生产者—消费者问题

目录

班级：2015211314

学号：2015211527

姓名：罗暄澍

一、实验环境

本实验使用 Win32 环境，由 C++语言实现。故采用如下环境

操作系统	Windows 10 16299.98
编译环境	GCC v6.3.0

二、程序设计思路方法

1、同步与互斥

进程互斥是进程之间发生的一种间接性作用，一般是程序不希望的。通常的情况是两个或两个以上的进程需要同时访问某个共享变量。我们一般将发生能够问共享变量的程序段成为临界区。两个进程不能同时进入临界区，否则就会导致数据的不一致，产生与时间有关的错误。解决互斥问题应该满足互斥和公平两个原则，即任意时刻只能允许一个进程处于同一共享变量的临界区，而且不能让任何进程无限期地等待。互斥问题可以用硬件方法解决；也可以用软件方法。

同步是指在互斥的基础上（大多数情况），通过其它机制实现访问者对资源的有序访问。在大多数情况下，同步已经实现了互斥，特别是所有写入资源的情况必定是互斥的。少数情况是指可以允许多个访问者同时访问资源。

2、设计方法

(1) 主要利用 windows 标准库中自带互斥锁和信号量,实现线程互斥。并对生产者和消费者线程增加,可多个消费者和生产者的给功能,及开多个相同功能线程。

- **Produce 线程**
  随机的时间段内生产产品，并在互斥允许以及缓冲区有空间的时候，加入缓冲区，将产品编号存入缓冲区。
- **Consume 线程**
  随机时间段内消费产品，并在互斥允许以及缓冲区有产品的时候，消费产品，将缓冲区对应位置赋零。

(2) 在主进程里输入参数设置仓库大小，请求个数及各自类型和时间。读入参数后，完成仓库等的初始化，之后从 0 开始计时按时间顺序启动线程。对于每个线程，输出相应的信息告知我们它开始提出请求，它获得允许开始执行相应操作，它结束操作释放相应使用权，线程结束。

三、数据结构

- 用数组模拟循环队列的缓冲区来表示缓冲区，BUFFER_SIZE 标记缓冲区大小，in, out 分别表示上下界。

- 每个对缓冲区的请求需要提供请求者类型和请求时间这两项内容，所以定义 request 结构如下：

```
struct request
{
    int p_c;    //请求者类型
    int ti;     //请求时间,单位 ms
} req[MAX_REQ];//请求序列
```

