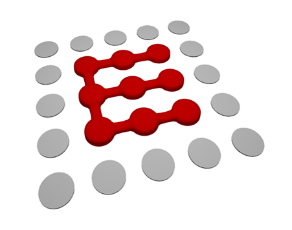
 

**本科实验报告**

**实验名称： 读者写者问题**

|  |  |  |  |
| --- | --- | --- | --- |
| 课程名称： | 操作系统原理 | 实验时间： | 2018/3/28 |
| 任课教师： | 王耀威 | 实验地点： | 理学楼信抗实验中心 |
| 实验教师： | 苏京霞 | 实验类型： | ■ 原理验证  □ 综合设计  □ 自主创新 |
| 学生姓名： | 施念 |
| 学号/班级： | 1120161302/05011609 | 组 号： | 53 |
| 学 院： | 信息与电子学院 | 同组搭档： |  |
| 专 业： | 电子信息工程 | 成 绩： |  |





# 实验二：读者写者问题

## 一、实验目的

1. 通过编写和调试程序以加深对进程、线程管理方案的理解；

2. 熟悉Windows多线程程序设计方法；

## 二、实验题目

在Windows环境下，创建一个控制台进程，此进程包含n个线程。用这n个线程来表示n个读者或写者。每个线程按相应测试数据文件（后面介绍）的要求进行读写操作。用信号量机制分别实现读者优先和写者优先问题。

读者-写者问题的读写操作限制（包括读者优先和写者优先）

1. 写-写互斥：不能有两个写者同时进行写操作
2. 读-写互斥：不能同时有一个线程在读，而另一个线程在写。
3. 读-读允许：可以有一个或多个读者在读。

读者优先的附加限制：如果读者申请进行读操作时已有另一个读者正在进行读操作，则该读者可直接开始读操作。

运行结果显示要求：要求在每个线程创建、发出读写申请、开始读写操作和结束读写操作时分别显示一行提示信息，以确定所有处理都遵守相应的读写操作限制。

测试数据文件包括 n行测试数据，分别描述创建的n个线程是读者还是写者，以及读写操作的开始时间和持续时间。每行测试数据包括四个字段，每个字段间用空格分隔。第1个字段为正整数，表示线程的序号。第2个字段表示线程的角色，R表示读者，W表示写者。第3个字段为一个正数，表示读写开始时间：线程创建后，延迟相应时间（单位为秒）后发出对共享资源的读写申请。第4个字段为一个正数，表示读写操作的延迟时间。当线程读写申请成功后，开始对共享资源进行读写操作，该操作持续相应时间后结束，释放该资源。

下面是一个测试数据文件的例子(在记事本手工录入数据)：

1 R 3 5

2 W 4 5

3 R 5 2

4 R 6 5

5 W 5.1 3

## 三、实验基础知识

#### 1.进程、线程

进程和线程的主要差别在于它们是不同的操作系统资源管理方式。进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮，但在进程切换时，耗费资源较大，效率要差一些。但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程。

**1)** 简而言之,一个程序至少有一个进程,一个进程至少有一个线程.

**2)** 线程的划分尺度小于进程，使得多线程程序的并发性高。

**3)** 另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。

**4)** 线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。

**5)** 从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。这就是进程和线程的重要区别。

## 四、实验设计方法

1) 关系分析。由题目分析读者和写者是互斥的，写者和写者也是互斥的，而读者和读者不存在互斥问题。

2) 整理思路。两个进程，即读者和写者。写者是比较简单的，它和任何进程互斥，用互斥信号量的P操作、V操作即可解决。读者的问题比较复杂，它必须实现与写者互斥的同时还要实现与其他读者的同步，因此，仅仅简单的一对P操作、V操作是无法解决的。那么，在这里用到了一个计数器，用它来判断当前是否有读者读文件。当有读者的时候写者是无法写文件的，此时读者会一直占用文件，当没有读者的时候写者才可以写文件。同时这里不同读者对计数器的访问也应该是互斥的。

