实验一 生产者和消费者问题

一、 实验题目

在 Windows 7 环境下，创建一个控制台进程，在此进程中创建 n 个线程来模拟生产者或消费者。这些线程的信息由我们在本程序定义的“测试用例文件”中予以指定。

实验任务：输出下面 2 个“测试用例文件”的运行结果，并加以解释。

“测试用例文件 1”内容：

```
3
1  P  3
2  P  4
3  C  4  1
4  P  2
5  C  3  1  4  2
```

“测试用例文件 2”内容：

```
2
1  P  2
2  C  1  3  1  4
3  P  4
4  P  3
```

二、实验步骤

实验主要代码：

首先定义一个结构，记录在测试文件中指定的每一个线程的参数：

```
struct ThreadInfo
{
    int serial;                             //线程序列号
    char entity;                            //是P还是C
    float delay;                            //线程延迟
    int thread_request[MAX_THREAD_NUM];     //线程请求队列，消费者才有
    int n_request;                          //请求个数，消费者才有
};

int main(void)
{
    int i, j;
    for(i=0;i<MAX_THREAD_NUM;i++)       //初始化每个线程的请求队列；
    {
        for(j=0;j<n_Thread;j++)
            Thread_Info[i].thread_request[j]=-1;
        Thread_Info[i].n_request=0;
    }
    for(i=0;i<MAX_THREAD_NUM+1;i++)      //产品要消费的次数设置为0
        num_of_c_time[i]=0;
    for(i=0;i<MAX_BUFFER_NUM;i++)      //初始化缓冲区
        Buffer_Critical[i]=-1;
    for(i=0;i<MAX_BUFFER_NUM;i++)
    InitializeCriticalSection(&PC_Critical[i]);
    readFile();
    printf("输入文件是:\n");    //回显获得的线程信息，便于确认正确性
    printf("%d\n",n_Buffer_or_Critical);
    for(i=0;i< n_Thread;i++)
    {
        int Temp_serial=Thread_Info[i].serial;
        char Temp_entity=Thread_Info[i].entity;
        float Temp_delay=Thread_Info[i].delay;
        printf("thread%2d %c %f",Temp_serial,Temp_entity,Temp_delay);
        int Temp_request=Thread_Info[i].n_request;
        for(j=0;j<Temp_request;j++)
            printf("%d",Thread_Info[i].thread_request[j]);
        printf("\n");
    }
    printf("\n\n");
    empty_semaphore=CreateSemaphore(NULL,n_Buffer_or_Critical,n_Buffer_or_Criti
cal, "semaphore_for_empty");
    h_mutex=CreateMutex(NULL,FALSE,"mutex_for_produce");
    for (i=0;i<n_Buffer_or_Critical;i++)
    {
        char name[30]="semaphore_for_buffer_";
        char serial[2];
        itoa(i,serial,10);
        strcat(name,serial);
        h_mutex_Buffer[i]=CreateMutex(NULL,FALSE,name);
    }
    //下面这个循环用线程的 ID 号来为相应生产线程的产品读写时所使用的同步信号量命
名；
    for(j=0;j<n_Thread;j++)
    {
        if(Thread_Info[j].entity=='P')
        {
            char name[30] ={"semaphore_for_producer_"};
            char serial[2];
            itoa(j+1,serial,10);
            strcat(name,serial);
            h_Semaphore[j]=CreateSemaphore(NULL,0,n_Thread,name);
        }
    }
    for(i=0;i< n_Thread;i++)     //创建生产者和消费者线程；
    {
        if(Thread_Info[i].entity=='P')
    h_Thread[i]=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)(Produce,&(Thread_I
nfo[i]),0,NULL);
        else
    h_Thread[i]=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)(Consume),&(Thread_
Info[i]),0,NULL) x;
    }
    WaitForMultipleObjects(n_Thread,h_Thread,TRUE,INFINITE);
    printf("\n\nALL Producer and consumer have finished their work.\n");
    return 0;
}
void readFile()
{
    FILE *inFile;
    inFile=fopen("test.txt","r");
    //打开输入文件，按照规定的格式提取线程等信息；
    if (inFile==NULL)
    {
        printf("Can't open test.txt.\n");
        exit(EXIT_FAILURE);
    }
    //从文件中获得实际的缓冲区的数目；
    fscanf(inFile,"%d",&n_Buffer_or_Critical);
    char c=getc(inFile);
    while(c!=EOF)
    {
        if (fscanf(inFile,"%d",&Thread_Info[n_Thread].serial)<1) break;
        getc(inFile);
        fscanf(inFile,"%c",&Thread_Info[n_Thread].entity);
        fscanf(inFile,"%f",&Thread_Info[n_Thread].delay);
        c=getc(inFile);
        while(c!= ’\n’&&c!=EOF)
        {
fscanf(inFile,"%d",&Thread_Info[n_Thread].thread_request[Thread_Info[n_Thread].
n_request]);
num_of_c_time[Thread_Info[n_Thread].thread_request[Thread_Info[n_Thread].n_requ
est]]++;
            Thread_Info[n_Thread].n_request++;
            c=getc(inFile);
        }
        n_Thread++;
    }
    fclose(inFile);
}
```

