实验一 生产者和消费者问题

一、实验目的

①掌握基本的同步互斥算法，理解生产者和消费者模型。

②了解 Windows 7 中多线程的并发执行机制，线程间的同步和互斥。

③学习使用 Windows 7 中基本的同步对象，掌握相应的 API。

二、实验内容

文件的格式和含义如下：

3

1 P 3

2 P 4

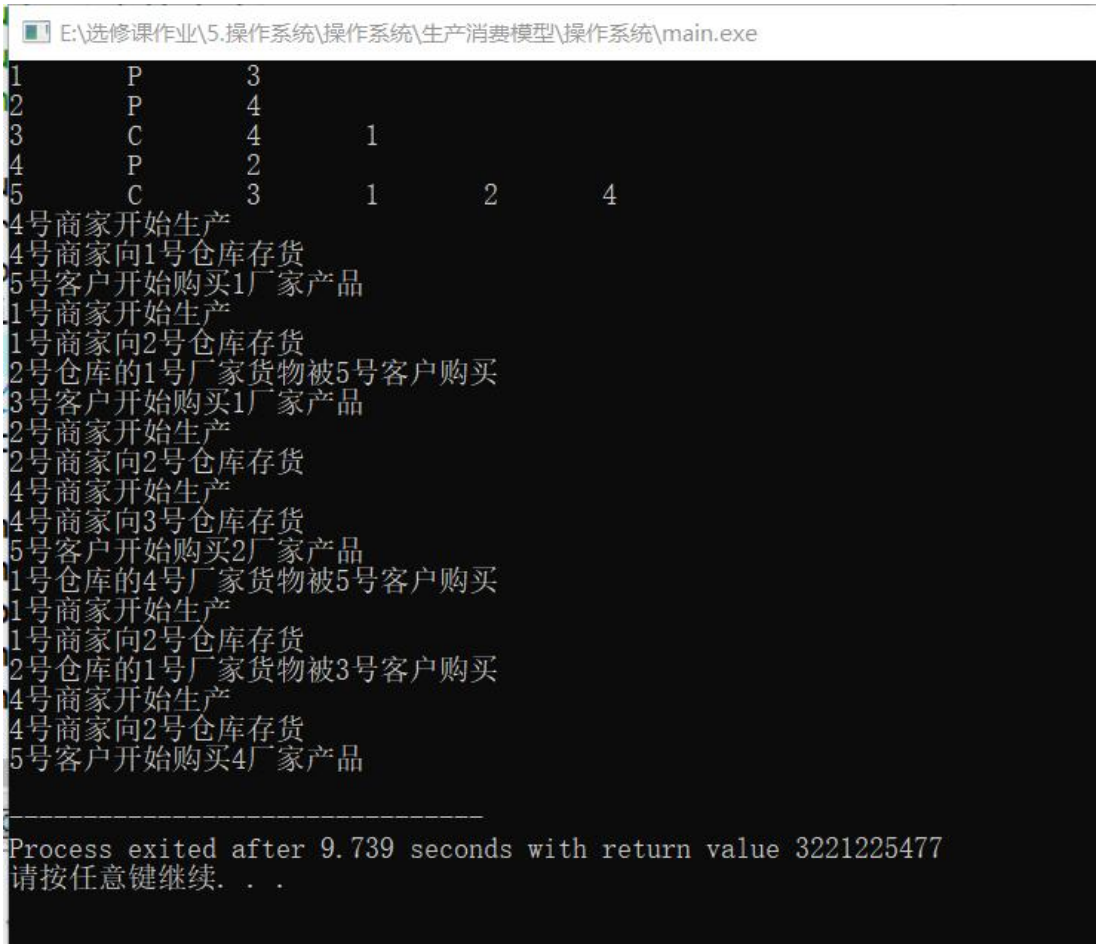
3 C 4 1

4 P 2

5 C 3 1 2 4

第一行说明程序中设置几个临界区，其余每行分别描述了一个生产者或者消费者线程的信息。每一行的各字段间用 Tab 键隔开。不管是生产者还是消费者，都有一个对应的线程号，即每一行开始字段那个整数。第二个字段用字母 P 或者 C 区分是生产者还是消费者。第三个字段表示在进入相应线程后，在进行生产和消费动作前的休眠时间，以秒计时；这样做的目的是可以通过调整这一列参数，控制开始进行生产和消费动作的时间。如果是代表生产者，则该行只有三个字段。如果代表消费者，则该行后面还要若干字段，代表要求消费的产

品所对应的生产者的线程号。所以务必确认这些对应线程号存在并且该线程代表一个生产者。



下面来看一个例子，测试用例文件如下：

3

1 P 5

2 P 4

3 P 2

4 C 3 1 3 2

由于我们在一个循环中创建了这四个线程，所以可以近似地认为它们是同时开始运转的。基于这样的假设，再观察第三列的时间参数，可以发现，最早动作的应该是 3 号生产者线程。这时它在 3 个缓