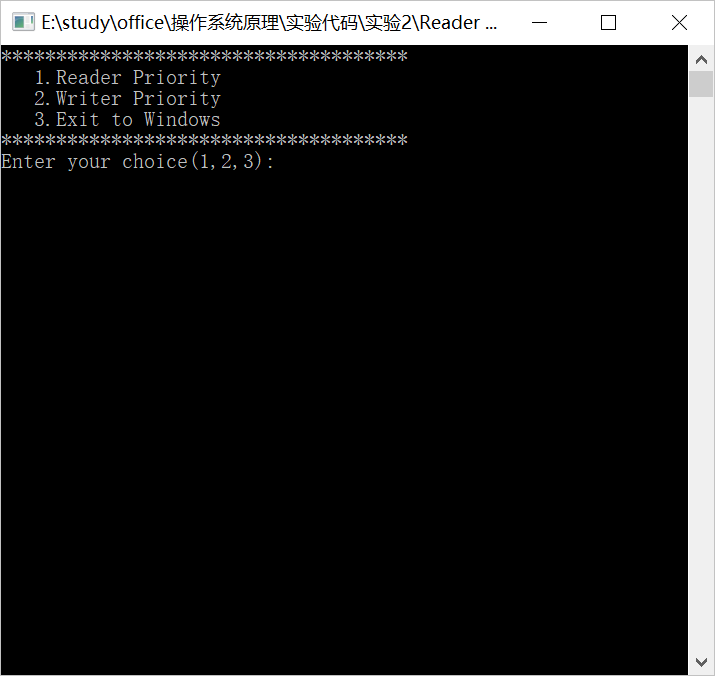
3) 信号量设置。首先设置信号量count为计数器，用来记录当前读者数量，初值为0; 设置mutex为互斥信号量，用于保护更新count变量时的互斥；设置互斥信号量rw用于保证读者和写者的互斥访问。

读者优先。读者优先指的是除非有写者在写文件，否则读者不需要等待。所以可以用一个整数变量readcount记录当前的读者数目，用于确定是否需要释放正在等待的写者进程（readcount=0时，表明所有的读者读完，需要释放写者等待队列中的一个写者）。每当一个读者开始读文件时，必须修改readcount变量。因此需要一个互斥对象mutex来实现对全局变量readcount修改时的互斥。

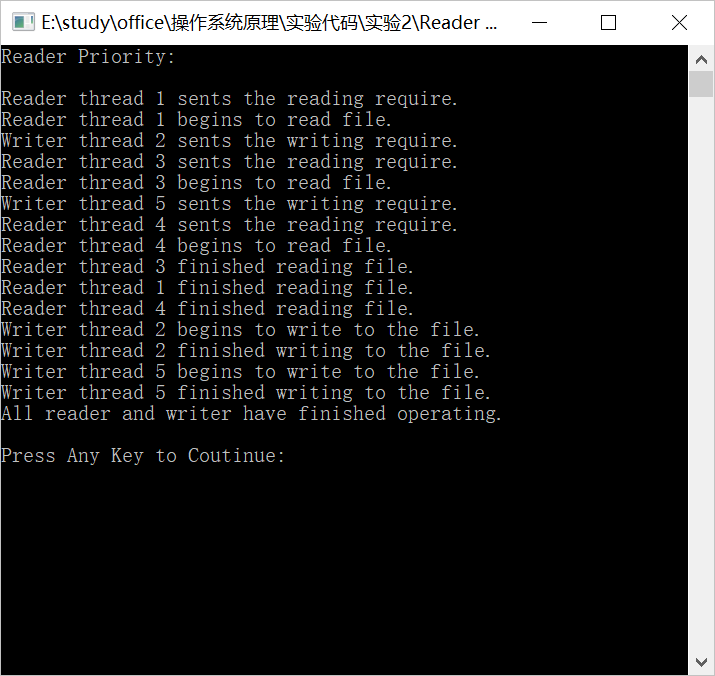
写者优先。写者优先与读者优先相类似。不同之处在于一旦一个写者到来，它应该尽快对文件进行写操作，如果有一个写者在等待，则新到来的读者不允许进行读操作。为此应当填加一个整形变量writecount，用于记录正在等待的写者的数目，当writecount=0时，才可以释放等待的读者线程队列。

