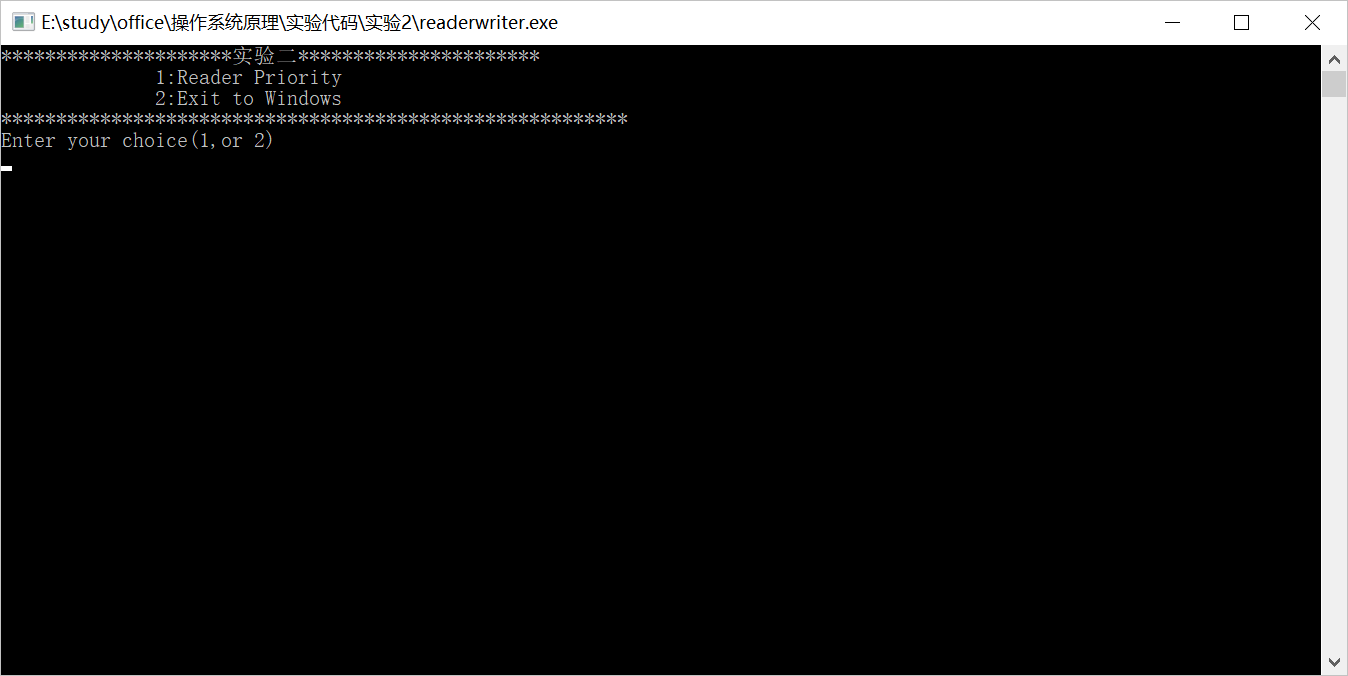
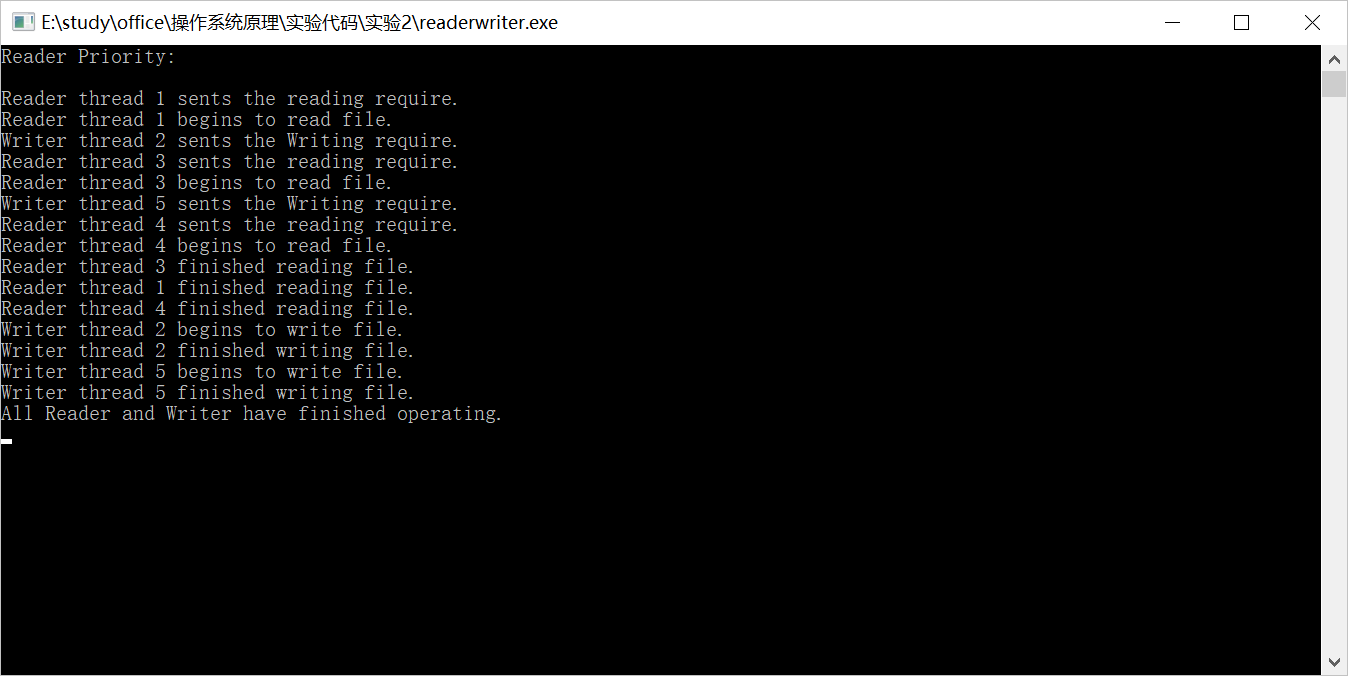
**实验二：读者写者问题**

