

# 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。



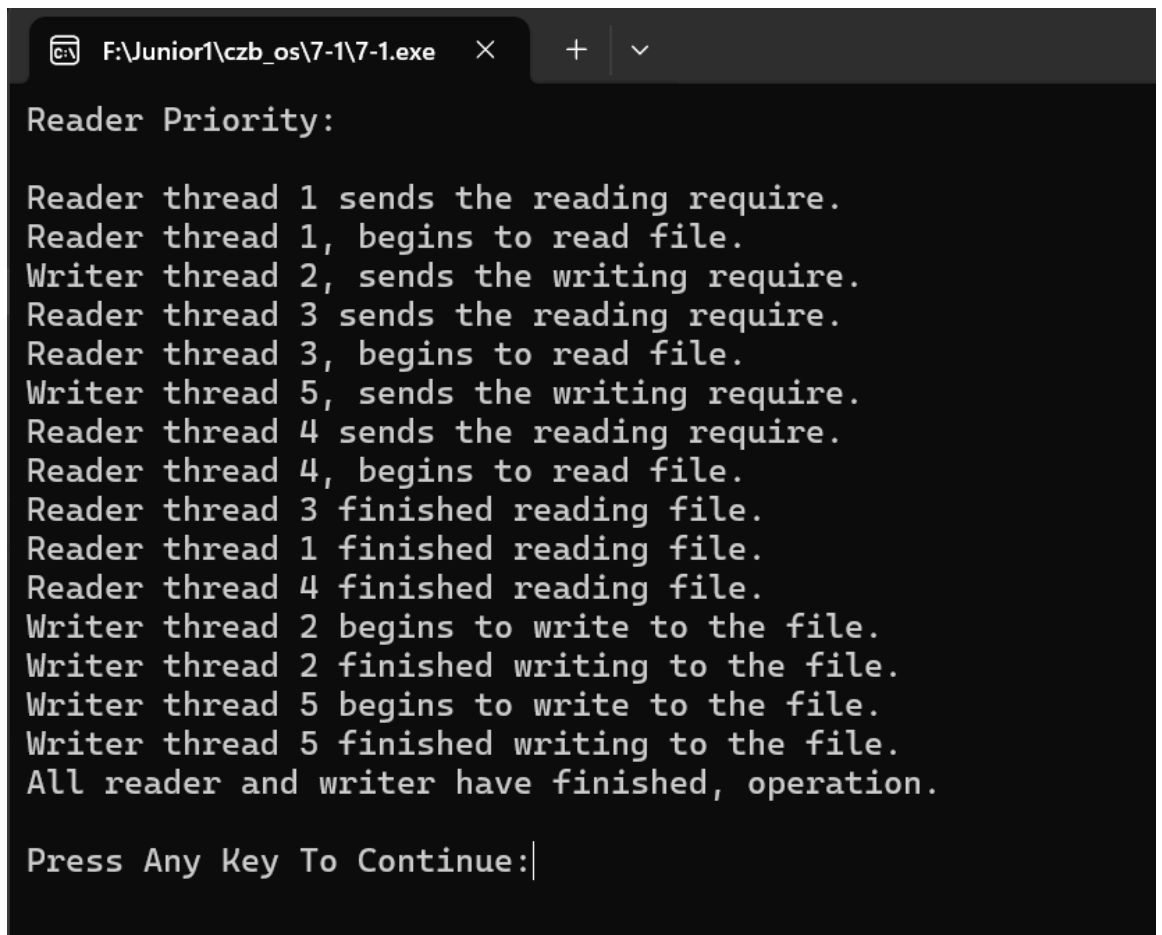
操作能否正常进行？如果不行，则可能的原因是什么？

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：

```
/*while (inFile)
{
    //读入第一个读者、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();
}*/
while (inFile >> thread_info[n_thread].serial >> thread_info[n_thread].entity >> thread_info[n_thread].delay >> thread_info[n_thread].persist) {
    inFile.get(); // 处理行尾的换行符
    n_thread++; // 计数器递增
}
```

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



The screenshot shows a Windows command prompt window with a single tab titled 'F:\Junior1\czb\_os\7-1\7-1.exe'. The window has a dark background and white text. The text output is as follows:

```
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, operation.

Press Any Key To Continue:|
```

图片 2 读者优先运行结果

```
F:\Junior1\czb_os\7-1\7-1.exe  ×  +  ∨

Writer 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.
Writer thread 5 sends the writing 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, operation.

Press Any Key To Continue:|
```

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:

```

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;
}

```

图片 4 修改主函数

```

11  #define CONSUMER 'C'          //消费者
12  #define PRODUCER 'P'         //生产者
13
14  #define INTE_PER_SEC 1000     //每秒钟中断数目
15  #define MAX_THREAD_NUM 64    //最大线程数目
16  #define MAX_FILE_NUM 32      //最大数据文件数目
17  #define MAX_STR_LEN 32       //字符串长度
18
19  #define N 2 //一个buffer最多有2个槽
20
21  int buffercount = 0;          //缓冲区数目
22
23  queue <int> Buffer;
24
25  CRITICAL_SECTION CP_Buffer; //临界区
26
27  struct ThreadInfo
28  {
29      int    serial;           //线程序号
30      char   entity;           //线程类别
31      double delay;           //线程延迟时间
32  };
33
34  // 定义信号量
35  HANDLE Mutex;
36  HANDLE Empty;
37  HANDLE Full;
38