## 五、实验结果及数据分析

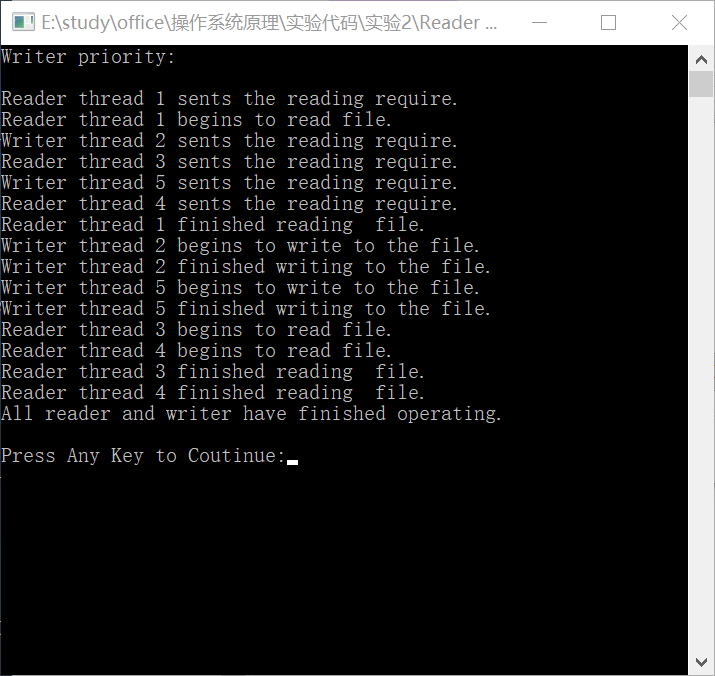
#### 1. 图像：



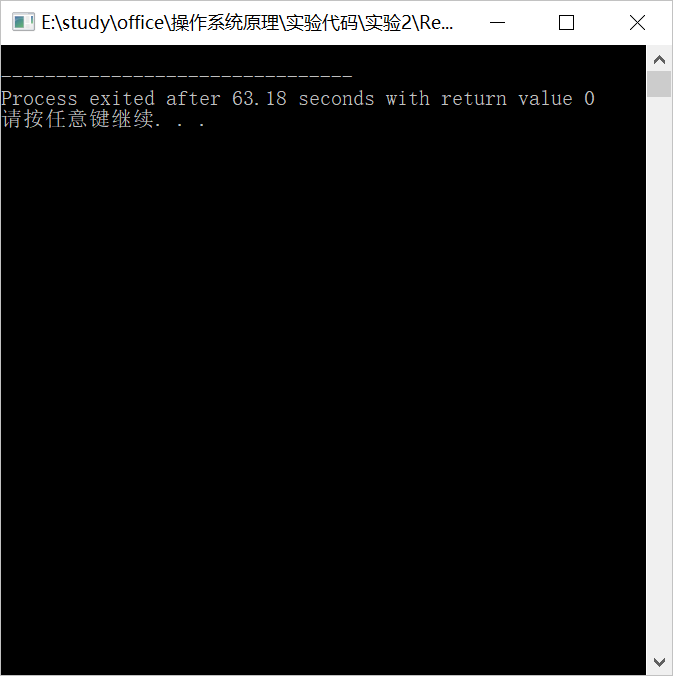
（开始界面）



（读者优先）



（写者优先）



（退出）

#### 2. 分析

由运行结果可知：

1. 写-写互斥：不存在两个写者同时进行写操作
2. 读-写互斥：不存在有一个线程在读，而另一个线程在写。
3. 读-读允许：存在一个或多个读者在读。

## 六、总结

此次实验我加深了对读者写者问题的理解，同时实践操作也使我对优先级、互斥有了更加深刻的了解。

读者的问题较为简单，但是写者就有那么一点难度，不过运用类比的方法，结合老师提供的帮助，还是能很好解决的。

其次，关于测试文件的编写以及数据类型的强制转换也值得注意。

## 七、附录

**代码清单**：

#include "windows.h"

