

r1=count;

\_sleep(500);

r1=r1+1;

count=r1;

printf("count in funcl =%d\n",count);

LeaveCriticalSection(hCriticalSection);

}