Windows操作系统

C/C++ 程序实验

姓名：\_\_\_\_\_\_\_陈展博\_\_\_\_\_\_

学号：\_\_\_\_\_\_\_1221001003\_\_

班级：\_\_\_\_\_\_\_计科1班\_\_\_\_

院系：\_\_\_\_\_\_\_信工\_\_\_\_\_\_\_\_

2024年11月17日

## 实验七 Windows 读者写者问题

### 一、背景知识

### 二、实验目的

### 三、工具/准备工作

### 四、实验步骤

#### 1. 读者写者问题

**步骤1**：登录进入Windows 。

**步骤2**：在“开始”菜单中单击“程序”-“Microsoft Visual Studio Code”。

**步骤3：**新建项目名为“7-1”，并且新建项“7-1.cpp”。

**步骤4：**将“thread.dat”文件复制到项目文件夹中。

**步骤5：**按“F5”开始调试，注意路径里不要含有中文。

**步骤6：**按暂停按钮可暂停程序的执行，按终止按钮可终止程序的执行。



操作能否正常进行？如果不行，则可能的原因是什么？

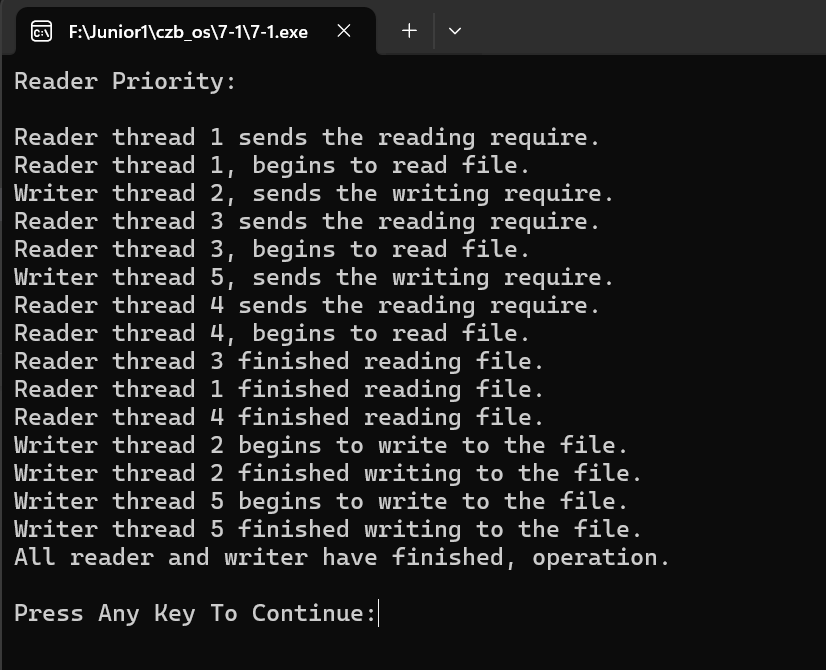
操作能够正常运行，如果不行，可能是因为文件路径中含有中文，或者代码中含有中文字符。

运行结果是：

文本

描述已自动生成

图片 1 对文件输入代码进行修改，改进开始会读入0号Write线程的问题



图片 2 读者优先运行结果

图形用户界面, 文本

描述已自动生成

图片 3 写者优先运行结果

**步骤7：**分析程序里是如何实现读者/写者优先的，详细描述实现流程。

读者优先：

ReaderPriority函数先从thread.dat文件中读入读者写者的基本信息，再分别创建读者，写者线程。在读者线程函数RP\_ReaderThread中定义了控制对readcount值修改的互斥变量HANDLE h\_Mutex;在读者线程数量变化时通过wait\_for\_mutex = WaitForSingleObject(h\_Mutex, -1);和ReleaseMutex(h\_Mutex);对h\_Mutex进行down/up操作，保证读者线程互斥访问readcount。并且在第一个读者进入临界区时，通过EnterCriticalSection(&RP\_Write);对读者写者共用的读写临界区的互斥信号量RP\_Write进行down操作，在最后一个读者离开临界区时，通过LeaveCriticalSection(&RP\_Write);对读写临界区的互斥信号量 RP\_Write进行up操作。实现了只要有读者在访问临界资源，写者必须等待全部读者访问完毕才能进入临界区的读者优先功能。

