操作系统实验一

读者-写者问题

目录

2017-12-10

[裴子祥 计科七班 学号 2015211921]

[指导老师：孟祥武]

一．实验内容与要求

题目：读者优先和写者优先的读者-写者问题实践解决。

实验内容：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的要求，进行读写操作。

读者-写者问题的读写操作限制：

1. 写-写互斥。
2. 读-写允许。
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。

实验环境：

MICSOFT WINDOW 10

Visual Studio 2015

二． 程序设计与实现

1． 读者优先

过程说明：在读者优先的情况下：除非有写者在写文件，否则没有一个读者需要等待。

新读者到：

- 无读者、写者，新读者可以读
- 有写者等，但有其它读者正在读，则新读者也可以读
- 有写者写，新读者等

新写者到：

- 无读者、写者，新写者可以写
- 有读者读，新写者等待
- 有其它写者，新写者等待

信号量：

```
int readCount = 0;            //当前读者数目
Semaphore mutex = 1;          //互斥信息，用于 readCount 的互斥修改
CRITICAL_SECTION CommonArea;//临界区域，初始时是空闲状态
```

读者：

```
P(mutex);           //占用互斥量
readCount++;         //读者数+1
if(readCount==1)
{//如果是第一个读者
    //则等待写者释放临界资源
    等待临界资源释放;
```

```
}

V(mutex);           //互斥量释放

开始读操作;

//读完后，读者离开
P(mutex);           //占用互斥量
readCount--;        //读者数减一

if(readCount==0)
{//如果没有读者在读了
    释放临界资源;
}
V(mutex);           //互斥量释放
```

写者：

```
等待临界资源释放;
开始写操作;
释放临界资源;
```

2． 写者优先

过程说明：一旦一个写者到来，它应该尽快对文件进行写操作。则新来到的读者不允许进行读操作。

新读者到：

- 无读者、写者，新读者可以读
- 有读者读，无写者等，新读者可以读
- 有写者等，但有写者等待，新读者等待
- 有写者写，新读者等

新写者到：

- 无读者、写者，新写者可以写
- 有读者读，新写者等待
- 有写者写，新写者等待

信号量：

```
int readCount = 0;          //当前读者数目
int writeCount = 0;         //当前写者数目
CRITICAL_SECTION ReadArea;  //读者临界区域，写者也能占用
CRITICAL_SECTION WriteArea; //写者临界区域，读者不能
Semaphore mutex1 = 1;        //互斥信息，用于读者临界区的状态互斥修改
Semaphore mutex2 = 1;       //互斥信息，用于 readCount 的互斥修改
Semaphore mutex3 = 1;       //互斥信息，用于 writeCount 的互斥修改
```

读者：

```
P(mutex1);          //占用互斥量，以修改读者临界区状态
等待 ReadArea 释放，然后占用它;
P(mutex2);          //占用互斥量，以 readCount
readCount++;        //读者数+1
if(readCount==1)
{//如果是第一个读者
    //等待写者写完
    等待 WriteArea 释放，然后占用它;
}
V(mutex2);          //释放 readCount 修改互斥量
释放 ReadArea;
V(mutex1);          //释放 ReadArea 状态修改互斥量

开始读操作;

P(mutex2);          //占用互斥量，以修改 readCount
readCount--;
if(readCount==0)
{//如果没有读者在读了
    释放 WriteArea，以唤醒读者;
}
V(mutex2);
```

写者：

```
P(mutex3);          //占用互斥量，以修改 writeCount
writeCount++;       //写者数+1
if(writeCount==1)
{//如果是第一个写者
    //等第一个读者读完
    等待 ReadArea 释放，然后占用它;
}
V(mutex3);

进入写者临界区;
开始写操作;

P(mutex3);
writeCount--;

if(writeCount==0)
{
    //所有写者写完，读者可以读
    释放 ReadArea;
}
```