四、测试数据集

1、如果全是生产者

Size of storage	1
Number of request	4

Request：

Producer / Consumer	Time
Producer	2
Producer	3
Producer	4
Producer	5

2、如果全是消费者

Size of storage	1
Number of request	4

Request：

Producer / Consumer	Time
Consumer	2
Consumer	3
Consumer	4
Consumer	5

3、生产者少于消费者且请求较慢

Size of storage	3
Number of request	3

Request：

Producer / Consumer	Time
Consumer	2
Consumer	3
Producer	100

4、综合情况

Size of storage	2
Number of request	8

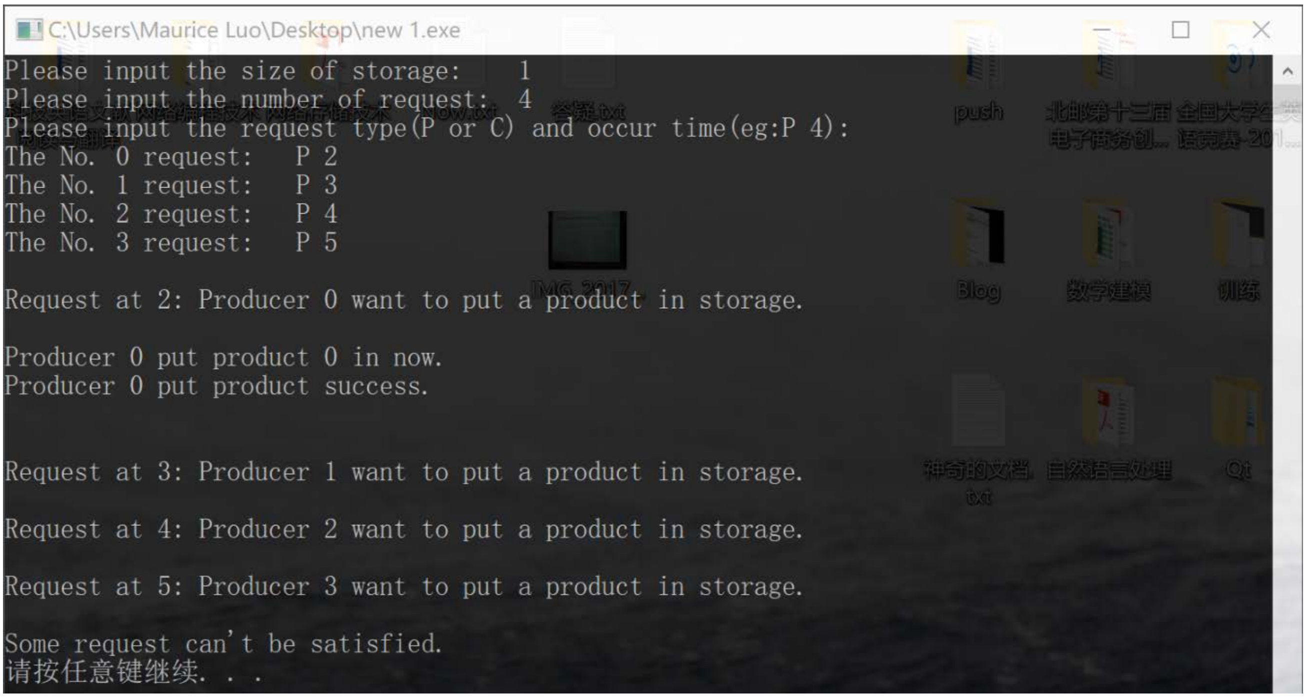
Request：

Producer / Consumer	Time
Producer	2
Producer	3
Producer	4
Consumer	100
Consumer	101
Consumer	102