其中，写者线程函数RP\_WriterThread就只有简单的对读写临界区的互斥信号量 RP\_Write进行down/up操作。但优先级较低。

写者优先：

WriterPriority函数先从thread.dat文件中读入读者写者的基本信息，再分别创建读者，写者线程。在读者线程函数WP\_ReaderThread中定义了HANDLE h\_Mutex1;和HANDLE h\_Mutex2;其中h\_Mutex1用于控制读者进入读者临界区，在对h\_Mutex1进行down操作之后，马上进行EnterCriticalSection(&cs\_Read);等待进入读者临界区，mutex2则保证对readcount的访问、修改互斥，如果是第一个读者，等待写者写完再对写者临界区的互斥信号量进行down操作EnterCriticalSection(&cs\_Write);直到最后一个读者结束才通过LeaveCriticalSection(&cs\_Write)唤醒写者。

在写者线程函数WP\_WriterThread中，定义了h\_Mutex3保证对writecount的访问、修改互斥，同样在第一个写者创建时，等待读者读完再对写者临界区的互斥信号量进行down操作EnterCriticalSection(&cs\_Read);直到最后一个写者结束写操作调用LeaveCriticalSection(&cs\_Read);函数离开读者临界区，读者才可以读。这样的互斥信号量控制实现了写者优先。

**选作：**在熟悉清单7-1源代码的基础上，用P、V操作实现多个生产者—消费者问题。

测试数据文件包括n+1行测试数据，第一行说明几个缓冲区，其余n行分别描述创建的n个线程是生产者还是消费者，以及生产产品（或消费产品）的时间。每行测试数据包括几个字段，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。第二字段表示相应线程角色，P表示生产者，C表示消费者。第三字段为一个正数，表示生产产品（或消费产品）的时间。消费者还可以有几个字段，分别表示此消费者消费哪些生产者（线程号）生产的产品。缓冲区需互斥访问。请描述你所做的工作：

