北京邮电大学　计算机学院

《操作系统概念》实验报告

姓名　王睿嘉

学号 *2015211906*

班级 *2015211307*

实验 进程同步

一、 实验要求和环境描述

1. 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个线程分别表示n个读者或写者。每个线程按测试数据文件的要求，进行读写操作。并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：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。

2. 环境描述

编程语言：C++

集成开发环境：VS2015

二、 输入输出定义

1. 输入定义

用户需输入序号进行策略选择，即：

1， 读者优先

2， 写者优先

3， 退出程序

输入其他字符，程序将不予理会，直到正确序号输入。

2. 输出定义

由入口函数 main 打印出菜单。用户选择策略后，在每个进程创建、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提示信息。

三、 实验设计

1. 整体说明

所谓读者-写者问题，可以这样描述：有一群写者和一群读者，写者在写同一本书，读者也在读这本书。多个读者可以同时读这本书，但是，只能有一个写者在写书。并且根据策略不同，有不同的优先顺序。若采取读者优先策略，读者和写者同时提出请求时，读者优先；若采取写者优先策略，读者和写者同时提出请求时，写者优先。其具体定义如下：

有一个许多进程共享的数据区，有一些只读取这个数据区的进程和一些只往数据区写数据的进程。任意多个读进程可以同时读这个数据区；一次只有一个写进程可以往数据区中写；如果一个写进程正在进行操作，禁止任何读进程读取数据区。

信号量是支持多道程序的并发操作系统中解决资源共享进程间同步和互斥的重要机制，而读者-写者问题则是这一机制的经典范例。

与记录性信号量解决读者-写者问题不同，本实验的信号量机制增加了一个限制，即最多允许 MAX_THREAD_NUM 个读、写者同时对文件进行操作。为此，线程对象数组 h_Thread 和线程信息数组 thread_info 在声明时，大小均为 MAX_THREAD_NUM。

运行过程中，相当于将所有读者和写者分别放入两个等待队列。当读允许时，就让读者队列释放一个或多个读者；当写允许时，就让写者队列释放第一个写者。

程序由入口函数 main 开始，打印出菜单。用户若输入 1，选择读者优先策略，调用 ReaderPriority("thread.dat")函数；若输入 2，选择写者优先策略，调用 WriterPriority("thread.dat")函数；若输入 3，退出；否则程序仍在循环中，直到正确序号输入。

读者优先策略：

ReaderPriority 函数首先读取目标文件 thread.dat。为文件中的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。

写者优先策略：

WriterPriority 函数首先读取目标文件 thread.dat，为文件中的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。

所有线程创建完毕后，均调用 WaitForMultipleObjects 函数等待所有线程结束，并输出相应提示信息。待用户输入任意字符后，调用 system("cls")，清空命令行界面，返回策略选择。

本实验使用的测试数据文件格式为：

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。第二字段表示线程角色，R 是读者，W 是写者。第三字段为一个正数，表示读写操作的开始时间，即线程创建后，延时相应时间（单位为秒）后，发出对共享资源的读写申请。第四字段为一个正数，表示读写操作的持续时间。当线程读写申请成功后，开始对共享资源的操作，该操作持续相应时间后结束，并释放共享资源。例如：

1 R 3 5

2 W 4 5

2. 关键点说明

2.1 RP_ReaderThread 函数

伪代码如下：

```
P(mutex);               //等待互斥信号，从而保证对 read_count 的访问和修改是原子的
read_count++;           //读者数目增加
if(read_count==1)       //若是第一个读者
    P(&RP_Write);       //若当前有写者正在进行写操作，等待其完成后，禁止写操作；否则，直接禁止写操作，从而保证读写互斥
V(mutex);               //释放对 read_count 的占用，使之可被访问和修改
读临界区……
P(mutex);               //等待互斥信号，从而保证对 read_count 的访问和修改是原子的
read_count--;           //读者数目减少
if(read_count==0)       //若读者全部读完
    V(&RP_Write);       //唤醒写者线程，允许写操作进行
V(mutex);               //释放对 read_count 的占用，使之可被访问和修改
```

2.2 RP_WriterThread 函数

伪代码如下：

```
P(&RP_Write);           //等待互斥信号，从而保证对临界区的写操作是互斥的
写临界区……
V(&RP_Write)            //释放对临界区的占用，使之可被访问和修改
```

