实验一 进程同步

学号：2017526019　　　　姓名：刘禾子　　　　班级：2015211307

一、 实验要求

在 Windows 环境下，创建一个包含 n 个线程的控制进程。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要求，进行读写操作。请用信号量机制分别实现读者优先和写者优先的读者-写者问题。

读者-写者问题的读写操作限制：

1）写-写互斥；

2）读-写互斥；

3）读-读允许；

读者优先的附加限制：如果一个读者申请进行读操作时已有另一读者正在进行读操作，则该读者可直接开始读操作。

写者优先的附加限制：如果一个读者申请进行读操作时已有另一写者在等待访问共享资源，则该读者必须等到没有写者处于等待状态后才能开始读操作。

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

二、 测试数据文件格式

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，各字段间用空格分隔。

第一字段为一个正整数，表示线程序号。第一字段表示相应线程角色，R 表示读者是，W 表示写者。

第二字段为一个正数，表示读写操作的开始时间。线程创建后，延时相应时间（单位为秒）后发出对共享资源的读写申请。

第三字段为一个正数，表示读写操作的持续时间。当线程读写申请成功后，开始对共享资源的读写操作，该操作持续相应时间后结束，并释放共享资源。

下面是一个测试数据文件的例子：

```
1 R 3 5
2 W 4 5
3 R 5 2
4 R 6 5
5 W 5.1 3
```

三、 设计过程

1. 与实验相关的 API

线程控制：

CreateThread 完成线程创建，在调用进程的地址空间上创建一个线程，以执行指定的函数；它的返回值为所创建线程的句柄。

```
HANDLE CreateThread（LPSECURITY_ATTRIBUTES
lpThreadAttributes, // SD
```

```
DWORD dwStackSize，// initial stack size
LPTHREAD_START_ROUTINE lpStartAddress，// thread
function
LPVOID lpParameter，// thread argument
DWORD dwCreationFlags，// creation option
LPDWORD lpThreadId // thread identifier
）；
```

ExitThread 用于结束当前线程。

```
VOID ExitThread（
DWORD dwExitCode // exit code for this thread
）；
```

Sleep 可在指定的时间内挂起当前线程。

```
VOID Sleep（
DWORD dwMilliseconds // sleep time
）；
```

信号量控制：

CreateMutex 创建一个互斥对象，返回对象句柄；

```
HANDLE CreateMutex（
LPSECURITY_ATTRIBUTES lpMutexAttributes，// SD
BOOL bInitialOwner，// initial owner
LPCTSTR lpName // object name
）；
```

OpenMutex 打开并返回一个已存在的互斥对象句柄用于后续访问；

```
HANDLE OpenMutex（
DWORD dwDesiredAccess，// access
BOOL bInheritHandle，// inheritance option
LPCTSTR lpName // object name
）；
```

ReleaseMutex 释放对互斥对象的占用，使之成为可用。

```
BOOL ReleaseMutex（
HANDLE hMutex // handle to mutex
）；
```

WaitForSingleObject 可在指定的时间内等待指定对象为可用状态；

```
DWORD WaitForSingleObject（
HANDLE hHandle，// handle to object
DWORD dwMilliseconds // time-out interval
）；
```

2. 程序说明

程序由入口函数Main开始，打印出菜单，选择1则选择读者优先，调用ReadPriority(“thread.dat”)函数；选择2则选择写者优先，调用WriterPirority(“thread.dat”)函数；选择3则退出。

读者优先：

ReaderPriority函数首先读取目标文件Thread.dat，为每一行请求创建一个线程，其中读请求创建读者线程，调用RP_ReaderThread函

数，写请求创建写者线程，调用 RP_WriterThread 函数。

RP_ReaderThread 函数的实现如下：

```
P（mutex）；
read_count++;
If(read_count==1)
     P(&RP_Write);
V(mutex);
读临界区…
P(mutex);
read_count--;
if(read_count==0)
  V(&RP_Write);
V(mutex);
```

RP_WriterThread 函数的实现如下：

```
P(&RP_Write);
写临界区…
V(&RP_Write);
```

写者优先：

WriterPriority 函数首先读取目标文件 Thread.dat,为每一行请求创建一个线程，其中读请求创建读者线程，调用 WP_ReaderThread 函数，写请求创建写者线程，调用 WP_WriterThread 函数。