Consumer	103
Producer	200

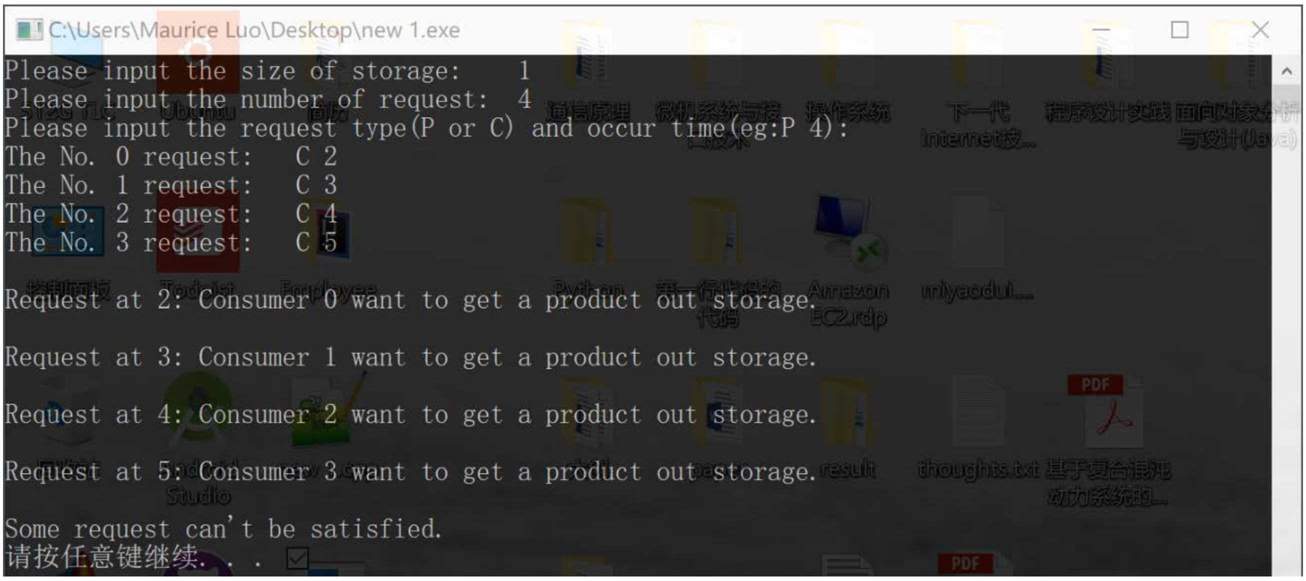
五、运行结果截屏

1、如果全是生产者



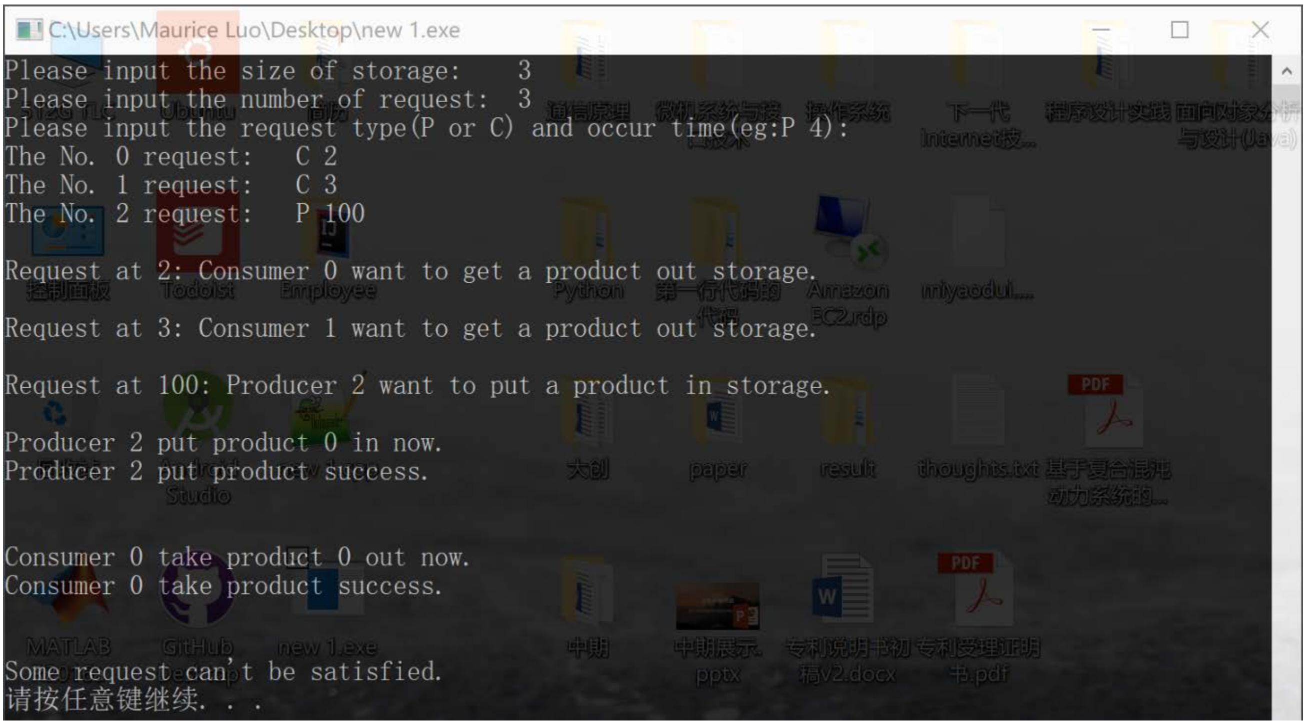
```
Please input the size of storage:    1
Please input the number of request:  4
Please input the request type(P or C) and occur time(eg:P 4):
The No. 0 request:   P 2
The No. 1 request:   P 3
The No. 2 request:   P 4
The No. 3 request:   P 5

Request at 2: Producer 0 want to put a product in storage.

Producer 0 put product 0 in now.
Producer 0 put product success.


Request at 3: Producer 1 want to put a product in storage.

Request at 4: Producer 2 want to put a product in storage.

Request at 5: Producer 3 want to put a product in storage.

Some request can't be satisfied.
请按任意键继续. . .
```