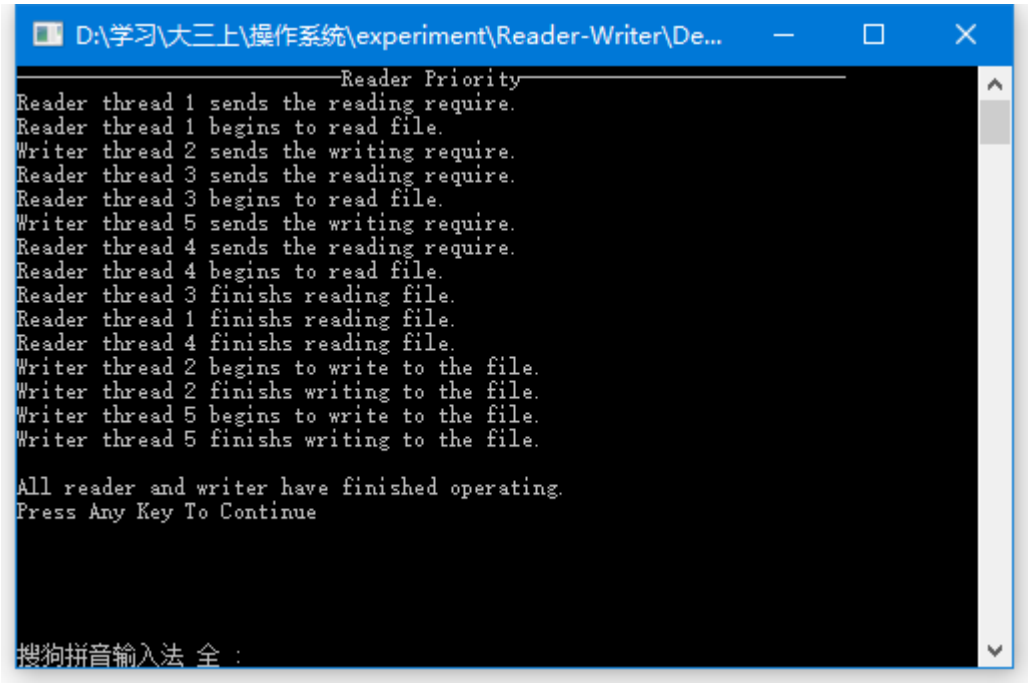
```
    V(mutex3);
```

3． 运行结果

测试用例 1eg：

线程序号	（读者 OR 写者）	开始时间	持续时间
1	R	3	5
2	W	4	5
3	R	5	2
4	R	6	5
5	W	5.1	3

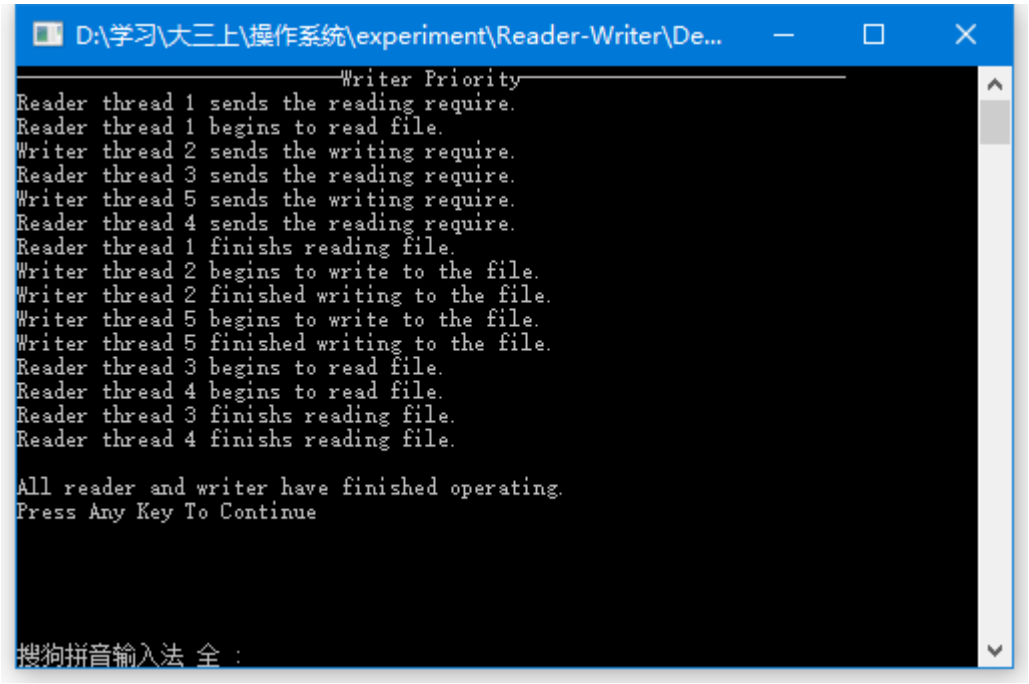
读者优先：




```
Reader thread 1 sends the reading require.
Reader thread 1 begins to read file.
Writer thread 2 sends the writing require.
Reader thread 3 sends the reading require.
Reader thread 3 begins to read file.
Writer thread 5 sends the writing require.
Reader thread 4 sends the reading require.
Reader thread 4 begins to read file.
Reader thread 3 finishs reading file.
Reader thread 1 finishs reading file.
Reader thread 4 finishs reading file.
Writer thread 2 begins to write to the file.
Writer thread 2 finishs writing to the file.
Writer thread 5 begins to write to the file.
Writer thread 5 finishs writing to the file.
```

```
All reader and writer have finished operating.
```