文本

描述已自动生成

图片 4 修改主函数

文本

描述已自动生成

图片 5 预先设置缓冲区，缓冲区槽个数为2，临界区，缓冲区队列，信号量Mutex/Empty/Full

文本

描述已自动生成

图片 6 生产者消费者问题函数（1）

文本

描述已自动生成

图片 7 生产者消费者问题函数（2），读入文件，创建线程

表格

中度可信度描述已自动生成

图片 8 thread.dat文件内容（这里只用了一个缓冲区）

文本

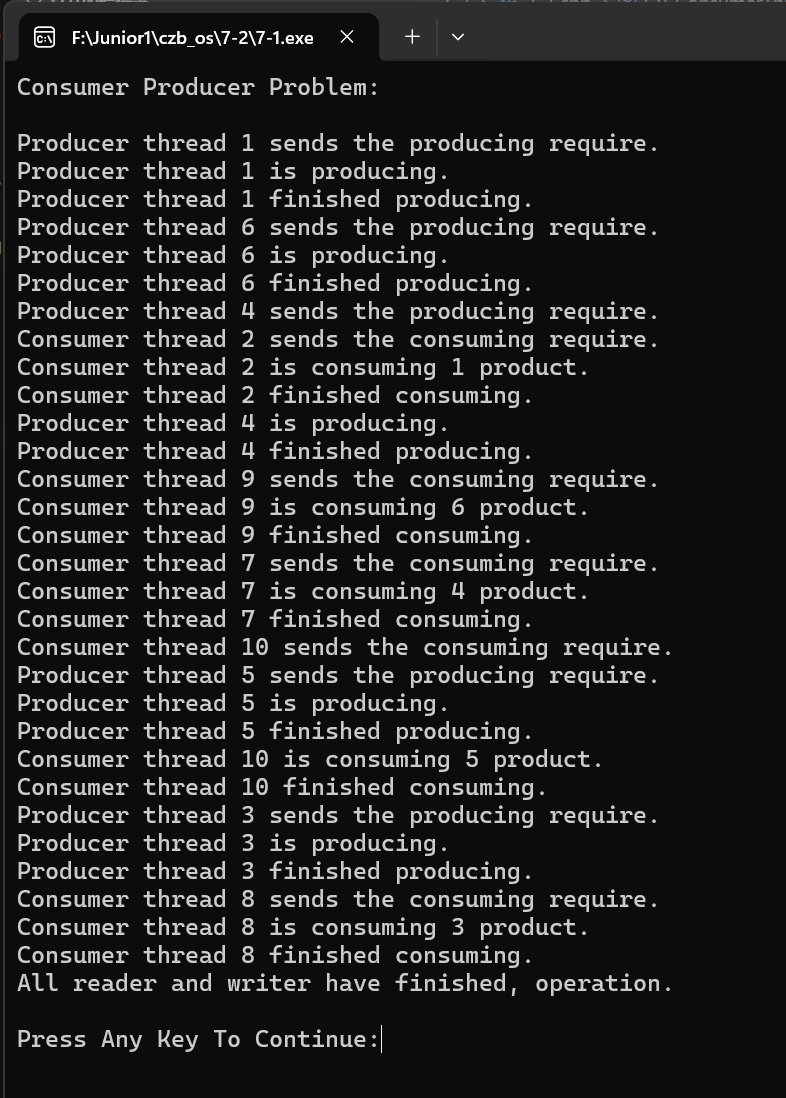
描述已自动生成

图片 9 生产者线程函数，先同步再互斥访问缓冲区临界资源，生产时将自己的线程序号压入Buffer队列，离开临界区时，给Full信号量up

文本

描述已自动生成

图片 10 消费者线程函数，先同步再互斥访问缓冲区临界资源，消费时输出消费的产品序号，并且将该产品推出队列，离开临界区时给Empty信号量up



图片 11 运行结果，由于缓冲区只有两个槽存放产品，所以生产者线程4在发出请求后被阻塞，等到消费者线程2消费掉产品1后才允许生产者4继续生产。同理，当缓冲区槽为空时，消费者线程10被阻塞，直到生产者线程5生产完之后才能消费。

附源代码7-2.cpp:

#include "windows.h"

#include <conio.h>

#include <stdlib.h>

#include <fstream>

#include <io.h>

#include <string.h>

#include <queue>

#include <stdio.h>

using namespace std;

#define CONSUMER 'C' //消费者

#define PRODUCER 'P' //生产者

#define INTE\_PER\_SEC 1000 //每秒钟中断数目

#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

#define N 2 //一个buffer最多有2个槽

int buffercount = 0; //缓冲区数目

queue <int> Buffer;

CRITICAL\_SECTION CP\_Buffer; //临界区

struct ThreadInfo

{

int serial; //线程序号

char entity; //线程类别

double delay; //线程延迟时间

};

// 定义信号量

HANDLE Mutex;

HANDLE Empty;

HANDLE Full;

void CP\_ProducerThread(void \*p)

{

//互斥变量

Mutex = OpenMutex(MUTEX\_ALL\_ACCESS, FALSE, "Mutex");

DWORD wait\_for\_empty; //等待互斥变量所有权

DWORD wait\_for\_mutex; //等待互斥变量对缓冲区的所有权

DWORD m\_delay; //延迟时间

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);

Sleep(m\_delay); //延迟等待

printf("Producer thread %d sends the producing require.\n", m\_serial);

wait\_for\_empty = WaitForSingleObject(Empty, INFINITE);

wait\_for\_mutex = WaitForSingleObject(Mutex, -1);

//进入缓冲区

EnterCriticalSection(&CP\_Buffer);

printf("Producer thread %d is producing.\n", m\_serial);//生产过程

Buffer.push(m\_serial);

LeaveCriticalSection(&CP\_Buffer);

//退出线程

printf("Producer thread %d finished producing.\n", m\_serial);

ReleaseMutex(Mutex);

// 函数原型

// BOOL ReleaseSemaphore(HANDLE hSemaphore, LONG lReleaseCount,LPLONG lpPreviousCount );信号量的句柄/要增加的计数值/（可选）指向变量的指针，用于接收信号量释放操作前的计数值