2.3 WP_ReaderThread 函数

伪代码如下：

```
P(mutex1);              //等待互斥信号，从保证对 cs_Read 的访问和修改是互斥的
P(&cs_Read);            //进入读者临界区
P(mutex2);              //等待互斥信号，从而保证对 read_count 的访问和修改是原子的
read_count++;           //读者数目增加
if(read_count==1)       //若是第一个读者
    P(&cs_Write);       //若当前有写者正在进行写操作，等待全部写者完成后，禁止写操作；否则，直接禁止写操作，从而保证读写互斥
V(mutex2);              //释放对 read_count 的占用，使之可被访问和修改
```

```
V(&cs_Read);            //释放对读者临界区的占用，使之可被访问和修改
V(mutex1);              //释放对 read_count 的占用，使之可被访问和修改
读临界区……
P(mutex2);              //等待互斥信号，从而保证对 read_count 的访问和修改是原子的
read_count--;           //读者数目减少
if(read_count==0)       //若读者全部读完
    V(&cs_Write);       //唤醒写者线程，允许写操作进行
V(mutex2);              //释放对 read_count 的占用，使之可被访问和修改
```

2.4 WP_WriterThread 函数

伪代码如下：

```
P(mutex3);              //等待互斥信号，从而保证对 write_count 的访问和修改是原子的
write_count++;          //写者数目增加
if(write_count==1)      //若是第一个写者
    P(&cs_Read);        //若当前有读者正在进行读操作，等待其完成后，禁止读操作；否则，直接禁止读操作，从而保证读写互斥
V(mutex3);              //释放对 write_count 的占用，使之可被访问和修改
P(&cs_Write);           //等待互斥信号，从而保证对临界区的写操作是互斥的
写临界区……
V(&cs_Write);           //释放对临界区的占用，使之可被访问和修改
P(mutex3);              //等待互斥信号，从而保证对 write_count 的访问和修改是原子的
write_count--;          //写者数目减少
if(write_count==0)      //若写者全部写完
    V(&cs_Read);        //唤醒读者线程，允许读操作进行
V(mutex3);              //释放对 write_count 的占用，使之可被访问和修改
```