2、如果全是消费者



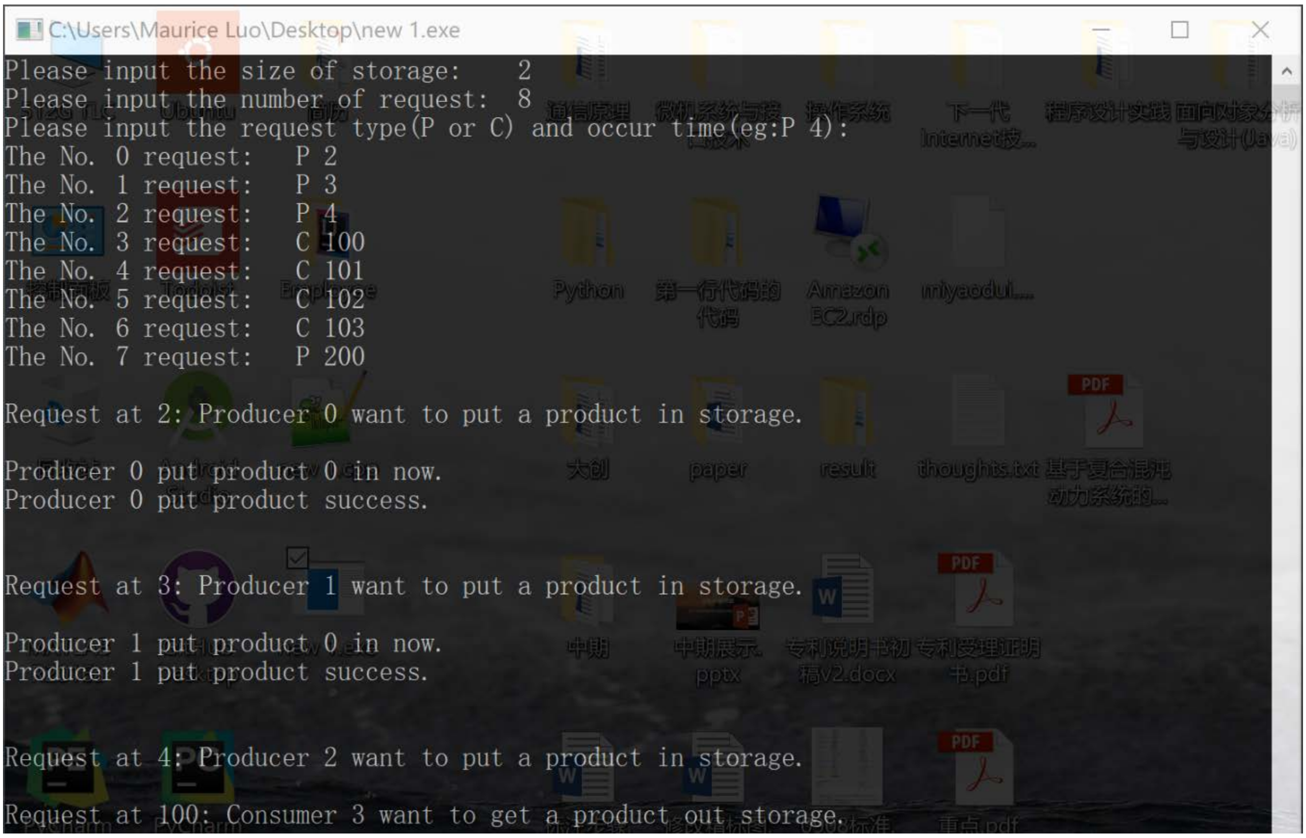
```
Please input the size of storage:    1
Please input the number of request:  4
Please input the request type(P or C) and occur time(eg:P 4):
The No. 0 request:   C 2
The No. 1 request:   C 3
The No. 2 request:   C 4
The No. 3 request:   C 5

Request at 2: Consumer 0 want to get a product out storage.

Request at 3: Consumer 1 want to get a product out storage.

Request at 4: Consumer 2 want to get a product out storage.

Request at 5: Consumer 3 want to get a product out storage.

Some request can't be satisfied.
请按任意键继续. . .
```