```

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

```

130 void ConsumerProducer(char* file)
131 {
132     DWORD n_thread = 0;           //线程数目
133     DWORD thread_ID;              //线程ID
134     DWORD wait_for_all;           //等待所有线程结束
135
136     //互斥对象
137     Mutex = CreateMutex(NULL, FALSE, "Mutex");
138     // 参数表 CreateSemaphore(LPSECURITY_ATTRIBUTES lpSemaphoreAttributes, LONG lInitialCount, LONG lMaximumCount, LPCSTR lpName);
139     Empty = CreateSemaphore(NULL, N, N, NULL); //初始值为N最大值为N
140     Full = CreateSemaphore(NULL, 0, N, NULL); //初始值为0最大值为N
141
142     //线程对象
143     HANDLE h_Thread[MAX_THREAD_NUM];
144     ThreadInfo thread_info[MAX_THREAD_NUM];
145
146     buffercount = 0;               //初始化readcount
147     InitializeCriticalSection(&CP_Buffer); //初始化临界区
148     ifstream inFile;
149     inFile.open(file);             //打开文件
150     printf("Consumer Producer Problem:\n\n");
151     bool first_in_flag = 1;
152
153

```

图片 6 生产者消费者问题函数（1）

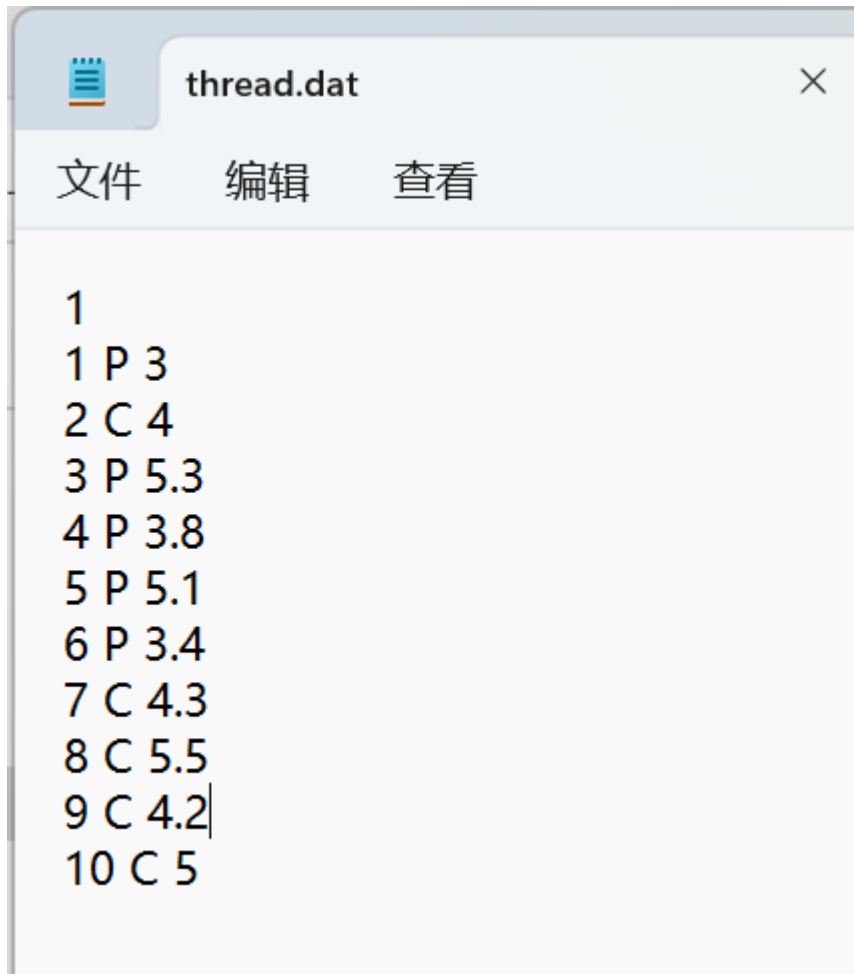
```

154 // 读取首次的特殊信息
155 if (first_in_flag && (inFile >> buffercount)) {
156     inFile.get(); // 消费掉后面的换行符或其他字符
157     printf("first read\n");
158     first_in_flag = 0; // 确保这段代码不会再次执行
159 }
160
161 // 读取每个线程的信息
162 while (inFile >> thread_info[n_thread].serial >> thread_info[n_thread].entity >> thread_info[n_thread].delay) {
163     inFile.get(); // 处理行尾的换行符
164     n_thread++; // 计数器递增
165 }
166
167 for (int i = 0; i < (int)(n_thread); i++)
168 {
169     if (thread_info[i].entity == CONSUMER || thread_info[i].entity == 'c')
170     {
171         //创建消费者线程
172         h_Thread[i] = CreateThread(NULL, 0,
173             (LPTHREAD_START_ROUTINE)(CP_ConsumerThread),
174             &thread_info[i], 0, &thread_ID);
175     }
176     else{
177         //创建生产者线程
178         h_Thread[i] = CreateThread(NULL, 0,
179             (LPTHREAD_START_ROUTINE)(CP_ProducerThread),
180             &thread_info[i], 0, &thread_ID);
181     }
182 }
183
184 //等待所有线程结束
185 wait_for_all = WaitForMultipleObjects(n_thread, h_Thread, TRUE, -1);
186 printf("All reader and writer have finished, operation.\n");
187