WP_ReaderThread 函数实现如下：

```
P(mutex1);
P(&cs_Read);
P(mutex2);
read_count++;
if(read_count==1)
   P(&cs_Write);
V(mutex2);
V(&cs_Read);
V(mutex1);
读临界区…
P(mutex2);
read_count--;
if(read_count==0)
  V(&cs_Write);
V(mutex2);
```

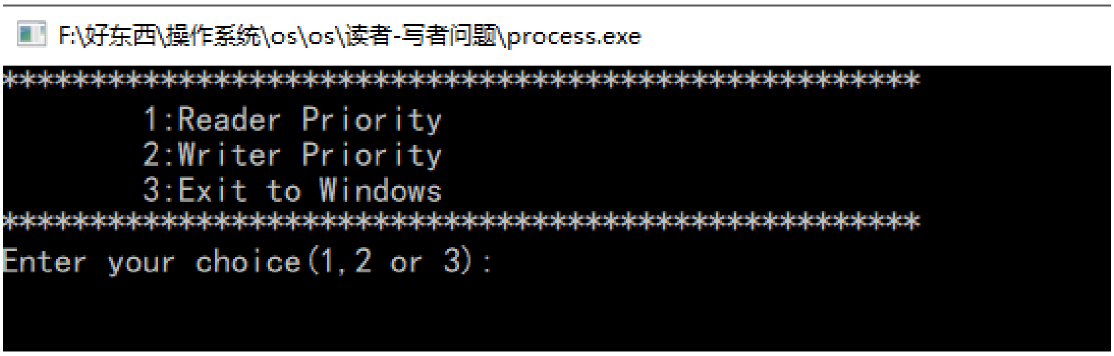
WP_WriterThread 函数实现如下：

```
P（mutex3）；
wirte_count++;
if(write_count==1)
   P(&cs_Read);
V(mutex3);
P(&cs_Write);
```

```
写临界区…
V(&cs_Write);
P(mutex3);
write_count--;
if(write_count==0)
   V(&cs_Read);
V(mutex3);
```

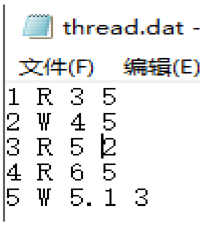
四、 实验结果

初始菜单界面：



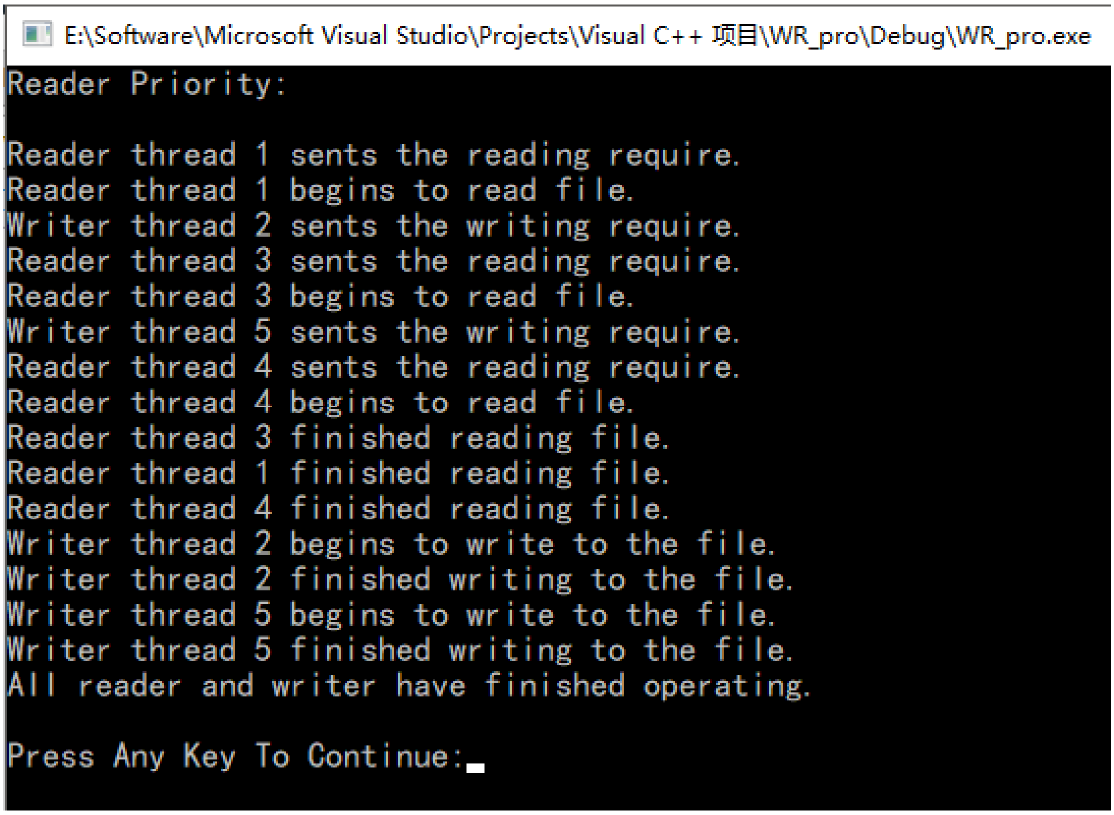
```
****************************************************
        1:Reader Priority
        2:Writer Priority
        3:Exit to Windows
****************************************************
Enter your choice(1,2 or 3):
```