3、生产者少于消费者且请求较慢

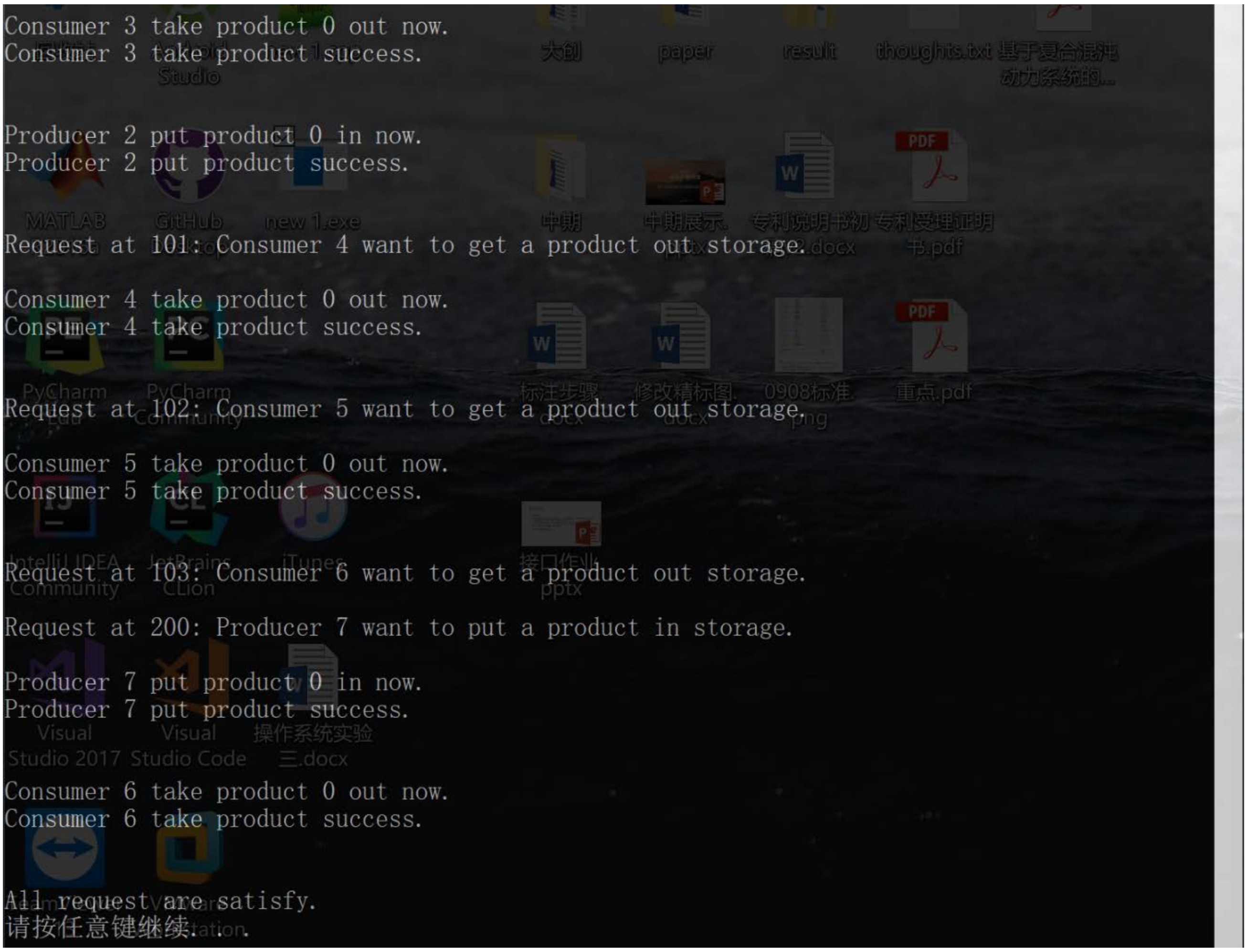


```
Please input the size of storage:    3
Please input the number of request:  3
Please input the request type(P or C) and occur time(eg:P 4):
The No. 0 request:   C 2
The No. 1 request:   C 3
The No. 2 request:   P 100

Request at 2: Consumer 0 want to get a product out storage.

Request at 3: Consumer 1 want to get a product out storage.

Request at 100: Producer 2 want to put a product in storage.

Producer 2 put product 0 in now.
Producer 2 put product success.


Consumer 0 take product 0 out now.
Consumer 0 take product success.


Some request can't be satisfied.
请按任意键继续. . .
```