#include <conio.h>

#include <stdlib.h>

#include <io.h>

#include <string.h>

#include <stdio.h>

#include "fstream"

#define READER 'R' //读者

#define WRITER 'W' //写者

#define INTE\_PER\_SEC 1000 //每秒时钟中断的数目

#define MAX\_THREAD\_NUM 64 //最大线程数目

#define MAX\_FILE\_NUM 32 //最大文件数目

#define MAX\_STR\_LEN 32 //字符串的长度

using namespace std;

int readcount=0; //读者数目

int writecount=0; //写者数目

CRITICAL\_SECTION RP\_Write; //临界区

CRITICAL\_SECTION cs\_Write;

CRITICAL\_SECTION cs\_Read;

struct ThreadInfo

{

int serial; //线程序号

char entity; //线程类别(判断是读者线程还是写者线程)

double delay; //线程延迟时间

double persist; //线程读写操作时间

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//读者优先---读者线程

//P:读者线程信息

void RP\_ReaderThread(void \*p)

{

//互斥变量

HANDLE h\_Mutex;

h\_Mutex=OpenMutex(MUTEX\_ALL\_ACCESS,FALSE,"mutex\_for\_readcount");

DWORD wait\_for\_mutex; //等待互斥变量所有权

DWORD m\_delay; //延迟时间

DWORD m\_persist; //读文件持续时间

int m\_serial; //线程序号

//从参数中获得信息

m\_serial=((ThreadInfo\*)(p))->serial;

m\_delay=(DWORD)(((ThreadInfo\*)(p))->delay \*INTE\_PER\_SEC);

m\_persist=(DWORD)(((ThreadInfo\*)(p))->persist \*INTE\_PER\_SEC);

Sleep(m\_delay); //延迟等待

printf("Reader thread %d sents the reading require.\n",m\_serial);

//等待互斥信号,保证对ReadCount 的访问、修改互斥

wait\_for\_mutex=WaitForSingleObject(h\_Mutex,-1);

//读者数目增加

readcount++;

if(readcount==1)

{

//第一个读者,等待资源

EnterCriticalSection(&RP\_Write);

}

ReleaseMutex(h\_Mutex); //释放互斥信号

//读文件

printf("Reader thread %d begins to read file.\n",m\_serial);

Sleep(m\_persist);

//退出线程

printf("Reader thread %d finished reading file.\n",m\_serial);

//等待互斥信号,保证对ReadCount的访问,修改互斥

wait\_for\_mutex=WaitForSingleObject(h\_Mutex,-1);

//读者数目减少

readcount--;

if(readcount==0)

{

//如果所有的读者读完,唤醒写者

LeaveCriticalSection(&RP\_Write);

}

ReleaseMutex(h\_Mutex); //释放互斥信号

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//读者优先---写者线程

//P:写者线程信息

void RP\_WriterThread(void \*p)

{

DWORD m\_delay; //延迟时间

DWORD m\_persist; //写文件持续时间

int m\_serial; //线程序号

// 从参数中获得信息

m\_serial=((ThreadInfo\*)(p))->serial ;

m\_delay=(DWORD)(((ThreadInfo\*)(p))->delay \*INTE\_PER\_SEC);

m\_persist=(DWORD)(((ThreadInfo\*)(p))->persist \*INTE\_PER\_SEC);

Sleep(m\_delay);

printf("Writer thread %d sents the writing require.\n",m\_serial);

//等待资源

EnterCriticalSection(&RP\_Write);

//写文件

printf("Writer thread %d begins to write to the file.\n",m\_serial);

Sleep(m\_persist);

//退出线程

printf("Writer thread %d finished writing to the file.\n",m\_serial);

//释放资源

LeaveCriticalSection(&RP\_Write);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//读者优先处理函数

//file:文件名

void ReaderPriority(char \*file)