原本的数据：



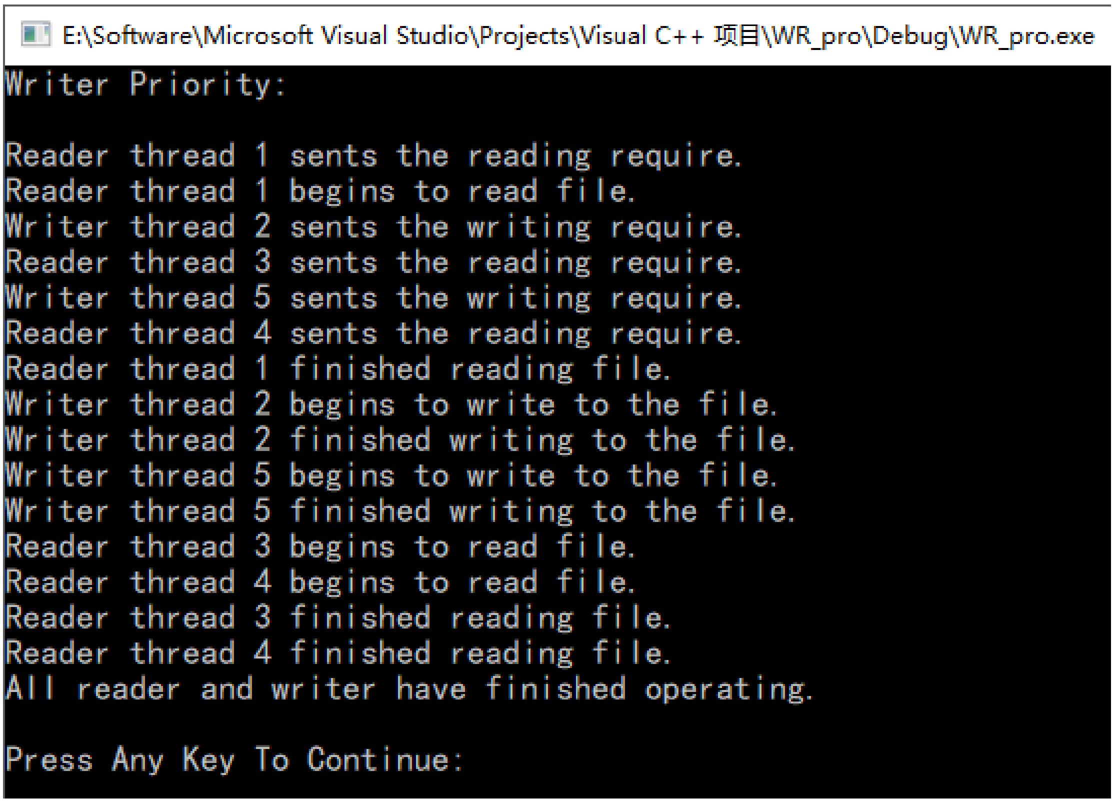
```
1 R 3 5
2 W 4 5
3 R 5 2
4 R 6 5
5 W 5.1 3
```