写者优先：



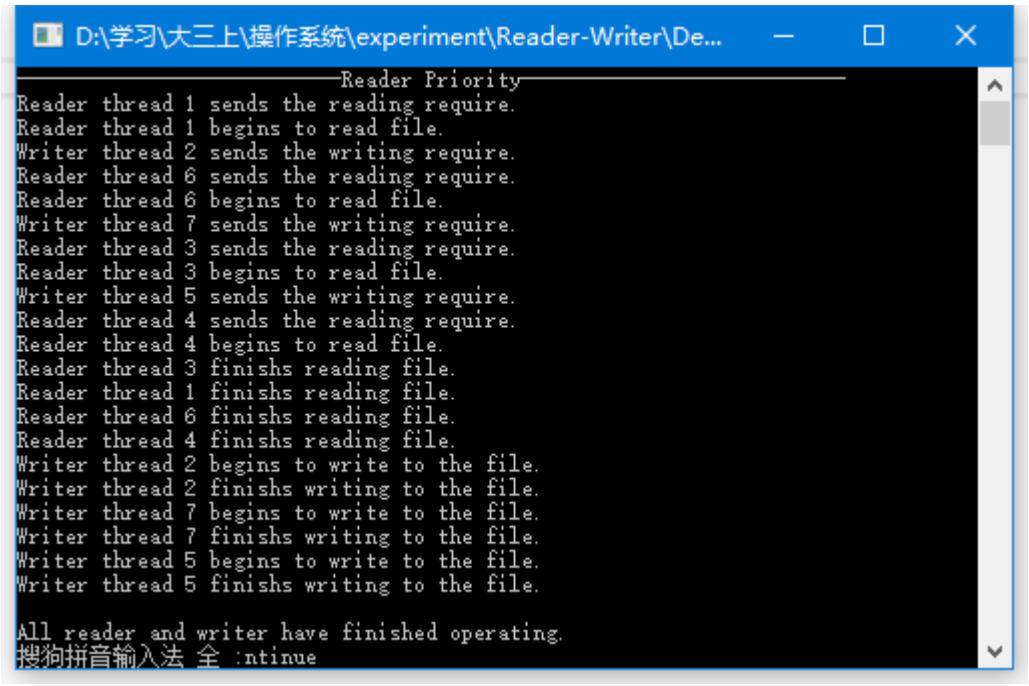

```
Reader thread 1 sends the reading require.
Reader thread 1 begins to read file.
Writer thread 2 sends the writing require.
Reader thread 3 sends the reading require.
Writer thread 5 sends the writing require.
Reader thread 4 sends the reading require.
Reader thread 1 finishs reading file.
Writer 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 finishs reading file.
Reader thread 4 finishs reading file.

All reader and writer have finished operating.
```