{

DWORD n\_thread=0; //线程数目

DWORD thread\_ID; //线程ID

DWORD wait\_for\_all; //等待所有线程结束

//互斥对象

HANDLE h\_Mutex;

h\_Mutex=CreateMutex(NULL,FALSE,"mutex\_for\_readcount");

//线程对象的数组

HANDLE h\_Thread[MAX\_THREAD\_NUM];

ThreadInfo thread\_info[MAX\_THREAD\_NUM];

readcount=0; //初始化readcount

InitializeCriticalSection(&RP\_Write); //初始化临界区

ifstream inFile;

inFile.open (file);

printf("Reader Priority:\n\n");

while(inFile)

{

//读入每一个读者,写者的信息

inFile>>thread\_info[n\_thread].serial;

inFile>>thread\_info[n\_thread].entity;

inFile>>thread\_info[n\_thread].delay;

inFile>>thread\_info[n\_thread++].persist;

inFile.get();

}

for(int i=0;i<(int)(n\_thread);i++)

{

if(thread\_info[i].entity==READER||thread\_info[i].entity =='r')

{

//创建读者进程

h\_Thread[i]=CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)(RP\_ReaderThread),&thread\_info[i],0,&thread\_ID);

}

else

{

//创建写线程

h\_Thread[i]=CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)(RP\_WriterThread),&thread\_info[i],0,&thread\_ID);

}

}

//等待所有的线程结束

wait\_for\_all=WaitForMultipleObjects(n\_thread,h\_Thread,TRUE,-1);

printf("All reader and writer have finished operating.\n");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//写者优先---读者线程

//P:读者线程信息

void WP\_ReaderThread(void \*p)

{

//互斥变量

HANDLE h\_Mutex1;

h\_Mutex1=OpenMutex(MUTEX\_ALL\_ACCESS,FALSE,"mutex1");

HANDLE h\_Mutex2;

h\_Mutex2=OpenMutex(MUTEX\_ALL\_ACCESS,FALSE,"mutex2");

DWORD wait\_for\_mutex1; //等待互斥变量所有权

DWORD wait\_for\_mutex2;

DWORD m\_delay; //延迟时间

DWORD m\_persist; //读文件持续时间

int m\_serial; //线程的序号

//从参数中得到信息

m\_serial=((ThreadInfo\*)(p))->serial ;

m\_delay=(DWORD)(((ThreadInfo\*)(p))->delay \*INTE\_PER\_SEC);

m\_persist=(DWORD)(((ThreadInfo\*)(p))->persist \*INTE\_PER\_SEC);

Sleep(m\_delay); //延迟等待

printf("Reader thread %d sents the reading require.\n",m\_serial);

wait\_for\_mutex1=WaitForSingleObject(h\_Mutex1,-1);

//进入读者临界区

EnterCriticalSection(&cs\_Read);

//阻塞互斥对象Mutex2,保证对readCount的访问和修改互斥

wait\_for\_mutex2=WaitForSingleObject(h\_Mutex2,-1);

//修改读者的数目

readcount++;

if(readcount==1)

{

// 如果是第1个读者,等待写者写完

EnterCriticalSection(&cs\_Write);

}

ReleaseMutex(h\_Mutex2);// 释放互斥信号 Mutex2

//让其他读者进去临界区

LeaveCriticalSection(&cs\_Read);

ReleaseMutex(h\_Mutex1);

//读文件

printf("Reader thread %d begins to read file.\n",m\_serial);

Sleep(m\_persist);

//退出线程

printf("Reader thread %d finished reading file.\n",m\_serial);

//阻塞互斥对象Mutex2,保证对readcount的访问,修改互斥

wait\_for\_mutex2=WaitForSingleObject(h\_Mutex2,-1);

readcount--;

if(readcount==0)

{

//最后一个读者,唤醒写者

LeaveCriticalSection(&cs\_Write);

}

ReleaseMutex(h\_Mutex2); //释放互斥信号

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//写者优先---写者线程

//P:写者线程信息

void WP\_WriterThread(void \*p)