4、综合情况



```
Please input the size of storage:    2
Please input the number of request:  8
Please input the request type(P or C) and occur time(eg:P 4):
The No. 0 request:   P 2
The No. 1 request:   P 3
The No. 2 request:   P 4
The No. 3 request:   C 100
The No. 4 request:   C 101
The No. 5 request:   C 102
The No. 6 request:   C 103
The No. 7 request:   P 200

Request at 2: Producer 0 want to put a product in storage.

Producer 0 put product 0 in now.
Producer 0 put product success.


Request at 3: Producer 1 want to put a product in storage.

Producer 1 put product 0 in now.
Producer 1 put product success.


Request at 4: Producer 2 want to put a product in storage.

Request at 100: Consumer 3 want to get a product out storage.
```



```
Consumer 3 take product 0 out now.
Consumer 3 take product success.


Producer 2 put product 0 in now.
Producer 2 put product success.


Request at 101: Consumer 4 want to get a product out storage.

Consumer 4 take product 0 out now.
Consumer 4 take product success.


Request at 102: Consumer 5 want to get a product out storage.

Consumer 5 take product 0 out now.
Consumer 5 take product success.


Request at 103: Consumer 6 want to get a product out storage.

Request at 200: Producer 7 want to put a product in storage.

Producer 7 put product 0 in now.
Producer 7 put product success.

Consumer 6 take product 0 out now.
Consumer 6 take product success.


All request are satisfy.
请按任意键继续. . .
```

六、源代码

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <algorithm>
#include <windows.h>

using namespace std;
const int MAX_BUF = 1024;   //最大缓冲区大小
const int MAX_REQ = 20;     //最大请求数
const int P = 1;            //生产者
const int C = 0;            //消费者

int BUFFER_SIZE;            //缓冲区大小，即用户设定的仓库容量
int Pro_no;                 //生产的产品号，从 1 开始
int in;                     //缓冲区里产品的下界
int out;                    //缓冲区里产品的上界
int buffer[MAX_BUF];        //用数组模拟循环队列的缓冲区
int req_num;                //对仓库的操作请求数
struct request
{
    int p_c;                //请求者类型
    int ti;                 //请求时间,单位 ms
} req[MAX_REQ];             //请求序列

//定义三个信号量
HANDLE mutex;               //用于进程对仓库的互斥操作
HANDLE full_sema;           //当仓库满时生产者必须等待
HANDLE empty_sema;          //当仓库空时消费者必须等待
HANDLE thread[MAX_REQ];     //各线程的 handle
DWORD pro_id[MAX_REQ];      //生产者线程的标识符
DWORD con_id[MAX_REQ];      //消费者线程的标识符

//对请求按时间排序的比较函数
bool cmp(request a, request b){    return a.ti<b.ti; }
/*初始化函数*/
void initial()
{
    Pro_no = 1;
    in=out=0;
    memset(buffer, 0, sizeof(buffer));

    printf("Please input the size of storage:    ");//读入仓库大小，即缓冲区大小
    scanf("%d", &BUFFER_SIZE);
    printf("Please input the number of request:  ");//读入仓库操作请求个数
    scanf("%d", &req_num);
    printf("Please input the request type(P or C) and occur time(eg:P 4):\n");