四、 文件说明

共有一个源码文件：

Reader-Writers Problem.cpp：读者-写者问题的具体实现。

共有一个数据文件：

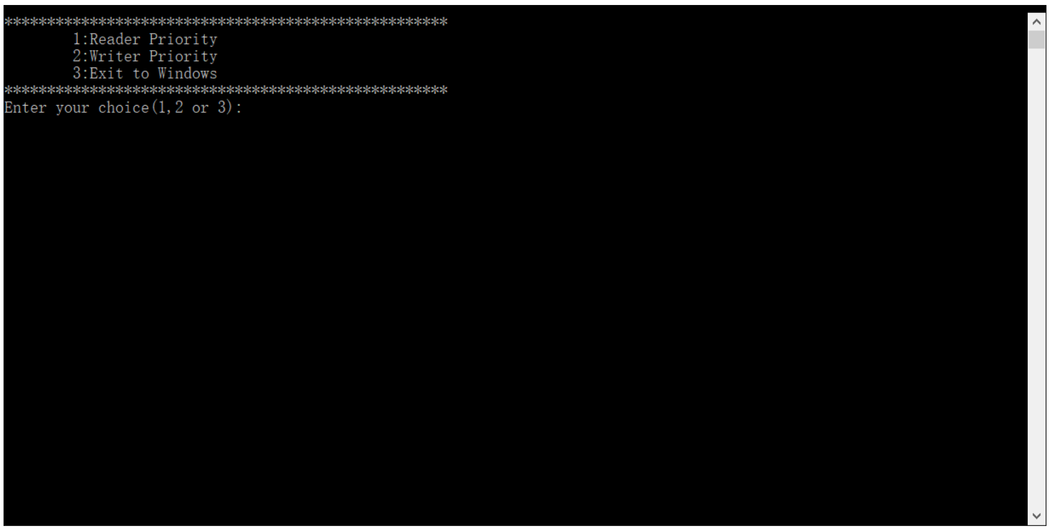
thread.dat：测试数据文件。

五、 实验结果

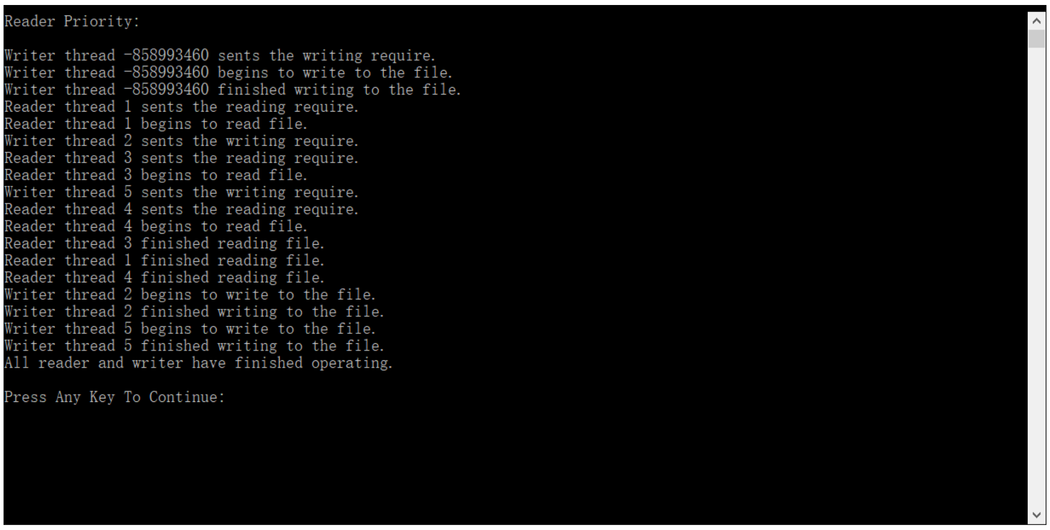
测试数据文件为：

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

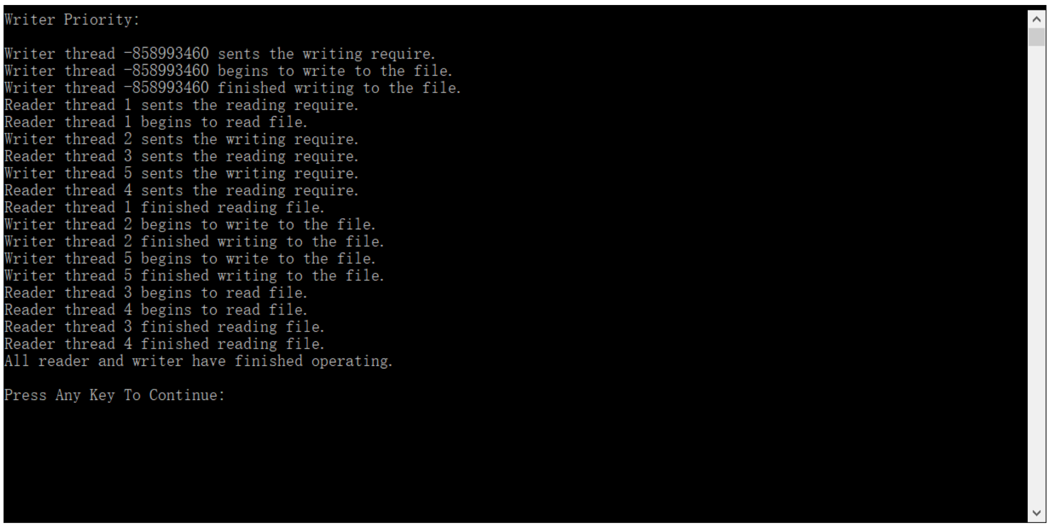
运行结果如下：



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



```
Reader Priority:

Writer thread -858993460 sents the writing require.
Writer thread -858993460 begins to write to the file.
Writer thread -858993460 finished writing to the file.
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:
```

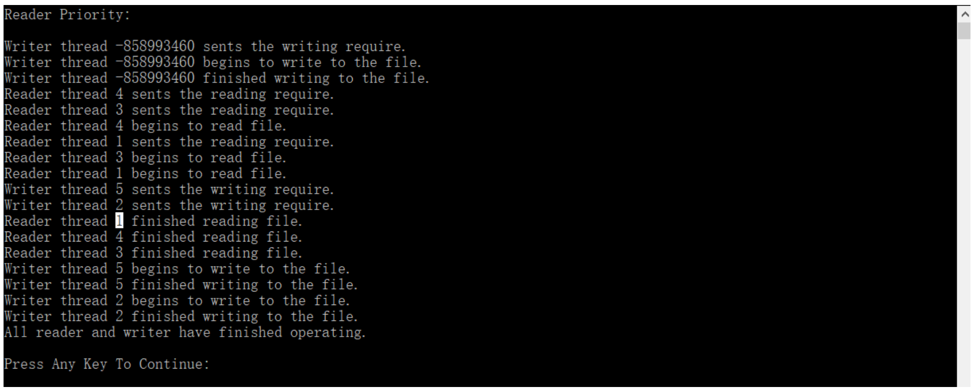


```
Writer Priority:

Writer thread -858993460 sents the writing require.
Writer thread -858993460 begins to write to the file.
Writer thread -858993460 finished writing to the file.
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:
```