冲区之一中生产了产品。接下来应该是四号消费者，其开始请求 1 号生产者的产品，由于该产品还不存在，故本线程被阻塞。接下来是 2 号生产者生产产品，而 4 号消费者依然处于阻塞状态。等待第 5 秒钟 1 号生产者完成生产后，4 号消费者被激活，由于此时所要求的 1，3，2 号生产者的产品都已就绪，所以 4 号消费者即依次进行 3 个产品的消费。由于这里对每一个产品的第一次消费也是对它的最后一次消费，所以每消费一个产品后随即释放该产品所占缓冲区空间。至此，模拟程序成功运行完毕。

```
#define    _CRT_SECURE_NO_WARNINGS
#include <Windows.h>
#include <stdlib.h>
#include <stdio.h>
#define MAX_THREAD_NUM 64
int Buffer_number = 0, in = 0, out = 0;
//Buffer_number 缓冲区大小  in 生产标记  out 消费标记
int product_number = 0;
//生产者个数
int consumer_number = 0;
//消费者个数
int sign[MAX_THREAD_NUM] = { 0 };
//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
    double   delay;                                 //线程延迟
    int   thread_request[MAX_THREAD_NUM]; //线程请求队列
    int   n_request;                                //请求个数
};
ThreadInfo   Thread_Info[MAX_THREAD_NUM];
//线程信息数组；
HANDLE    h_Thread[MAX_THREAD_NUM];
//用于存储每个线程句柄的数组；句柄是资源的唯一标识
HANDLE h_mutex;
//创建互斥量，实现生产者查询并保留缓冲区下一个空位置时进行互斥。
HANDLE h_Semaphore[MAX_THREAD_NUM];
//创建信号量，该组信号量用于表示相应产品已生产
HANDLE empty_semaphore;
//创建信号量，指示缓冲区中空位置数量，以便开始生产下一个产品。
HANDLE full_semaphore;
 //创建信号量，指示缓冲区中满位置数量，以便开始生产下一个产品。
```

```
CRITICAL_SECTION *PC_Critical;              //临界区
int* Buffer_Critical;                       //创建缓冲区
int GetInformation(char* name, int* product_number,int * consumer_number);                    //从文件读取线程信息
void producer(PVOID Info) ;     //生产过程  PVOID 相当于 void *
void consumer(PVOID Info) ;                 //消费过程
int main(int argc, char** argv)
{
    char cn[] = "test.txt";
    int i;
    Buffer_number=GetInformation(cn,&product_number,&consumer_number);                    //生产者个数
    //printf("%d\n", Buffer_number);
    Buffer_Critical = (int*)malloc(sizeof(int) * Buffer_number); //创建缓冲区
    for (i = 0; i < Buffer_number; i++) {
     Buffer_Critical[i] = 0;
    }
    PC_Critical=(CRITICAL_SECTION*)malloc(sizeof(CRITICAL_SECTION) * product_number);//初始化临界区
    for(i = 0; i < product_number; i++)
      InitializeCriticalSection(&PC_Critical[i]);
```

```
//printf("%d\n", product_number);
//实例互斥量对象
h_mutex= CreateMutex(NULL,false,NULL);
//实例化信号量对象
for (i = 0; i < product_number+consumer_number; i++) {
 h_Semaphore[i] = CreateMutex(NULL, false, NULL);
}
empty_semaphore= CreateSemaphore(NULL, Buffer_number, Buffer_number, NULL);
full_semaphore= CreateSemaphore(NULL, 0, Buffer_number, NULL);
//创建线程
for (i = 0; i < product_number+ consumer_number; i++) {
 if(Thread_Info[i].entity=='P')