测试用例 2eg：

线程序号	（读者 OR 写者）	开始时间	持续时间

1	R	1	6
2	W	3	5
3	R	5	2
4	R	6	5
5	W	5	3
6	R	3	4
7	W	4	2

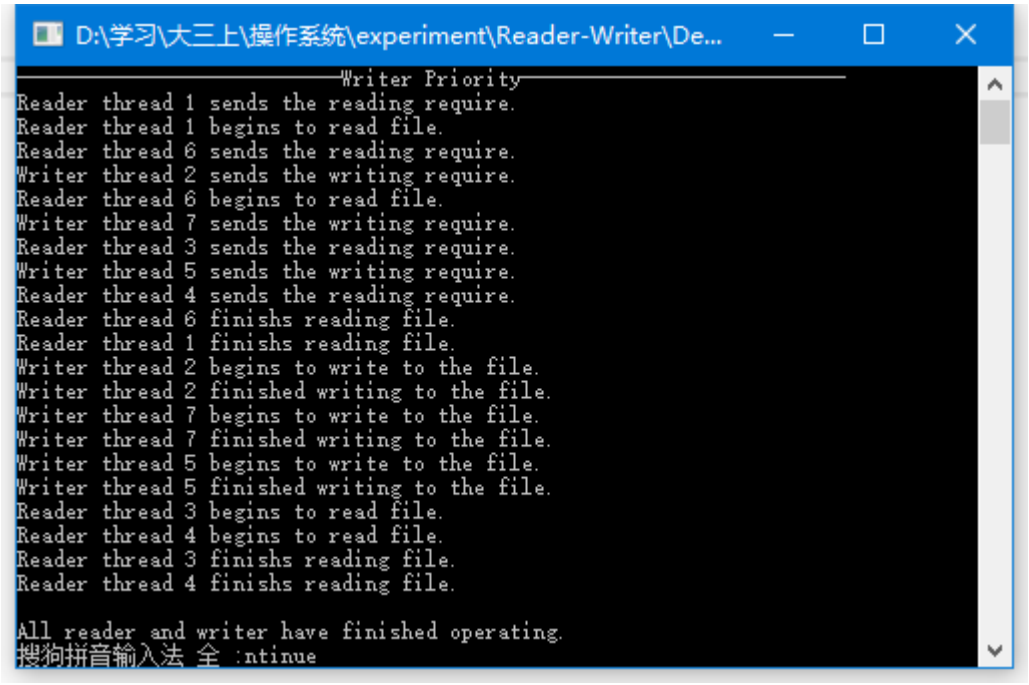
读者优先：




```
Reader thread 1 sends the reading require.
Reader thread 1 begins to read file.
Writer thread 2 sends the writing require.
Reader thread 6 sends the reading require.
Reader thread 6 begins to read file.
Writer thread 7 sends the writing require.
Reader thread 3 sends the reading require.
Reader thread 3 begins to read file.
Writer thread 5 sends the writing require.
Reader thread 4 sends the reading require.
Reader thread 4 begins to read file.
Reader thread 3 finishs reading file.
Reader thread 1 finishs reading file.
Reader thread 6 finishs reading file.
Reader thread 4 finishs reading file.
Writer thread 2 begins to write to the file.
Writer thread 2 finishs writing to the file.
Writer thread 7 begins to write to the file.
Writer thread 7 finishs writing to the file.
```

```
Writer thread 5 begins to write to the file.
Writer thread 5 finishs writing to the file.

All reader and writer have finished operating.
```

写者优先：




```
Reader thread 1 sends the reading require.
Reader thread 1 begins to read file.
Reader thread 6 sends the reading require.
Writer thread 2 sends the writing require.
Reader thread 6 begins to read file.
Writer thread 7 sends the writing require.
Reader thread 3 sends the reading require.
Writer thread 5 sends the writing require.
Reader thread 4 sends the reading require.
Reader thread 6 finishs reading file.
Reader thread 1 finishs reading file.
Writer thread 2 begins to write to the file.
Writer thread 2 finished writing to the file.
Writer thread 7 begins to write to the file.
Writer thread 7 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 finishs reading file.
Reader thread 4 finishs reading file.
```

```
All reader and writer have finished operating.
```