选择 1：读者优先


E:\Software\Microsoft Visual Studio\Projects\Visual C++ 项目\WR_pro\Debug\WR_pro.exe

```
Reader Priority:

Reader thread 1 sents the reading require.
Reader thread 1 begins to read file.
Writer thread 2 sents the writing require.
Reader thread 3 sents the reading require.
Reader thread 3 begins to read file.
Writer thread 5 sents the writing require.
Reader thread 4 sents the reading require.
Reader thread 4 begins to read file.
Reader thread 3 finished reading file.
Reader thread 1 finished reading file.
Reader thread 4 finished reading file.
Writer thread 2 begins to write to the file.
Writer thread 2 finished writing to the file.
Writer thread 5 begins to write to the file.
Writer thread 5 finished writing to the file.
All reader and writer have finished operating.

Press Any Key To Continue:_
```

线程 1 首先在时刻 3 发送读请求持续时间为 5，并开始进行读操作；
线程 2 在时刻 4 发送写请求，由于此时有线程在进行读操作所以线程 2 将自己挂起；
线程 3 在时刻 5 发送读请求，持续时间为 2，并开始进行读操作；
线程 5 在时刻 5.1 发送写请求，持续时间为 3，由于此时有线程正在进行读操作，所以线程 5 将自己挂起；
线程 4 在时刻 6 发送读请求，持续时间为 5，并且开始进行读操作；
在时刻 7，线程 3 完成读操作，在时刻 8，线程 1 完成读操作，在时刻 11，线程 4 完成读操作，此时所有申请读操作的线程都已完成；
线程 2 开始进行写操作，持续 5 个时刻完成操作；
线程 5 开始进行写操作，持续 3 个时刻完成操作；（不允许同时有两个写者对临界区进行操作）。
至此，所有的读者和写者都完成操作。
选择 2：写者优先



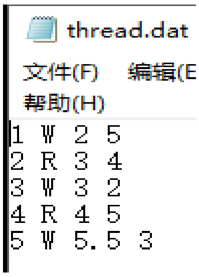
E:\Software\Microsoft Visual Studio\Projects\Visual C++ 项目\WR_pro\Debug\WR_pro.exe

```
Writer Priority:

Reader thread 1 sents the reading require.
Reader thread 1 begins to read file.
Writer thread 2 sents the writing require.
Reader thread 3 sents the reading require.
Writer thread 5 sents the writing require.
Reader thread 4 sents the reading require.
Reader thread 1 finished reading file.
Writer thread 2 begins to write to the file.
Writer thread 2 finished writing to the file.
Writer thread 5 begins to write to the file.
Writer thread 5 finished writing to the file.
Reader thread 3 begins to read file.
Reader thread 4 begins to read file.
Reader thread 3 finished reading file.
Reader thread 4 finished reading file.
All reader and writer have finished operating.

Press Any Key To Continue:
```

线程 1 在时刻 3 发送读请求，持续时间为 5，此时没有写者在临界区进行操作，所以线程 1 开始进行读操作；
线程 2 在时刻 4 发送写请求，持续时间为 5，此时线程 1 正在进行读操作，所以线程 2 将自己挂起；
线程 3 在时刻 5 发送读请求，持续时间为 2，此时线程 1 正在进行读操作，所以线程 3 将自己挂起；
线程 5 在时刻 5.1 发送写请求，持续时间为 3，此时线程 1 正在进行读操作，所以线程 5 将自己挂起；
线程 4 在时刻 6 发送读请求，持续时间为 5，此时线程 1 正在进行读操作，所以线程 4 将自己挂起；