该函数用于确认是否还有对同一产品的消费请求未执行：

```
bool IfInOtherRequest(int req)
{
    for(int i=0;i<n_Thread;i++)
        for(int j=0;j<Thread_Info[i].n_request;j++)
            if(Thread_Info[i].thread_request[j]==req)
                return TRUE;
    return FALSE;
}
```

该函数用于找出当前可以进行产品生产的空缓冲区位置：

```
int FindProducePosition()
{
    int EmptyPosition;
    for (int i=0;i<n_Buffer_or_Critical;i++)
        if(Buffer_Critical[i]==-1)
        {
            EmptyPosition=i;
            Buffer_Critical[i]=-2;
            break;
        }
    return  EmptyPosition;
}
```

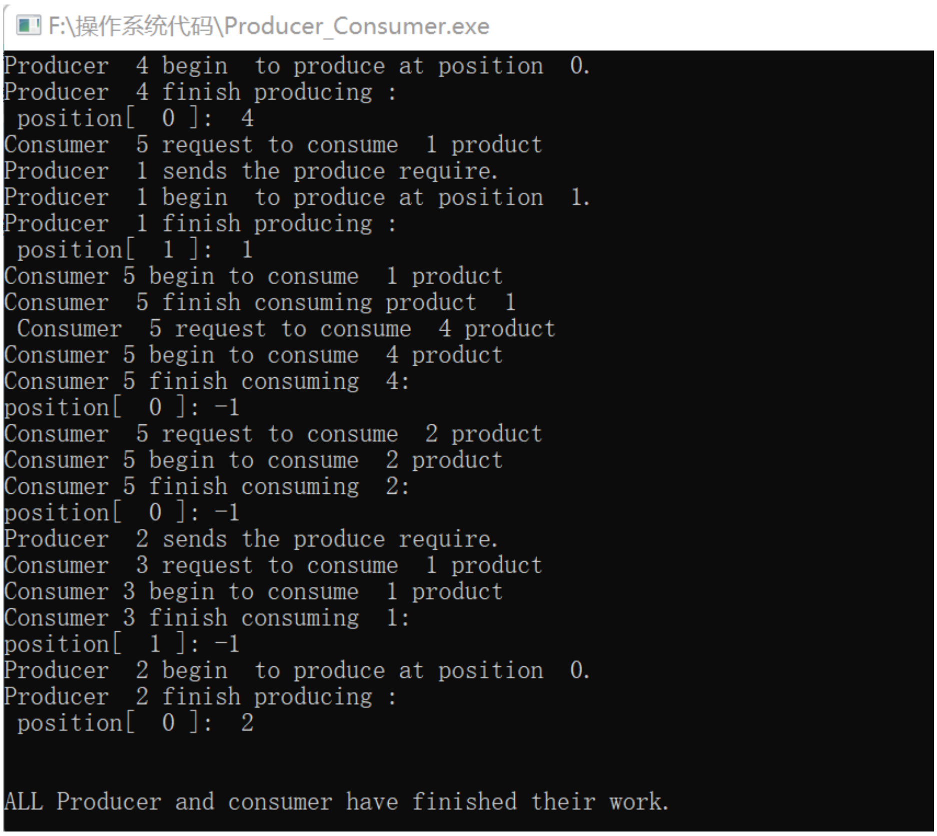
该函数用于找出当前所需生产者生产的产品的位置：

```
int FindBufferPosition(int ProPos) //Propos 是生产者序号
{
    int TempPos;
    for (int i=0;i<n_Buffer_or_Critical;i++)
        if(Buffer_Critical[i]==ProPos)
        {
            TempPos=i;
            break;
        }
    return TempPos;
}

//生产者进程
void Produce(void *p)
{
    DWORD wait_for_semaphore,wait_for_mutex,m_delay;
    int m_serial;
    m_serial=((ThreadInfo*)(p))->serial;   //获得本线程的信息；
    m_delay=(DWORD)(((ThreadInfo*)(p))->delay *INTE_PER_SEC);
    Sleep(m_delay);
//开始请求生产
    printf("Producer %2d sends the produce require.\n",m_serial);
//确认有空缓冲区可供生产，同时将空位置数 empty 减 1；用于生产者和消费者的同步；
    wait_for_semaphore=WaitForSingleObject(empty_semaphore,-1);
//互斥访问下一个可用于生产的空临界区，实现写写互斥；
    wait_for_mutex=WaitForSingleObject(h_mutex,-1);
    int  ProducePos=FindProducePosition();
    ReleaseMutex(h_mutex);
    printf("Producer %2d begin to produce at
position %2d.\n",m_serial,ProducePos);
    Buffer_Critical[ProducePos]=m_serial;
    printf("Producer %2d finish producing :\n ",m_serial);