4． 源程序附件

```
#include <Windows.h>
#include <conio.h>
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <cstdio>
#include <ctime>
#include <string>

using namespace std;

#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
#define MAX_FILE_NUM 32          //最大数据文件数目
#define MAX_STR_LEN 32           //字符串长度

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;

typedef struct threadInfo
{
    int serial;                 //线程序号
    char type;                  //线程类别(R 为读者，W 为写者
    double delay;               //线程创建后，延时时间
    double persist;             //线程持续时间
}ThreadInfo;

/************************
 *  读者优先-读者进程  *
 *infoPtr:读者线程信息 *
 ************************/
void ReadPriority_RThread(ThreadInfo *infoPtr)
{
```

```
    HANDLE h_Mutex;            //互斥变量
    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

    //从参数中获取信息
    m_serial = infoPtr->serial;
    m_delay = infoPtr->delay*INTE_PER_SEC;
    m_persist = infoPtr->persist*INTE_PER_SEC;
    Sleep(m_delay);            //暂时挂起，延迟时间的时长

    printf("Reader thread %d sends 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);      //P 操作
                                                            //读者数目增加
    readCount++;


    if (readCount == 1)
    {
        //如果是第一个读者，等待资源被写者写完
        EnterCriticalSection(&RP_Write);
    }

    ReleaseMutex(h_Mutex);      //V 操作

    printf("Reader thread %d begins to read file.\n", m_serial);
    Sleep(m_persist);          //持续时间

    //退出线程
    printf("Reader thread %d finishs reading file.\n", m_serial);

    //等待互斥信号，保证对 reaadCount 的访问、修改互斥
    wait_for_mutex = WaitForSingleObject(h_Mutex, -1);
    //读者书减少
    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-写者进程  *
*infoPtr:写者线程信息 *
************************/
void ReadPriority_WThread(ThreadInfo *infoPtr)
{
    DWORD m_delay;              //延迟时间
    DWORD m_persist;            //持续时间
    int m_serial;               //线程序号

    //从参数中获取信息
    m_serial = infoPtr->serial;
    m_delay = infoPtr->delay*INTE_PER_SEC;
    m_persist = infoPtr->persist*INTE_PER_SEC;
    Sleep(m_delay);             //延迟等待

    printf("Writer thread %d sends the writing require.\n",m_serial);
    //等待临界资源
    EnterCriticalSection(&RP_Write);

    //开始写文件
    printf("Writer thread %d begins to write to the file.\n",m_serial);
    Sleep(m_persist);

    //退出程序
    printf("Writer thread %d finishs writing to the file.\n", m_serial);

    //释放临界资源
    LeaveCriticalSection(&RP_Write);
}

/*******************************
*  读者优先函数                *
*  fileName:文件名(初始化文件)  *
*******************************/
void ReaderPriority(char *fileName)
{
```

```
    DWORD n_thread = 0;            //线程数目
    DWORD serial_thread;           //线程序号
    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);      //初始化临界资源

    ifstream inFile;

    inFile.open(fileName);
    if (!inFile.is_open())
    {
        cout << "Failed to open the " << fileName << endl;
        return;
    }

    cout << "--------------------------Reader Priority--------------------------" <<
endl;

    while (!inFile.eof())
    {
        //读入读者或写者初始信息
        inFile >> thread_info[n_thread].serial;
        inFile >> thread_info[n_thread].type;
        inFile >> thread_info[n_thread].delay;
        inFile >> thread_info[n_thread].persist;

        //下一个读者或写者
        n_thread++;

        //读取回车换行符
        inFile.get();
    }

    for (int i = 0; i < (int)n_thread; ++i)
    {
        if (thread_info[i].type == READER || thread_info[i].type == 'r')
```

```
        {
            //创建读者进程
            h_Thread[i] = CreateThread(NULL, 0,
                (LPTHREAD_START_ROUTINE)ReadPriority_RThread,
                &thread_info[i],
                0, &serial_thread);
        }
        else
        {
            //创建写者进程
            h_Thread[i] = CreateThread(NULL, 0,
                (LPTHREAD_START_ROUTINE)ReadPriority_WThread,
                &thread_info[i],
                0, &serial_thread);
        }
    }

    //等待所有线程结束
    wait_for_all = WaitForMultipleObjects(n_thread, h_Thread, true, -1);
    cout << endl << "All reader and writer have finished operating." << endl;
}


/************************
*  写者优先-读者线程  *
*infoPtr:读者线程信息 *
************************/
void WritePriority_RThread(ThreadInfo *infoPtr)
{
    //互斥变量
    HANDLE h_mutex1;                    //用于访问 cs_Read 临界区
    h_mutex1 = OpenMutex(MUTEX_ALL_ACCESS, false, "mutex1");
    HANDLE h_mutex2;                    //用于 readCount