 h_Thread[i] = CreateThread(NULL, 0,(LPTHREAD_START_ROUTINE)(producer), &(Thread_Info[i]), 0, NULL);
 else
 h_Thread[i] = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)(consumer), &(Thread_Info[i]), 0, NULL);
}
```

```
        WaitForMultipleObjects(product_number
+consumer_number,h_Thread,TRUE,INFINITE);
        return 0;
    }
    //从文件读取线程信息
    int GetInformation(char* name, int* product_number, int*
consumer_number) {
        int len, n=0, Thread_number = 0;//len 记录行长度，n 用于返
回缓冲区个数
        char ch[2], buf[60];
        FILE* fp;
        if ((fp = fopen(name, "r")) == NULL) {
         printf("打开失败");
        }
        fscanf(fp,"%d",&n);       //读取缓存区个数
        fgetc(fp);
        while (fgets(buf, 60, fp) != NULL) {
         int temp = 0;
         len = strlen(buf);
         buf[len - 1] = '\0';
         ch[0] = buf[0];
         Thread_Info[Thread_number].serial = atoi(ch);
```

```
            Thread_Info[Thread_number].entity = buf[2];
            //记录生产者消费者个数
            if (buf[2] == 'P') {
                (*product_number)++;
            }
            else {
                (*consumer_number)++;
            }
            ch[0] = buf[4];
            Thread_Info[Thread_number].delay = atof(ch);
            while (len - 5 - temp * 2 > 0) {
                ch[0] = buf[7 + temp * 2 - 1];
                Thread_Info[Thread_number].thread_request[temp]        =
atoi(ch);
                Thread_Info[Thread_number].n_request = temp;
                temp++;
            }
            Thread_number++;
            printf("%s \n", buf);
        }
        return n;
    }
```

```
void producer(PVOID Info) {
    do {
        ThreadInfo temp = *((ThreadInfo *)Info);
        Sleep(temp.delay * 1000);
        WaitForSingleObject(empty_semaphore, INFINITE);
    //程序处于等待状态，直到信号量 empty_semaphore 出现
        WaitForSingleObject(h_mutex, INFINITE);
     //程序处于等待状态，直到信号量 h_mutex 出现
        printf("%d 号商家开始生产\n", temp.serial);
        //EnterCriticalSection(PC_Critical);
    //该函数用于等待指定临界区对象的所有权。当调用线程被赋予所有权时，该函数返回。
        while (Buffer_Critical[in] != 0)
            in = (in + 1) % Buffer_number;
        printf("%d 号商家向%d 号仓库存货\n", temp.serial, in + 1);
        Buffer_Critical[in] = temp.serial;
        //LeaveCriticalSection(PC_Critical);
    //该函数释放指定临界区对象的所有权。
        ReleaseMutex(h_mutex);
//打开互斥锁
        ReleaseSemaphore(full_semaphore, 1, NULL);
     //将所指信号量加上指定大小的一个量，执行成功，则返回非 0
```