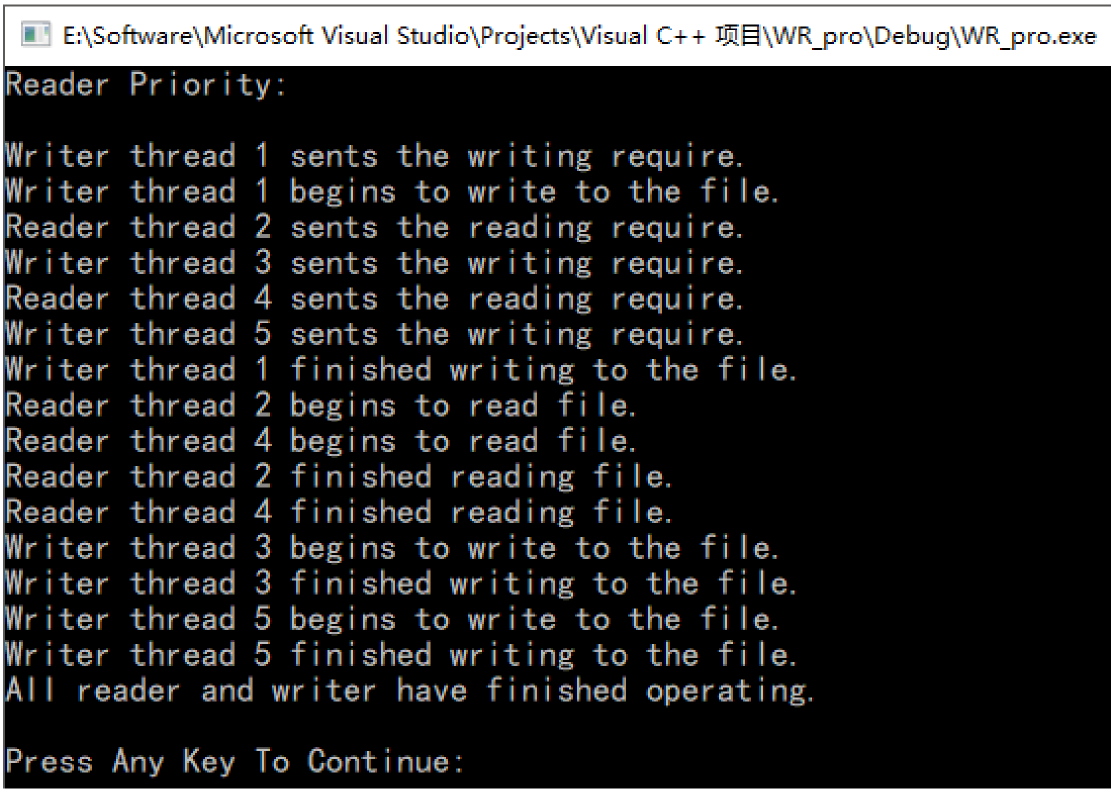
线程 1 在时刻 8 完成读操作，之后线程 2 和 5 陆续完成写操作，然后线程 3 和 4 都开始进行读操作，并结束操作。

另外测试数据：



```
1 W 2 5
2 R 3 4
3 W 3 2
4 R 4 5
5 W 5.5 3
```

选择 1：读者优先


E:\Software\Microsoft Visual Studio\Projects\Visual C++ 项目\WR_pro\Debug\WR_pro.exe

```
Reader Priority:

Writer thread 1 sents the writing require.
Writer thread 1 begins to write to the file.
Reader thread 2 sents the reading require.
Writer thread 3 sents the writing require.
Reader thread 4 sents the reading require.
Writer thread 5 sents the writing require.
Writer thread 1 finished writing to the file.
Reader thread 2 begins to read file.
Reader thread 4 begins to read file.
Reader thread 2 finished reading file.
Reader thread 4 finished reading file.
Writer thread 3 begins to write to the file.
Writer thread 3 finished writing to the file.
Writer thread 5 begins to write to the file.
Writer thread 5 finished writing to the file.
All reader and writer have finished operating.

Press Any Key To Continue:
```