    printf("position[%2d]:%3d \n" ,ProducePos,Buffer_Critical[ProducePos]);
//使生产者写的缓冲区可以被多个消费者使用，实现读写同步；
    ReleaseSemaphore(h_Semaphore[m_serial],n_Thread,NULL);
}

//消费者进程
void Consume(void *p)
{
    DWORD wait_for_semaphore,m_delay;
    int m_serial,m_requestNum; //消费者的序列号和请求的数目;
    int m_thread_request[MAX_THREAD_NUM];//本消费线程的请求队列;
//提取本线程的信息到本地;
    m_serial=((ThreadInfo*)(p))->serial;
    m_delay=(DWORD)(((ThreadInfo*)(p))->delay *INTE_PER_SEC);
    m_requestNum=((ThreadInfo *)(p))->n_request;
    for (int i=0;i<m_requestNum;i++)
    m_thread_request[i]=((ThreadInfo*)(p))->thread_request[i];
    Sleep(m_delay);
//循环进行所需产品的消费
    for(int i=0;i<m_requestNum;i++){
//请求消费下一个产品
    printf("Consumer %2d request to consume %2d
product\n",m_serial,m_thread_request[i]);
//如果对应生产者没有生产，则等待；如果生产了,允许的消费者数目-1；实现了读写同步；
    wait_for_semaphore=WaitForSingleObject(h_Semaphore[m_thread_request[i]],-1)
;
//查询所需产品放到缓冲区的号
    DWORD BufferPos=FindBufferPosition(m_thread_request[i]);
    EnterCriticalSection(&PC_Critical[BufferPos]);
    printf("Consumer%2d begin to consume %2d product
\n",m_serial,m_thread_request[i]);
    ((ThreadInfo*)(p))->thread_request[i] =-1;
    if(!IfInOtherRequest(m_thread_request[i])){
    Buffer_Critical[BufferPos] = -1; //标记缓冲区为空;
    printf("Consumer%2d finish consuming %2d:\n",m_serial,m_thread_request[i]);
    printf("position[ %2d ]:%3d \n",BufferPos,Buffer_Critical[BufferPos]);
    ReleaseSemaphore(empty_semaphore,1,NULL);
}
else{
    printf("Consumer %2d finish consuming product %2d\n
",m_serial,m_thread_request[i]);
}
LeaveCriticalSection(&PC_Critical[BufferPos]);   //离开临界区
}
}
```