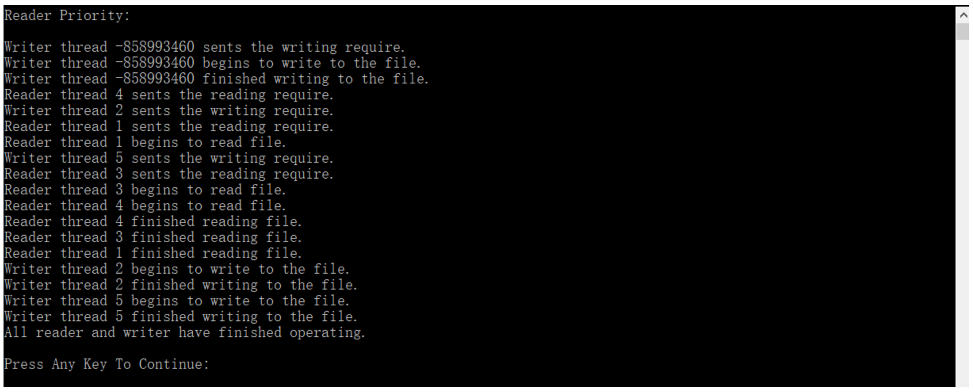
其执行顺序与手动计算结果一致，确认程序运行无误。修改测试数据文件为：

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

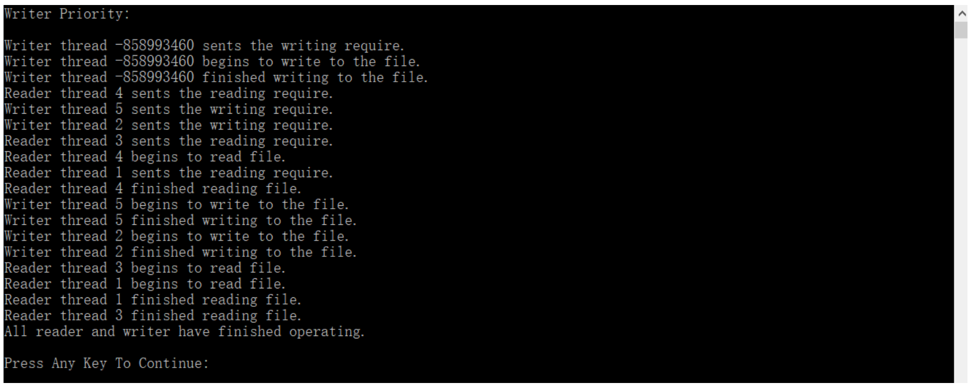
运行结果如下：



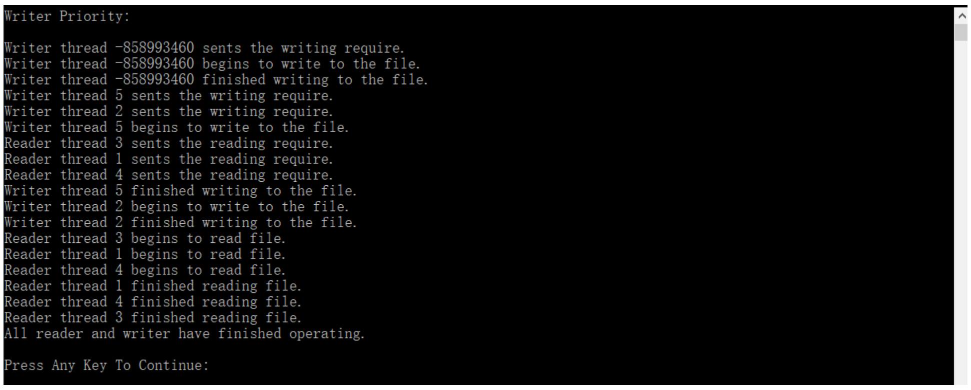
```
Reader Priority:

Writer thread -858993460 sents the writing require.
Writer thread -858993460 begins to write to the file.
Writer thread -858993460 finished writing to the file.
Reader thread 4 sents the reading require.
Reader thread 3 sents the reading require.
Reader thread 4 begins to read file.
Reader thread 1 sents the reading require.
Reader thread 3 begins to read file.
Reader thread 1 begins to read file.
Writer thread 5 sents the writing require.
Writer thread 2 sents the writing require.
Reader thread 1 finished reading file.
Reader thread 4 finished reading file.
Reader thread 3 finished reading file.
Writer thread 5 begins to write to the file.
Writer thread 5 finished writing to the file.
Writer thread 2 begins to write to the file.
Writer thread 2 finished writing to the file.
All reader and writer have finished operating.

Press Any Key To Continue:
```



```
Reader Priority:

Writer thread -858993460 sents the writing require.
Writer thread -858993460 begins to write to the file.
Writer thread -858993460 finished writing to the file.
Reader thread 4 sents the reading require.
Writer thread 2 sents the writing require.
Reader thread 1 sents the reading require.
Reader thread 1 begins to read file.
Writer thread 5 sents the writing require.
Reader thread 3 sents the reading require.
Reader thread 3 begins to read file.
Reader thread 4 begins to read file.
Reader thread 4 finished reading file.
Reader thread 3 finished reading file.
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.
All reader and writer have finished operating.

Press Any Key To Continue:
```



```
Writer Priority:

Writer thread -858993460 sents the writing require.
Writer thread -858993460 begins to write to the file.
Writer thread -858993460 finished writing to the file.
Reader thread 4 sents the reading require.
Writer thread 5 sents the writing require.
Writer thread 2 sents the writing require.