{

DWORD wait\_for\_mutex3; //互斥变量

DWORD m\_delay; //延迟时间

DWORD m\_persist; //写文件持续时间

int m\_serial; //线程序号

HANDLE h\_Mutex3;

h\_Mutex3=OpenMutex(MUTEX\_ALL\_ACCESS,FALSE,"mutex3");

//从参数中获得信息

m\_serial=((ThreadInfo\*)(p))->serial ;

m\_delay=(DWORD)(((ThreadInfo\*)(p))->delay \*INTE\_PER\_SEC);

m\_persist=(DWORD)(((ThreadInfo\*)(p))->persist \*INTE\_PER\_SEC);

Sleep(m\_delay); //延迟等待

printf("Writer thread %d sents the reading require.\n",m\_serial);

wait\_for\_mutex3=WaitForSingleObject(h\_Mutex3,-1);

writecount++; //修改写者数目

if(writecount==1)

{

EnterCriticalSection(&cs\_Read);

}

ReleaseMutex(h\_Mutex3);

//进入写者临界区

EnterCriticalSection(&cs\_Write);

printf("Writer thread %d begins to write to the file.\n",m\_serial);

Sleep(m\_persist);

printf("Writer thread %d finished writing to the file.\n",m\_serial);

LeaveCriticalSection(&cs\_Write);

wait\_for\_mutex3=WaitForSingleObject(h\_Mutex3,-1);

writecount--;

if(writecount==0)

{

LeaveCriticalSection(&cs\_Read);

}

ReleaseMutex(h\_Mutex3);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//写者优先处理函数

// file:文件名

void WriterPriority(char \* file)

{

DWORD n\_thread=0;

DWORD thread\_ID;

DWORD wait\_for\_all;

HANDLE h\_Mutex1;

h\_Mutex1=CreateMutex(NULL,FALSE,"mutex1");

HANDLE h\_Mutex2;

h\_Mutex2=CreateMutex(NULL,FALSE,"mutex2");

HANDLE h\_Mutex3;

h\_Mutex3=CreateMutex(NULL,FALSE,"mutex3");

HANDLE h\_Thread[MAX\_THREAD\_NUM];

ThreadInfo thread\_info[MAX\_THREAD\_NUM];

readcount=0;

writecount=0;

InitializeCriticalSection(&cs\_Write);

InitializeCriticalSection(&cs\_Read);

ifstream inFile;

inFile.open (file);

printf("Writer priority:\n\n");

while(inFile)

{

inFile>>thread\_info[n\_thread].serial;

inFile>>thread\_info[n\_thread].entity;

inFile>>thread\_info[n\_thread].delay;

inFile>>thread\_info[n\_thread++].persist;

inFile.get();

}

for(int i=0;i<(int)(n\_thread);i++)

{

if(thread\_info[i].entity==READER||thread\_info[i].entity =='r')

{

//创建读者进程

h\_Thread[i]=CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)(WP\_ReaderThread),&thread\_info[i],0,&thread\_ID);

}

else

{

//创建写线程

h\_Thread[i]=CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)(WP\_WriterThread),&thread\_info[i],0,&thread\_ID);

}

}

//等待所有的线程结束

wait\_for\_all=WaitForMultipleObjects(n\_thread,h\_Thread,TRUE,-1);

printf("All reader and writer have finished operating.\n");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//主函数

int main(int argc,char \*argv[])

{

char ch;

while(true)

{

//打印提示信息

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" 1.Reader Priority\n");

printf(" 2.Writer Priority\n");

printf(" 3.Exit to Windows\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Enter your choice(1,2,3): ");

//如果输入信息不正确，继续输入

do{

ch=(char)\_getch();

}while(ch!='1'&&ch!='2'&&ch!='3');

system("cls");

//选择3，返回

if(ch=='3')

return 0;

//选择1，读者优先

else if(ch=='1')

ReaderPriority((char\*)"thread.txt");

//选择2，写者优先

else

WriterPriority((char\*)"thread.txt");

//结束

printf("\nPress Any Key to Coutinue:");

\_getch();

system("cls");

}

return 0;

}