



北京理工大学
Beijing Institute of Technology

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：	<input checked="" type="checkbox"/> 原理验证 <input type="checkbox"/> 综合设计 <input type="checkbox"/> 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：	



信息与电子学院

SCHOOL OF INFORMATION AND ELECTRONICS

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。

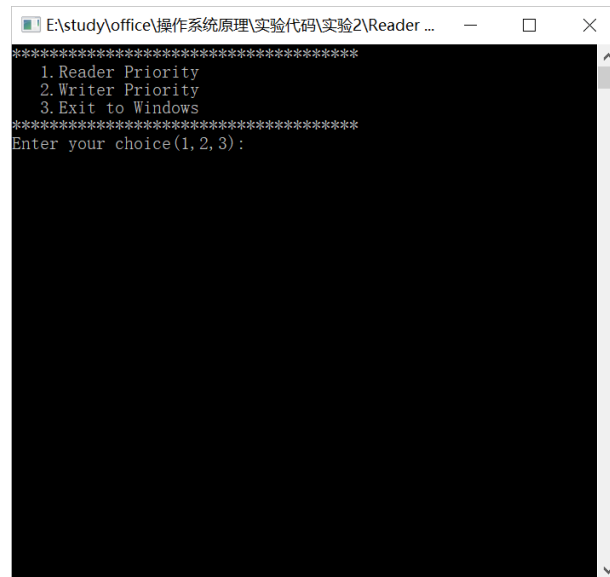
3) 信号量设置。首先设置信号量 `count` 为计数器，用来记录当前读者数量，初值为 0；设置 `mutex` 为互斥信号量，用于保护更新 `count` 变量时的互斥；设置互斥信号量 `rw` 用于保证读者和写者的互斥访问。

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。为此应当添加一个整形变量 `writcount`，用于记录正在等待的写者的数目，当 `writcount=0` 时，才可以释放等待的读者线程队列。

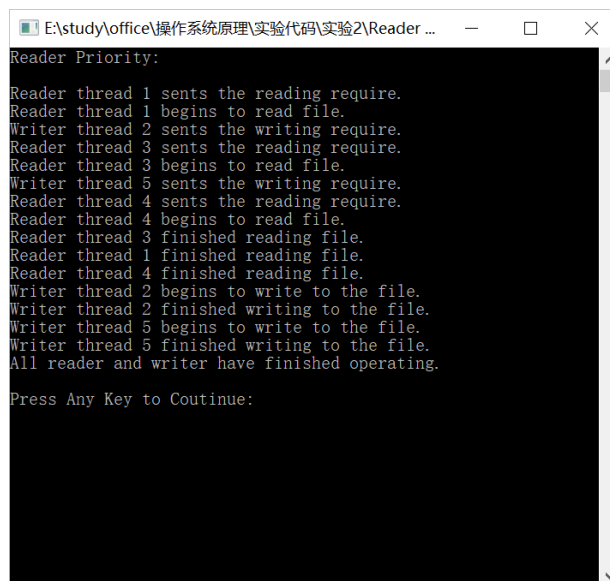
五、实验结果及数据分析

1. 图像：



```
E:\study\office\操作系统原理\实验代码\实验2\Reader ...
*****
1.Reader Priority
2.Writer Priority
3.Exit to Windows
*****
Enter your choice(1,2,3):
```

(开始界面)



```
E:\study\office\操作系统原理\实验代码\实验2\Reader ...
Reader Priority:
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 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 Coutinue:
```

(读者优先)

```
E:\study\office\操作系统原理\实验代码\实验2\Reader ...
Writer priority:
Reader thread 1 sends the reading require.
Reader thread 1 begins to read file.
Writer thread 2 sends the reading require.
Reader thread 3 sends the reading require.
Writer thread 5 sends the reading require.
Reader thread 4 sends 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 Coutinue: _
```

(写者优先)

```
E:\study\office\操作系统原理\实验代码\实验2\Re...
-----
Process exited after 63.18 seconds with return value 0
请按任意键继续. . .
```

(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

```



```
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 sends 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 sends 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;
}

```