ReleaseSemaphore(Full, 1, NULL); //Full信号量+1

}

void CP\_ConsumerThread(void \*p)

{

//互斥变量

Mutex = OpenMutex(MUTEX\_ALL\_ACCESS, FALSE, "Mutex");

DWORD wait\_for\_full; //等待互斥变量所有权

DWORD wait\_for\_mutex; //等待互斥变量对缓冲区的所有权

DWORD m\_delay; //延迟时间

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);

Sleep(m\_delay); //延迟等待

printf("Consumer thread %d sends the consuming require.\n", m\_serial);

wait\_for\_full = WaitForSingleObject(Full, INFINITE);

wait\_for\_mutex = WaitForSingleObject(Mutex, -1);

//进入缓冲区

EnterCriticalSection(&CP\_Buffer);

printf("Consumer thread %d is consuming %d product.\n", m\_serial,Buffer.front());//生产过程

Buffer.pop();

LeaveCriticalSection(&CP\_Buffer);

//退出线程

printf("Consumer thread %d finished consuming.\n", m\_serial);

ReleaseMutex(Mutex);

ReleaseSemaphore(Empty, 1, NULL); //Empty信号量+1

}

void ConsumerProducer(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

Mutex = CreateMutex(NULL, FALSE, "Mutex");

// 参数表 CreateSemaphore(LPSECURITY\_ATTRIBUTES lpSemaphoreAttributes, LONG lInitialCount, LONG lMaximumCount, LPCSTR lpName);

Empty = CreateSemaphore(NULL, N, N, NULL); //初始值为N最大值为N

Full = CreateSemaphore(NULL, 0, N, NULL); //初始值为0最大值为N

//线程对象

HANDLE h\_Thread[MAX\_THREAD\_NUM];

ThreadInfo thread\_info[MAX\_THREAD\_NUM];

buffercount = 0; //初始化readcount

InitializeCriticalSection(&CP\_Buffer); //初始化临界区

ifstream inFile;

inFile.open(file); //打开文件

printf("Consumer Producer Problem:\n\n");

bool first\_in\_flag = 1;

// 读取首次的特殊信息

if (first\_in\_flag && (inFile >> buffercount)) {

inFile.get(); // 消费掉后面的换行符或其他字符

// printf("first read\n");

first\_in\_flag = 0; // 确保这段代码不会再次执行

}

// 读取每个线程的信息

while (inFile >> thread\_info[n\_thread].serial >> thread\_info[n\_thread].entity >> thread\_info[n\_thread].delay) {

inFile.get(); // 处理行尾的换行符

n\_thread++; // 计数器递增

}

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

{

if (thread\_info[i].entity == CONSUMER || thread\_info[i].entity == 'c')

{

//创建消费者线程

h\_Thread[i] = CreateThread(NULL, 0,

(LPTHREAD\_START\_ROUTINE)(CP\_ConsumerThread),

&thread\_info[i], 0, &thread\_ID);

}

else{

//创建生产者线程

h\_Thread[i] = CreateThread(NULL, 0,

(LPTHREAD\_START\_ROUTINE)(CP\_ProducerThread),

&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, operation.\n");

}

////////////////////////////////////////////////////////////////////////////

//主函数

int main(int argc, char\* argv[])

{

char ch;

while (true)

{

//打印提示信息

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

printf(" 1:Consumer Producer Problem\n");

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

printf("Enter your choice(1 or 2):");

//如果输入信息不正确，继续输入

do

{

ch = (char)\_getch();

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

system("cls");

//选择2，返回

if (ch == '2')

return 0;

//选择1，生产者消费者问题

else if (ch == '1')

ConsumerProducer("thread.dat");

printf("\nPress Any Key To Continue:");

\_getch();

system("cls");

}

return 0;

}

附thread.dat

1

1 P 3

2 C 4

3 P 5.3

4 P 3.8

5 P 5.1

6 P 3.4

7 C 4.3

8 C 5.5

9 C 4.2

10 C 5