值,满位置加一

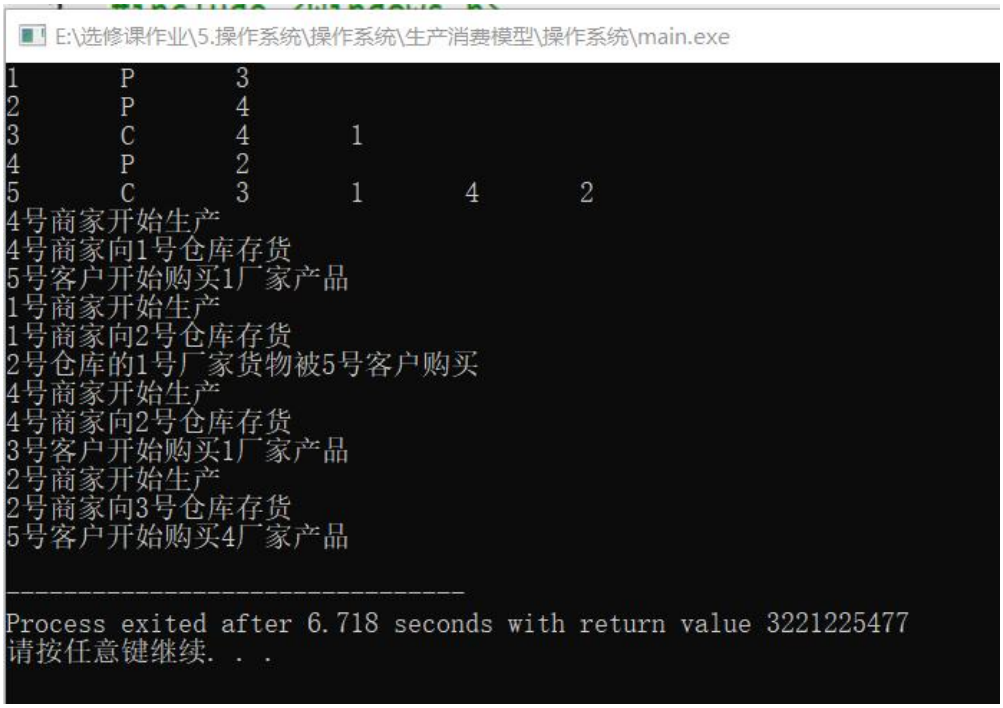
```
            ReleaseMutex(h_Semaphore[temp.serial]);
        //通知消费者已经生产好了
        } while (TRUE);
    }
    void consumer(PVOID Info) {
        ThreadInfo temp = *((ThreadInfo*)Info);
        do {
         if (temp.n_request < 0)   //如果没有请求个数就终止线程
            break;
         for (int i = 0; i < temp.n_request; i++) {
            Sleep(temp.delay * 1000);
            WaitForSingleObject(full_semaphore, INFINITE);
    //等待满位置没有到达上限
            WaitForSingleObject(h_Semaphore[temp.serial],
INFINITE);//等待生产者的通知
            printf("%d 号客户开始购买%d 厂家产品\n", temp.serial,
temp.thread_request[i]);
        EnterCriticalSection(&PC_Critical[temp.thread_request[i]-1]);
//该函数用于等待指定临界区对象的所有权。当调用线程被赋予所有
权时，该函数返回。
            while (temp.thread_request[i] != Buffer_Critical[out])   //
```

在缓冲区寻找自己需要的指定商家的产品

```
                out = (out + 1) % Buffer_number;
                printf("%d 号仓库的%d 号厂家货物被%d 号客户购买\n",
out + 1, Buffer_Critical[out], temp.serial);
                Buffer_Critical[out] = 0;
                LeaveCriticalSection(&PC_Critical[temp.thread_request[i]
- 1]);
                ReleaseSemaphore(empty_semaphore, 1, NULL);
            }
            break;
        } while (TRUE);
    }
```

“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