线程 1 在时刻 2 发送写请求，持续时间为 5，此时没有读者在进行操作所以线程 1 开始进行写操作；

线程 2 在时刻 3 发送读请求，持续时间为 4，此时线程 1 在进行写操作，所以线程 2 将自己挂起；

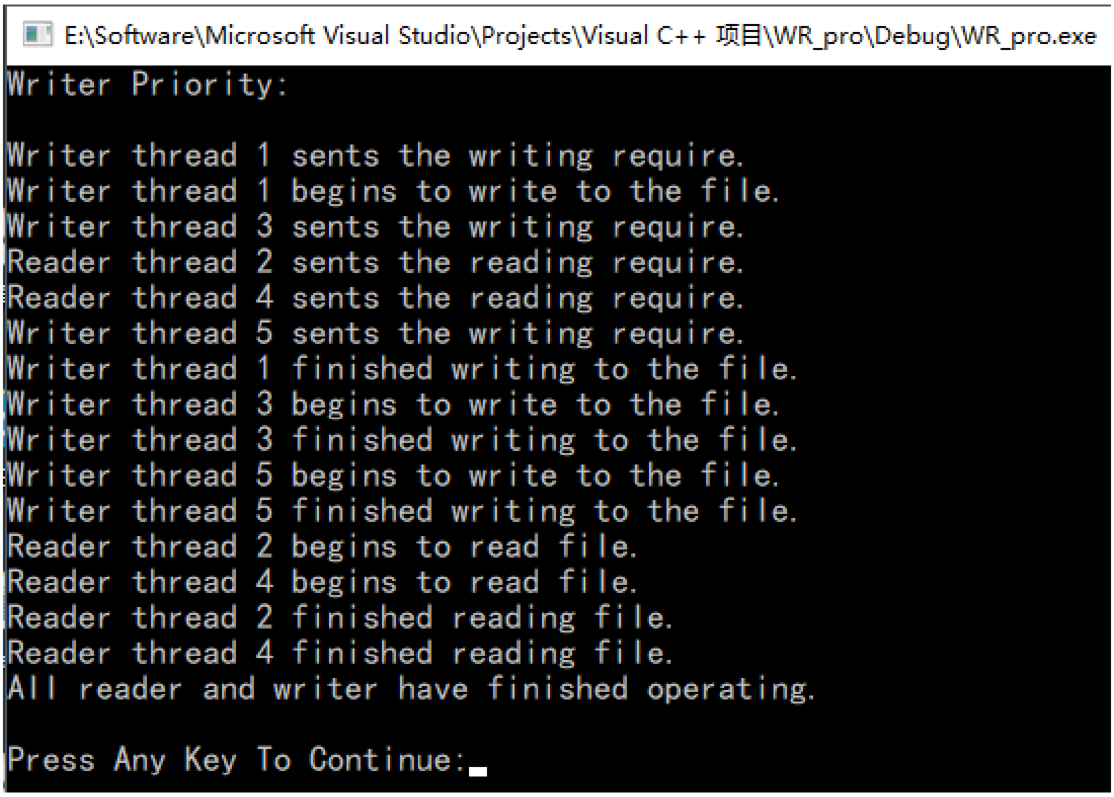
线程 3 在时刻 3 发送写请求，持续时间为 2，此时线程 1 在进行写操作，所以线程 3 将自己挂起；

线程 4 在时刻 4 发送读请求，持续时间为 5，此时线程 1 在进行写操作，所以线程 4 将自己挂起；

线程 5 在时刻 5.5 发送写请求，持续时间为 3，此时线程 1 在进行写操作，所以线程 5 将自己挂起；

线程 1 在时刻 7 完成写操作，线程 2 和 4 均开始进行读操作并结束操作之后线程 3 和线程 5 分别开始进行写操作，最后所有线程均完成操作。

选择 2：写者优先



E:\Software\Microsoft Visual Studio\Projects\Visual C++ 项目\WR_pro\Debug\WR_pro.exe