    h_mutex2 = OpenMutex(MUTANT_ALL_ACCESS, false, "mutex2");

    DWORD wait_for_mutex1;               //等待互斥变量所有权
    DWORD wait_for_mutex2;               //用于 readCount

    DWORD m_delay;                      //延迟时间
    DWORD m_persist;                    //读操作持续时间
    int m_serial;                       //线程序号

    //从参数中获取线程信息
    m_serial = infoPtr->serial;
    m_delay = infoPtr->delay*INTE_PER_SEC;
```

```
    m_persist = infoPtr->persist*INTE_PER_SEC;

    Sleep(m_delay);                    //延迟等待

    printf("Reader thread %d sends 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，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

    LeaveCriticalSection(&cs_Read);     //让其它读者进入临界区

    ReleaseMutex(h_mutex1);            //V 操作

    //开始读操作
    printf("Reader thread %d begins to read file.\n", m_serial);
    Sleep(m_persist);                  //读持续时间

    //退出线程
    printf("Reader thread %d finishs 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  *
*infoPtr:写者线程信息 *
***********************/
void WritePriority_WThread(ThreadInfo* infoPtr)
{
    DWORD wait_for_mutex3;            //用于 writeCount
    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, "mutex3");

    //从参数中获得信息
    m_serial = infoPtr->serial;
    m_delay = infoPtr->delay*INTE_PER_SEC;
    m_persist = infoPtr->persist *INTE_PER_SEC;

    Sleep(m_delay);//延迟等待

    printf("Writer thread %d sends 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);

    //开始写操作
    printf("Writer thread %d begins to write to the file.\n", m_serial);
    Sleep(m_persist);             //写操作持续时间

    //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
}


/*******************************
*  写者优先函数                *
*  fileName:文件名(初始化文件)  *
*******************************/
void WriterPriority(char *fileName)
{
    DWORD n_thread = 0;                    //线程数目
    DWORD serial_thread;                       //线程 ID
    DWORD wait_for_all;                    //等待所有线程结束

    //创建互斥对象
    HANDLE h_Mutex1;                       //用于 cs_Read 临界区状态修改
    h_Mutex1 = CreateMutex(NULL, FALSE, "mutex1");
    HANDLE h_Mutex2;                       //用于 readCount 互斥修改
    h_Mutex2 = CreateMutex(NULL, FALSE, "mutex2");
    HANDLE h_Mutex3;                       //用于 writeCount 互斥修改
    h_Mutex3 = CreateMutex(NULL, FALSE, "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(fileName);

    if (!inFile.is_open())
    {
        cout << "Failed to open the " << fileName << endl;
        return;
    }

    cout << "--------------------------Writer Priority--------------------------" <<
endl;

    while (!inFile.eof())
    {
        //读入读者或写者初始信息
        inFile >> thread_info[n_thread].serial;
        inFile >> thread_info[n_thread].type;
        inFile >> thread_info[n_thread].delay;
        inFile >> thread_info[n_thread].persist;

        //下一个读者或写者
        n_thread++;

        //读取回车换行符
        inFile.get();
    }

    for (int i = 0; i < (int)n_thread; ++i)
    {
        if (thread_info[i].type == READER || thread_info[i].type == 'r')
        {
            //创建读者进程
            h_Thread[i] = CreateThread(NULL, 0,
                (LPTHREAD_START_ROUTINE)WritePriority_RThread,
                &thread_info[i],
                0, &serial_thread);
        }
        else
        {
            //创建写者进程
            h_Thread[i] = CreateThread(NULL, 0,
                (LPTHREAD_START_ROUTINE)WritePriority_WThread,
                &thread_info[i],
                0, &serial_thread);
        }
    }
```

```
    //等待所有线程结束
    wait_for_all = WaitForMultipleObjects(n_thread, h_Thread, true, -1);
    cout << endl << "All reader and writer have finished operating." << endl;

}



int main()
{
    char ch;

    while (true)
    {
        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 ):");

        do
        {
            scanf_s("%c", &ch);
            getchar();
        } while (ch != '1' && ch != '2' && ch != '3');

        system("cls");

        if (ch == '1')
        {
            //读者优先
            ReaderPriority("Reader_Writer.txt");
        }
        else if (ch == '2')
        {
            //写者优先
            WriterPriority("Reader_Writer.txt");
        }
        else
        {
            cout << "GOOD BYE" << endl;
            system("pause");
            return 0;             //退出
```

```
        }

        //结束
        cout << "Press Any Key To Continue";
        getchar();
        system("cls");
    }

    system("pause");
    return 0;
}
```