1. **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；

1. **实验内容**

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（见“三、测试数据”）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。

1. **测试数据**

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

1. **实验基础知识**

**(1)问题分析**

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。  
 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。

读者优先算法

***int*** count**=**0;  //用于记录当前的读者数量

semaphore mutex**=**1;  //用于保护更新count变量时的互斥

semaphore rw**=**1;  //用于保证读者和写者互斥地访问文件

**writer** () {  //写者进程

**while** (1){

**P**(rw); // 互斥访问共享文件

        Writing;  //写入

**V**(rw) ;  //释放共享文件

    }

}

**reader** () {  // 读者进程

**while**(1){

**P** (mutex) ;  //互斥访问count变量

**if** (count**==**0)  //当第一个读进程读共享文件时

**P**(rw);  //阻止写进程写

        count**++**;  //读者计数器加1

**V** (mutex) ;  //释放互斥变量count

        reading;  //读取

**P** (mutex) ;  //互斥访问count变量

        count**--**; //读者计数器减1

**if** (count**==**0)  //当最后一个读进程读完共享文件

**V**(rw) ;  //允许写进程写

**V** (mutex) ;  //释放互斥变量 count

    }

}

在上面的算法中，读进程是优先的，也就是说，当存在读进程时，写操作将被延迟，并且只要有一个读进程活跃，随后而来的读进程都将被允许访问文件。这样的方式下，会导致写进程可能长时间等待，且存在写进程“饿死”的情况。  
 如果希望写进程优先，即当有读进程正在读共享文件时，有写进程请求访问，这时应禁止后续读进程的请求，等待到已在共享文件的读进程执行完毕则立即让写进程执行，只有在无写进程执行的情况下才允许读进程再次运行。为此，增加一个信号量并且在上面的程序中 writer()和reader()函数中各增加一对PV操作，就可以得到写进程优先的解决程序。

**写者优先**

***int*** count **=** 0;  //用于记录当前的读者数量

semaphore mutex **=** 1;  //用于保护更新count变量时的互斥

semaphore rw**=**1;  //用于保证读者和写者互斥地访问文件

semaphore w**=**1;  //用于实现“写优先”

**writer**(){

**while**(1){

**P**(w);  //在无写进程请求时进入

**P**(rw);  //互斥访问共享文件

        writing;  //写入

**V**(rw);  // 释放共享文件

**V**(w) ;  //恢复对共享支件的访问

    }

}

**reader** () {  //读者进程

**while** (1){

**P** (w) ;  // 在无写进程请求时进入

**P** (mutex);  // 互斥访问count变量

**if** (count**==**0)  //当第一个读进程读共享文件时

**P**(rw);  //阻止写进程写

        count**++**;  //读者计数器加1

**V** (mutex) ;  //释放互斥变量count

**V**(w);  //恢复对共享文件的访问

        reading;  //读取

**P** (mutex) ; //互斥访问count变量

        count**--**;  //读者计数器减1

**if** (count**==**0)  //当最后一个读进程读完共享文件

**V**(rw);  //允许写进程写

**V** (mutex);  //释放互斥变量count

    }

}

1. **实验设计方法**

根据输入的数据，我们切换到读者优先和写者优先的处理程序中。

对于线程数据，采用了一个自定义类thread\_info来进行处理，然后利用类的构造函数来进行进程数据的读取

***class*** *thread\_info* {

***public:***

***int*** id;           //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*** lastTime;     //线程读写操作时间

***void*** **show\_all\_things**() {

        cout **<<** "thread id ->" **<<** id **<<** " entity -> " **<<** entity **<<** " delay ->" **<<** delay **<<**" lastTime -> " **<<** lastTime **<<** **endl**;

    }

**thread\_info**(*ifstream* & **inFile**) {

**inFile** **>>** id;

**inFile** **>>** entity;

**inFile** **>>** delay;

**inFile** **>>** lastTime;

**show\_all\_things**();

    }

};