结果：





代码：

//ReadWrite.cpp : 定义控制台应用程序的入口点。

//#include "stdafx.h"

#include <windows.h>

#include <conio.h>

#include <stdlib.h>

#include "fstream"

#include <io.h>

#include <string.h>

#include <stdio.h>

#define READER 'R'

#define WRITER 'W'

#define INTE\_PER\_SEC 100 //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;

//全局变量

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

double delay; //线程开始时间

double persist; //线程读写持续时间

};

//读者优先处理函数，已经给出

void ReaderPriority(char\* file);

void RP\_ReaderThread(void \*p);

void RP\_WriterThread(void \*p);

int main()

{

char ch;

while (true)

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*实验二\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" 1:Reader Priority\n");

printf(" 2:Exit to Windows\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Enter your choice(1,or 2) \n");

do

{

ch=(char) \_getch();

}while (ch!='1' && ch!='2');

system("cls");

if (ch=='2')

return 0;

else

ReaderPriority((char\*)"thread.dat");

printf("\nPress Any key to continue.\n");

\_getch();

system("cls");

}

return 0;

}

//读者优先处理函数

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;

n\_thread++;

inFile.get();

}

for (int i=0;i<(int)(n\_thread);i++)

{

if (thread\_info[i].entity==READER || thread\_info[i].entity=='r')

//创建读者线程

h\_Thread[i]=CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)(RP\_ReaderThread),&thread\_info[i],0,&thread\_ID);

else

//创建写者线程

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");

\_getch();

}

//读者优先-----读者线程

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

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\_mutex=WaitForSingleObject(h\_Mutex,-1); //等待互斥信号，保证对readcount的访问、修改互斥

readcount++; //读者数目增加

if (readcount==1)

EnterCriticalSection(&RP\_Write); //第一个读者，等待资源

ReleaseMutex(h\_Mutex);

//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);

wait\_for\_mutex=WaitForSingleObject(h\_Mutex,-1); //等待互斥信号，保证对readcount的访问、修改互斥

readcount--; //读者数目减少

if (readcount==0)

LeaveCriticalSection(&RP\_Write); //如果所有读者读完，唤醒写者

ReleaseMutex(h\_Mutex);

}

//读者优先-----写者线程

void RP\_WriterThread(void \*p)

{

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("Writer thread %d sents the Writing require.\n",m\_serial);

EnterCriticalSection(&RP\_Write);

//写文件

printf("Writer thread %d begins to write file.\n",m\_serial);

Sleep(m\_persist);

//退出线程

printf("Writer thread %d finished writing file.\n",m\_serial);

LeaveCriticalSection(&RP\_Write); //如果所有读者读完，唤醒写者

}