//读入各个请求的类型和时间
    int i;
    char ch[3];
    for(i=0; i<req_num; i++)
    {
        printf("The No.%2d request:   ", i);
        scanf("%s %d", ch, &req[i].ti);
        if(ch[0]=='P')req[i].p_c=P;
        else req[i].p_c=C;
    }
    //将请求按时间轴排序
    sort(req, req+req_num, cmp);
}
```

```
/*****生产者线程****/
DWORD WINAPI producer(LPVOID lpPara)
{
    WaitForSingleObject(full_sema, INFINITE); //等待空位
    WaitForSingleObject(mutex, INFINITE);     //对仓库的操作权

    //跳过生产过程
    //开始放产品进入仓库
    printf("\nProducer %d put product %d in now.\n", (long long)lpPara, Pro_no);
    buffer[in]=Pro_no++;
    in=(in+1)%BUFFER_SIZE;

    Sleep(5);
    printf("Producer %d put product success.\n\n", (long long)lpPara);

    ReleaseMutex(mutex);                    //释放仓库操作权
    ReleaseSemaphore(empty_sema, 1, NULL);    //非空位加一
    return 0;
}

/****消费者线程****/
DWORD WINAPI consumer(LPVOID lpPara)
{
    WaitForSingleObject(empty_sema, INFINITE);//等待非空位
    WaitForSingleObject(mutex, INFINITE);     //对仓库的操作权
    //开始从仓库取出产品
    printf("\nConsumer %d take product %d out now.\n", (long long)lpPara, buffer[out]);
    buffer[out]=0;
    out=(out+1)%BUFFER_SIZE;

    Sleep(5);
    printf("Consumer %d take product success.\n\n", (long long)lpPara);

    //跳过消费过程
    ReleaseMutex(mutex);                       //释放对仓库的操作权
    ReleaseSemaphore(full_sema, 1, NULL);     //空位加一
    return 0;
}

int main()
{
    initial();                //初始化各变量
    //创建各个互斥信号
    mutex=CreateMutex(NULL, false, NULL);
    full_sema=CreateSemaphore(NULL, BUFFER_SIZE, BUFFER_SIZE, NULL);
    empty_sema=CreateSemaphore(NULL, 0, BUFFER_SIZE, NULL);

    int pre=0;                //上一个请求的时间
    for(int i=0; i<req_num; i++)
    {
        if(req[i].p_c==P)       //创建生产者线程
        {
            thread[i]=CreateThread(NULL, 0, producer, (LPVOID)i, 0, &pro_id[i]);
            if(thread[i]==NULL)return -1;
            printf("\nRequest at %d: Producer %d want to put a product in storage.\n", req[i].ti, i);
        }
        else                  //创建消费者线程
        {
            thread[i]=CreateThread(NULL, 0, consumer, (LPVOID)i, 0, &con_id[i]);
            if(thread[i]==NULL)return -1;
            printf("\nRequest at %d: Consumer %d want to get a product out storage.\n", req[i].ti, i);
        }

        Sleep(req[i].ti-pre);   //模拟时间
        pre=req[i].ti;
    }

    //等待所有线程结束或超时，返回请求答复结果
    int nIndex = WaitForMultipleObjects(req_num, thread, TRUE, 500);
    if (nIndex == WAIT_TIMEOUT) //超时 500 毫秒
        printf("\nSome request can't be satisfied.\n");
    else
        printf("\nAll request are satisfy.\n");

    //销毁线程和信号量，防止线程的内存泄露
    for(int i=0; i<req_num; i++)
        CloseHandle(thread[i]);
    CloseHandle(mutex);
    CloseHandle(full_sema);
    CloseHandle(empty_sema);

    system("pause");
    return 0;
}
```