“测试用例文件 1”实验结果：



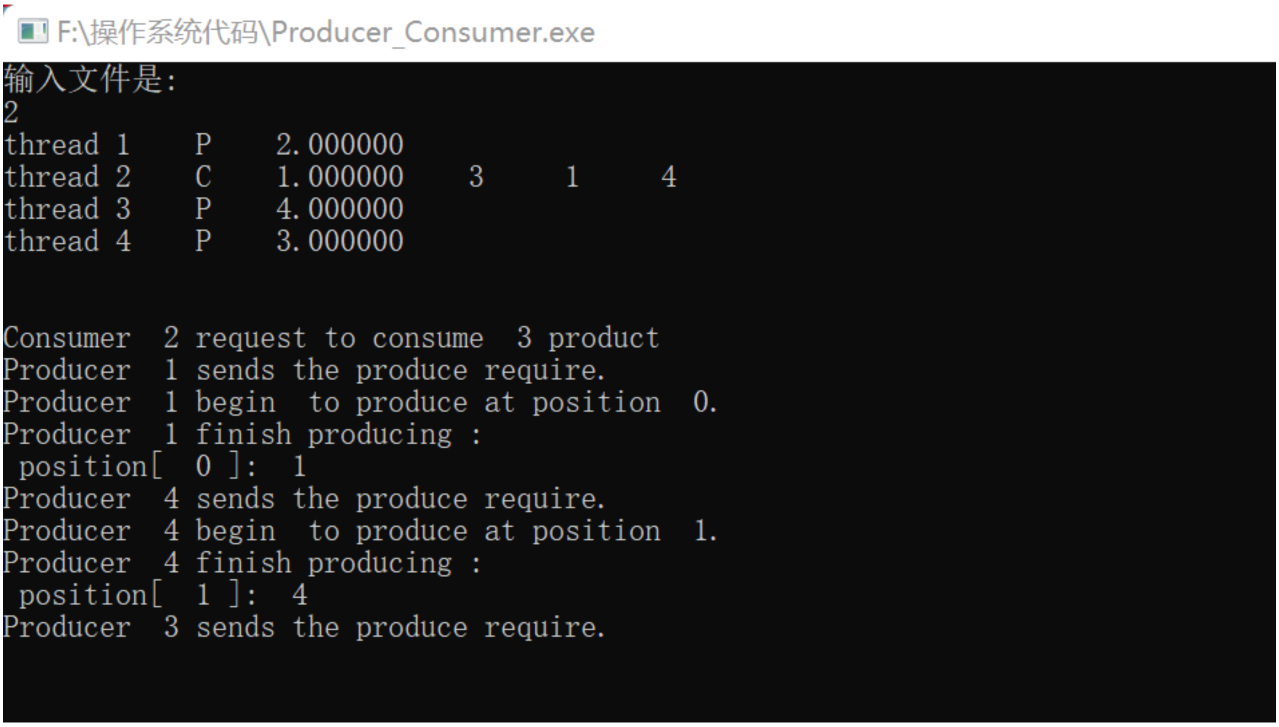
```
Producer  4 begin  to produce at position  0.
Producer  4 finish producing :
 position[  0 ]:  4
Consumer  5 request to consume  1 product
Producer  1 sends the produce require.
Producer  1 begin  to produce at position  1.
Producer  1 finish producing :
 position[  1 ]:  1
Consumer 5 begin to consume  1 product
Consumer  5 finish consuming product  1
 Consumer  5 request to consume  4 product
Consumer 5 begin to consume  4 product
Consumer 5 finish consuming  4:
position[  0 ]: -1
Consumer  5 request to consume  2 product
Consumer 5 begin to consume  2 product
Consumer 5 finish consuming  2:
position[  0 ]: -1
Producer  2 sends the produce require.
Consumer  3 request to consume  1 product
Consumer 3 begin to consume  1 product
Consumer 3 finish consuming  1:
position[  1 ]: -1
Producer  2 begin  to produce at position  0.
Producer  2 finish producing :
 position[  0 ]:  2


ALL Producer and consumer have finished their work.
```

结果解释：

首先生产者 4 开始生产产品，接下来消费者 5 开始请求 1 号生产者的产品，由于该产品还不存在，因此本线程被阻塞。接下来是生产者 1 生产产品，5 号消费者开始消费 1 号生产者的产品，继续消费 4 号生产者的产品，并且释放该产品所占的缓存区间，由于 2 号产品还未生产，5 号消费者被阻塞进程。3 号消费者第二次消费产品 1 后，产品 1 所占缓存区间释放。2 号生产者开始生产产品 2，5 号消费者消费 2 号产品，并释放 2 号产品的缓存区间。至此，模拟程序运行完毕。

“测试用例文件 2”实验结果：



```
输入文件是:
2
thread 1    P    2.000000
thread 2    C    1.000000    3    1    4
thread 3    P    4.000000
thread 4    P    3.000000


Consumer  2 request to consume  3 product
Producer  1 sends the produce require.
Producer  1 begin  to produce at position  0.
Producer  1 finish producing :
 position[  0 ]:  1
Producer  4 sends the produce require.
Producer  4 begin  to produce at position  1.
Producer  4 finish producing :
 position[  1 ]:  4
Producer  3 sends the produce require.
```

结果解释：

2 号消费者开始请求 3 号产品，由于产品不存在，因此本线程被阻塞。生产 1 开始生产产品，而 2 号消费者仍处于阻塞状态，4 号成产者开始生产产品，2 号依旧被阻塞，而由于缓存区间只有 2，因此生产者 3 不可进行产品生产，产生死锁，故无法运行出结果。

对于输入文件格式问题：

注意字符之间的空格是 Tab 键，每一行的结尾不能有任何字符，包括空格符，否则读入文件会产生错误使程序不能正确模拟整个过程。

编写过程出现的问题：

由于基础理论知识不扎实并不能顺利编写出程序，后面通过老师给出的代码和结合网上资料最后完成了这次实验。刚开始由于一些函数结构不完整跟变量未定义好未能成功运行，后面通过修改调试得出了运行结果。