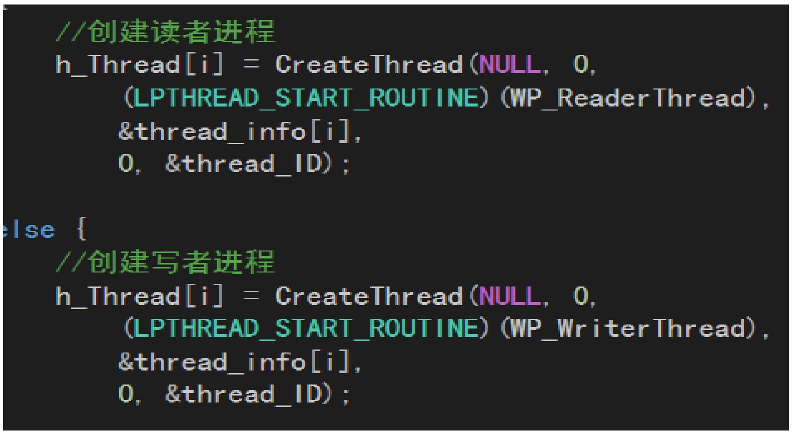
```
Writer Priority:

Writer thread 1 sents the writing require.
Writer thread 1 begins to write to the file.
Writer thread 3 sents the writing require.
Reader thread 2 sents the reading require.
Reader thread 4 sents the reading require.
Writer thread 5 sents the writing require.
Writer thread 1 finished writing to the file.
Writer thread 3 begins to write to the file.
Writer thread 3 finished writing to the file.
Writer thread 5 begins to write to the file.
Writer thread 5 finished writing to the file.
Reader thread 2 begins to read file.
Reader thread 4 begins to read file.
Reader thread 2 finished reading file.
Reader thread 4 finished reading file.
All reader and writer have finished operating.

Press Any Key To Continue:
```

线程 1 在时刻 2 发送写请求，持续时间为 5，并开始进行写操作；
线程 2 在时刻 3 发送读请求，持续时间为 4，此时线程 1 在进行写操作，所以线程 2 将自己挂起；
线程 3 在时刻 3 发送写请求，持续时间为 2，此时线程 1 在进行写操作，所以线程 3 将自己挂起；
线程 4 在时刻 4 发送读请求，持续时间为 5，此时线程 1 在进行写操作，所以线程 4 将自己挂起；
线程 5 在时刻 5.5 发送写请求，持续时间为 3，此时线程 1 在进行写操作，所以线程 5 将自己挂起；
线程 1 完成写操作，线程 3 和 5 依次进行并完成写操作，之后线程 2 和 4 均开始进行读操作，最后所有读者写者完成操作。

五、 源代码

源代码中原本有几处错误导致无法运行：

1. 未加入命名区间 using namespace std;导致 ifstream 无法识别
2. 在写者优先处理程序中，部分参数有误导致读者与写者优先两个函数输出结果一样



```
    //创建读者进程
    h_Thread[i] = CreateThread(NULL, 0,
        (LPTHREAD_START_ROUTINE)(WP_ReaderThread),
        &thread_info[i],
        0, &thread_ID);

lse {
    //创建写者进程
    h_Thread[i] = CreateThread(NULL, 0,
        (LPTHREAD_START_ROUTINE)(WP_WriterThread),
        &thread_info[i],
        0, &thread_ID);
```