读取到进程的数据之后，利用CreatThread函数创建读者和写者线程，然后等待其处理完之后进行。

对于读者优先的处理，我们利用一个read-counter变量来记录目前正在所有的读者数量，来确定要不要释放目前正在等待的写者线程，当读者数量为零的时候，这时候可以释放一个写线程。

另外当写者线程发出写请求的时候，要独占这个写资源并且其他的写线程无法使用。

对于写者有限的处理，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Write\_count，用于记录正在等待的写者的数目，当Write\_count=0时，才可以释放等待的读者线程队列。而当有写者在进行写文件或者等待的时候，读者也必须进行阻塞。

总原理：

1. **实验结果及数据分析**

输入数据为

1 R 3 5

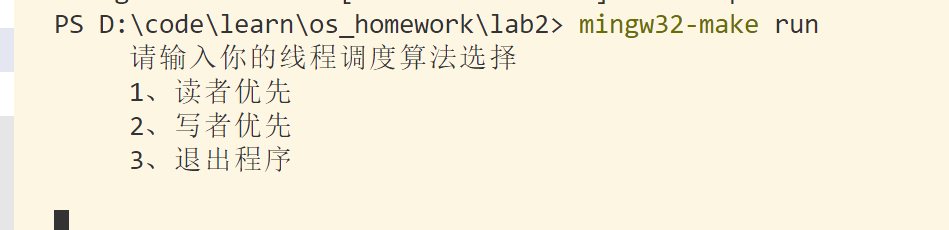
2 W 4 5

3 R 5 2

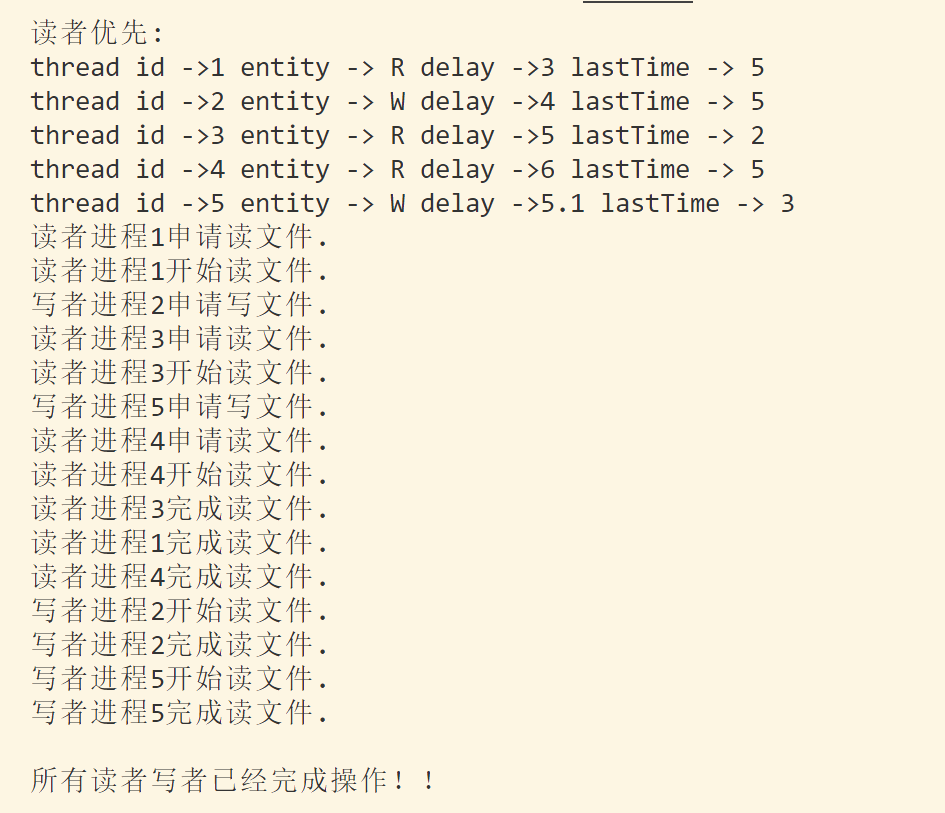
4 R 6 5

5 W 5.1 3