Reader thread 3 sents the reading require.
Reader thread 4 begins to read file.
Reader thread 1 sents the reading require.
Reader thread 4 finished reading file.
Writer thread 5 begins to write to the file.
Writer thread 5 finished writing to the file.
Writer thread 2 begins to write to the file.
Writer thread 2 finished writing to the file.
Reader thread 3 begins to read file.
Reader thread 1 begins to read file.
Reader thread 1 finished reading file.
Reader thread 3 finished reading file.
All reader and writer have finished operating.

Press Any Key To Continue:
```



```
Writer Priority:

Writer thread -858993460 sents the writing require.
Writer thread -858993460 begins to write to the file.
Writer thread -858993460 finished writing to the file.
Writer thread 5 sents the writing require.
Writer thread 2 sents the writing require.
Writer thread 5 begins to write to the file.
Reader thread 3 sents the reading require.
Reader thread 1 sents the reading require.
Reader thread 4 sents the reading require.
Writer thread 5 finished writing to the file.
Writer thread 2 begins to write to the file.
Writer thread 2 finished writing to the file.
Reader thread 3 begins to read file.
Reader thread 1 begins to read file.
Reader thread 4 begins to read file.
Reader thread 1 finished reading file.
Reader thread 4 finished reading file.
Reader thread 3 finished reading file.
All reader and writer have finished operating.

Press Any Key To Continue:
```

发现了一个有趣的现象。由于开始读写操作的时间是相同的，根据不同进程的创建速度，进程相应操作的申请顺序是随机的，因而，在满足各自优先策略的基础上，执行顺序也是随机的。

六、 实验心得

在本次实验中，主要感悟有以下三点：

a) 读写线程间的相互制约。课堂所讲的读写者互斥是通过信号量实现，其数据结构为 int。而实际代码中通过 EnterCriticalSection 函数进入相应临界区，从而保证互斥，其数据结构为 CRITICAL_SECTION。起初，因为这个差别对整个参考代码都存在一些困惑。经反复思考后，豁然开朗。读者优先策略中，第一位读者进入写者临界区，相当于将读者队列以一个写者的身份排入了写者队列。从而保证了若有写操作正在进行，操作结束后，不会再有写者进入数据区。因为，此时进入数据区的“写者”是穿着“写者外衣”的读者线程队列。而写者优先策略，同理；

b) API。在以前的编程中，只使用过简单的 CreateThread 和 ExitThread 函数。通过此次实验，了解到 C++中有许多现成的 API，如互斥信号量等，以供进程同步，十分方便；

c) 参考代码存在错误。在未放入工程前，发现参考代码在写者优先策略中，CreateThread 函数误用了 RP_ReaderThread。放入工程后，又出现了与 IDE 不兼容的头文件、函数参数等，一一修改。

实践出真知，本次进程同步实验是对课堂和书本所学知识的补充。读者-写者问题的实际情形与已了解到的原理大体一致，但又复杂许多。通过反复研究代码，加深了对进程同步的理解和记忆，收获颇丰。