将 RP_ReaderThread 修改为 WP_ReaderThread 后运行正常

```
#include  "windows.h"
#include  <conio.h>
#include  <stdlib.h>
#include  <fstream>
#include  <io.h>
#include  <string.h>
#include  <stdio.h>

#define  READER  'R'  //读者
#define  WRITER  'W' //写者
#define  INTE_PER_SEC  1000 //每秒时钟中断数目
#define  MAX_THREAD_NUM  64//最大线程数目
#define  MAX_FILE_NUM  32//最大数据文件数目
#define  MAX_STR_LEN  32//字符串长度
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//线程结构
{
    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
    double  persist;//线程读写操作持续时间
};

/////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////
//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, (LPCWSTR)"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、修改和互斥
    wait_for_mutex = WaitForSingleObject(h_Mutex, -1);//P操作
                                                            //读者数目增加
    readcount++;
    if (readcount == 1)
    {
        //读第一个读者，等待资源
        EnterCriticalSection(&RP_Write);
    }
    ReleaseMutex(h_Mutex);//V操作
    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);//P操作
                                                            //读者数目减少
    readcount--;
    if (readcount == 0)
    {
        //如果读者全部读完，唤醒写者
        LeaveCriticalSection(&RP_Write);
    }
    ReleaseMutex(h_Mutex);//V操作
}

///////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////
//读者优先-写者线程
//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);//延迟等待

    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, (LPCWSTR)"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);//打开文件
    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 {
            //创建写者进程
            h_Thread[i] = CreateThread(NULL, 0,
                (LPTHREAD_START_ROUTINE)(RP_WriterThread),
                &thread_info[i],
                0, &thread_ID);
        }
    }
    //等待所有线程结束
    wait_for_all = WaitForMultipleObjects(n_thread, h_Thread, TRUE, -1);
    printf("All reader and writer have finished operating.\n");
}

///////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////
//写者优先--读者进程
//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, (LPCWSTR)"mutex1");
HANDLE h_mutex2;
h_mutex2 = OpenMutex(MUTEX_ALL_ACCESS, FALSE, (LPCWSTR)"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);
wait_for_mutex1 = WaitForSingleObject(h_mutex1, -1);//P操作

                                                         //进入读者临界区
EnterCriticalSection(&cs_Read);//P操作

                                  //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);//P操作
                                                         //修改读者数目
readcount++;
if (readcount == 1)
{
    //如果是第一个读者，等待写者写完
    EnterCriticalSection(&cs_Write);
}
ReleaseMutex(h_mutex2);//V操作
                         //让其他读者进入临界区
LeaveCriticalSection(&cs_Read);
ReleaseMutex(h_mutex1);//V操作
                         //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);//P操作
readcount--;
```

```
    if (readcount == 0)
    {
        //如果所有读者读完，唤醒写者
        LeaveCriticalSection(&cs_Write);
    }
    ReleaseMutex(h_mutex2);//V操作
}

////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////
//写者优先--写者线程
//p：写者线程信息

void WP_WriterThread(void* p)
{
    DWORD wait_for_mutex3;
    DWORD m_delay;//延迟时间
    DWORD m_persist;//写文件持续时间
    int m_serial;//线程序号

                   //互斥对象
    HANDLE h_mutex3;
    h_mutex3 = OpenMutex(MUTEX_ALL_ACCESS, FALSE, (LPCWSTR)"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 writing require.\n", m_serial);

    //阻塞互斥对象mutex3,保证对writecount的访问、修改互斥
    wait_for_mutex3 = WaitForSingleObject(h_mutex3, -1);//P操作
                                                              //修改写者数目
    writecount++;
    if (writecount == 1)
    {
        //第一个写者，等待读者读完
        EnterCriticalSection(&cs_Read);
    }
    ReleaseMutex(h_mutex3);//V操作

                              //进入写者临界区
    EnterCriticalSection(&cs_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);
    //离开临界区
    LeaveCriticalSection(&cs_Write);

    //阻塞互斥对象mutex3,保证对writecount的访问、修改互斥
    wait_for_mutex3 = WaitForSingleObject(h_mutex3, -1);//P操作
    writecount--;
    if (writecount == 0)
    {
        //写者写完，读者可以读
        LeaveCriticalSection(&cs_Read);
    }
    ReleaseMutex(h_mutex3);//V操作
}

///////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
//写者优先处理函数
//file:文件名

void WriterPriority(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_Mutex1;
    h_Mutex1 = CreateMutex(NULL, FALSE, (LPCWSTR)"mutex1");
    HANDLE h_Mutex2;
    h_Mutex2 = CreateMutex(NULL, FALSE, (LPCWSTR)"mutex2");
    HANDLE h_Mutex3;
    h_Mutex3 = CreateMutex(NULL, FALSE, (LPCWSTR)"mutex3");

    //线程对象
    HANDLE h_Thread[MAX_THREAD_NUM];
    ThreadInfo thread_info[MAX_THREAD_NUM];
    readcount = 0;//初始化readcount
    writecount = 0;//初始化writecount
    InitializeCriticalSection(&cs_Write);//初始化临界区
    InitializeCriticalSection(&cs_Read);
    ifstream inFile;
    inFile.open(file);//打开文件
    printf("Writer 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)(WP_ReaderThread),
                &thread_info[i],
                0, &thread_ID);
        }
        else {
            //创建写者进程
            h_Thread[i] = CreateThread(NULL, 0,
                (LPTHREAD_START_ROUTINE)(WP_WriterThread),
                &thread_info[i],
                0, &thread_ID);
        }
    }
    //等待所有线程结束
    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 or 3):");
    //如果信息不正确，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("thread.dat");
    //选择2，写者优先
    else
        WriterPriority("thread.dat");
    //结束
    printf("\nPress Any Key To Continue:");
    _getch();
    system("cls");
}
return 0;
   }
```