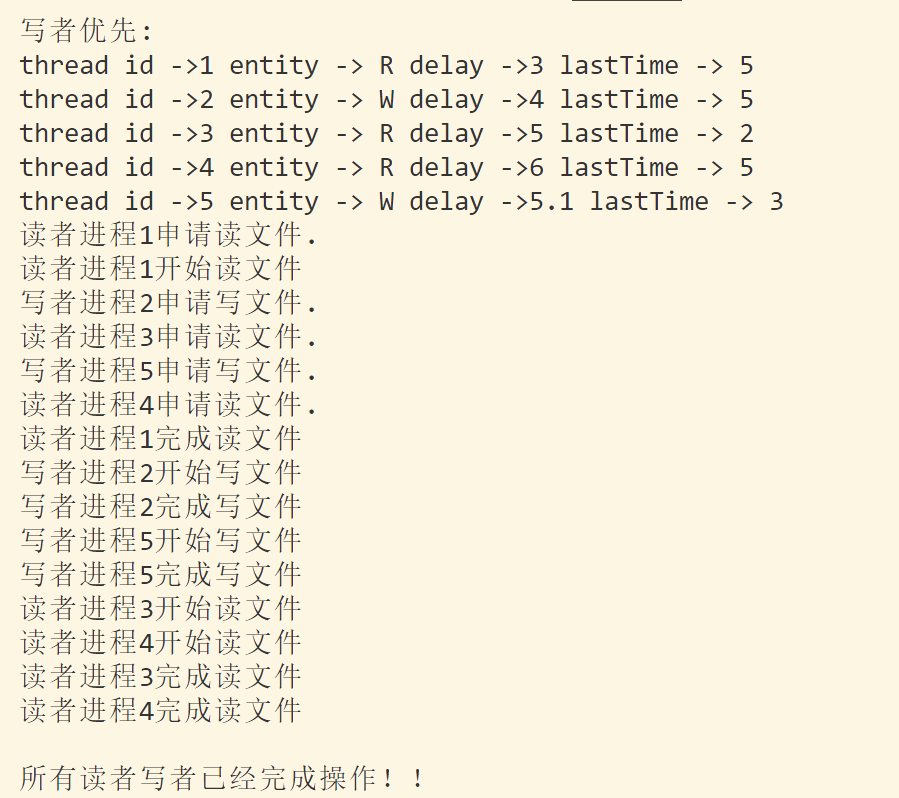
开始界面为



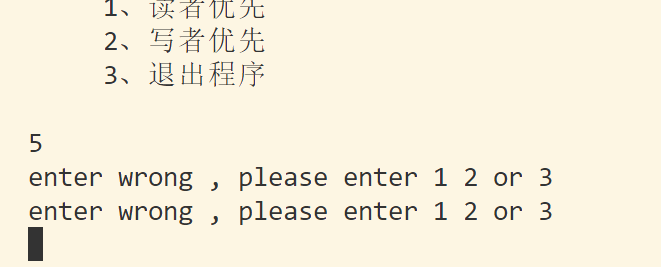
输入1，进入读者优先模式、



输入2，进入写者优先模式、



输入3，退出程序，键入其他数据提醒输入错误，请重新输入



1. **总结**

这一次课程设计，我完成了”读者-写者问题”的实现，更加系统的理解和掌握了读者写者问题的概念和解决方法，对于程序设计的思想也有了更加清晰的印象。总的感觉对于windows系统有了更加深刻的接触和了解。经过读者写者问题的编写，我对于同步机构的应用有了更加深入的了解，懂得了利用信号进行进程之间的互斥。实现了不让共享资源同时修改。用信号量的原语操作也使灵界段问题更加好解决了，更加简洁明了。

1. **附录**

├─ lab2

│  ├─ build

│  │  ├─ os\_homework\_2.d

│  │  └─ os\_homework\_2.exe

│  ├─ Makefile

│  ├─ os\_homework\_2.cpp

│  └─ thread.txt