```

图片 7 生产者消费者问题函数（2），读入文件，创建线程





图片 8 thread.dat 文件内容（这里只用了一个缓冲区）

```

39 void CP_ProducerThread(void *p)
40 {
41     //互斥变量
42     Mutex = OpenMutex(MUTEX_ALL_ACCESS, FALSE, "Mutex");
43
44     DWORD wait_for_empty;    //等待互斥变量所有权
45
46     DWORD wait_for_mutex;    //等待互斥变量对缓冲区的所有权
47
48     DWORD m_delay;           //延迟时间
49     int m_serial;            //线程序号
50     //从参数中获得信息
51     m_serial = ((ThreadInfo*)(p))->serial;
52     m_delay = (DWORD)(((ThreadInfo*)(p))->delay * INTE_PER_SEC);
53     Sleep(m_delay);          //延迟等待
54
55     printf("Producer thread %d sends the producing require.\n", m_serial);
56     wait_for_empty = WaitForSingleObject(Empty, INFINITE);
57     wait_for_mutex = WaitForSingleObject(Mutex, -1);
58
59     //进入缓冲区
60     EnterCriticalSection(&CP_Buffer);
61
62     printf("Producer thread %d is producing.\n", m_serial); //生产过程
63     Buffer.push(m_serial);
64     LeaveCriticalSection(&CP_Buffer);
65
66     //退出线程
67     printf("Producer thread %d finished producing.\n", m_serial);
68
69     ReleaseMutex(Mutex);
70     // 函数原型
71     // 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
72     ReleaseSemaphore(Full, 1, NULL); //Full信号量+1
73 }
74

```

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

```
75 void CP_ConsumerThread(void *p)
76 {
77     //互斥变量
78     Mutex = OpenMutex(MUTEX_ALL_ACCESS, FALSE, "Mutex");
79
80     DWORD wait_for_full;        //等待互斥变量所有权
81
82     DWORD wait_for_mutex;       //等待互斥变量对缓冲区的所有权
83
84     DWORD m_delay;              //延迟时间
85     int m_serial;               //线程序号
86     //从参数中获得信息
87     m_serial = ((ThreadInfo*)(p))->serial;
88     m_delay = (DWORD)((ThreadInfo*)(p))->delay * INTE_PER_SEC;
89     Sleep(m_delay);             //延迟等待
90
91     printf("Consumer thread %d sends the consuming require.\n", m_serial);
92     wait_for_full = WaitForSingleObject(Full, INFINITE);
93     wait_for_mutex = WaitForSingleObject(Mutex, -1);
94
95     //进入缓冲区
96     EnterCriticalSection(&CP_Buffer);
97
98     printf("Consumer thread %d is consuming %d product.\n", m_serial, Buffer.front()); //生产过程
99
100    Buffer.pop();
101
102    LeaveCriticalSection(&CP_Buffer);
103
104    //退出线程
105    printf("Consumer thread %d finished consuming.\n", m_serial);
106
107    ReleaseMutex(Mutex);
108
109    ReleaseSemaphore(Full, 1, NULL); //Full信号量+1
110 }
```

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 Full 信号量 up

```
F:\Junior1\czb_os\7-2\7-1.exe X + v
Consumer Producer Problem:

Producer thread 1 sends the producing require.
Producer thread 1 is producing.
Producer thread 1 finished producing.
Producer thread 6 sends the producing require.
Producer thread 6 is producing.
Producer thread 6 finished producing.
Producer thread 4 sends the producing require.
Consumer thread 2 sends the consuming require.
Consumer thread 2 is consuming 1 product.
Consumer thread 2 finished consuming.
Producer thread 4 is producing.
Producer thread 4 finished producing.
Consumer thread 9 sends the consuming require.
Consumer thread 9 is consuming 6 product.
Consumer thread 9 finished consuming.
Consumer thread 7 sends the consuming require.
Consumer thread 7 is consuming 4 product.
Consumer thread 7 finished consuming.
Consumer thread 10 sends the consuming require.
Producer thread 5 sends the producing require.
Producer thread 5 is producing.
Producer thread 5 finished producing.
Consumer thread 10 is consuming 5 product.
Consumer thread 10 finished consuming.
Producer thread 3 sends the producing require.
Producer thread 3 is producing.
Producer thread 3 finished producing.
Consumer thread 8 sends the consuming require.
Consumer thread 8 is consuming 3 product.
Consumer thread 8 finished consuming.
All reader and writer have finished, operation.

Press Any Key To Continue:
```

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
InitialCount, 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
```