操作系统实验 2：生产者——消费者问题

姓名：周嘉莹

学号：71118321

报告日期 2019/5/6

一、实验目的

通过实验，掌握Windows和Linux环境下互斥锁和信号量的实现方法，加深对临界区问题和进程同步机制的理解，同时熟悉利用Windows API和Pthread API进行多线程编程的方法。

二、实验内容

1. 在 Windows 操作系统上，利用 Win32 API 提供的信号量机制，编写应用程序实现生产者——消费者问题。

2. 在 Linux 操作系统上，利用 Pthread API 提供的信号量机制，编写应用程序实现生产者——消费者问题。

3. 两种环境下，生产者和消费者均作为独立线程，并通过 empty、full、mutex 三个信号量实现对缓冲进行插入与删除。

4. 通过打印缓冲区中的内容至屏幕，来验证应用程序的正确性。

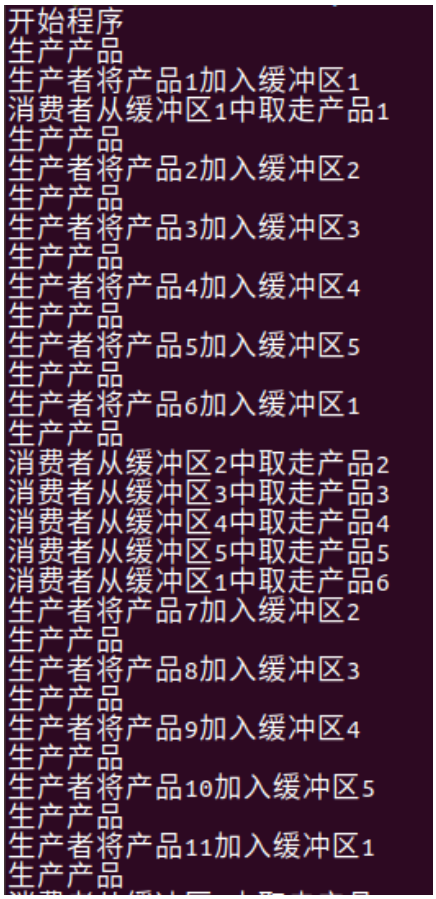
三、实验步骤

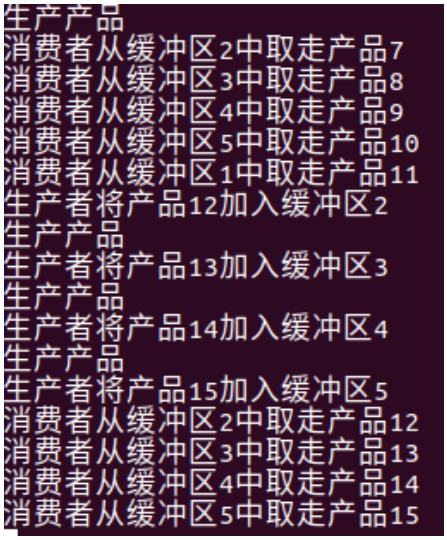
1.Linux 系统
编写 c 文件并编译实现
（1）代码

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define Max 15
sem_t empty,full,mutex;                 //定义全局同步信号量empty, full, mutex
static int Count_P=0;//用户记录生产者将产品加入缓冲区的次数
static int Count_C=0;//用户记录消费者将产品加入缓冲区的次数
static int bufferNum_P=0;//生产者的缓冲区编号
static int bufferNum_C=0;//消费者的缓冲区编号
void *producer( void *arg );    //生产者线程
void *consumer( void *arg );    //消费者线程

int main(int argc , char *argv[]){
    printf("开始程序\n");
    pthread_t thrd_prod , thrd_cons;

    sem_init (&mutex, 0, 1);    //初始化mutex
    sem_init (&empty, 0, 5);            //初始化empty信号量
    sem_init (&full, 0, 0);         //初始化full信号量
    //创建生产者和消费者线程
    if( pthread_create( &thrd_prod , NULL, producer ,
                NULL ) != 0 )
        printf( "thread create failed.\n" );

    if( pthread_create( &thrd_cons , NULL, consumer ,
                NULL ) != 0 )
        printf( "thread create failed.\n" );

    if( pthread_create( &thrd_cons , NULL, consumer ,
                NULL ) != 0 )
        printf( "thread create failed.\n" );


    //等待线程结束
    if( pthread_join( thrd_prod , NULL ) != 0 )
        printf( " wait thread failed.\n");
    if( pthread_join( thrd_cons , NULL ) != 0 )
        printf( " wait thread failed.\n");
    sem_destroy (&full);            //释放信号量
    sem_destroy(&empty);            //释放信号量
    sem_destroy(&mutex);            //释放信号量
    return 0;
}

void *producer( void *arg){
    while(Count_P<Max){
        printf("生产产品\n");
        sem_wait(&empty);  //判断缓冲区是否有剩余空间
        sem_wait(&mutex);  //等待使用缓冲区
        //成功占有互斥量，接下来可以对缓冲区进行操作
        Count_P++;
        printf( "生产者将产品%d加入缓冲区%d\n",Count_P,bufferNum_P+1);
        bufferNum_P=(bufferNum_P+1)%5;
        sem_post(&mutex); //释放缓冲区
        sem_post(&full);        //full+1
    }
}
void *consumer( void *arg ){
    while(Count_C<Max)
        {
        sem_wait(&full); //判断缓冲区是否有资源
        sem_wait(&mutex); //等待进入缓冲区
        //成功占有互斥量，接下来可以对缓冲区进行操作
        Count_C++;
        printf( "消费者从缓冲区%d中取走产品%d\n", bufferNum_C+1,Count_C);
```

```
        bufferNum_C=(bufferNum_C+1)%5;
        sem_post(&mutex);        //释放缓冲区
        sem_post(&empty);         //empty-1
    }
}
```

（2）运行结果





2.Windows 系统

编写 c++代码并编译实现

（1）代码

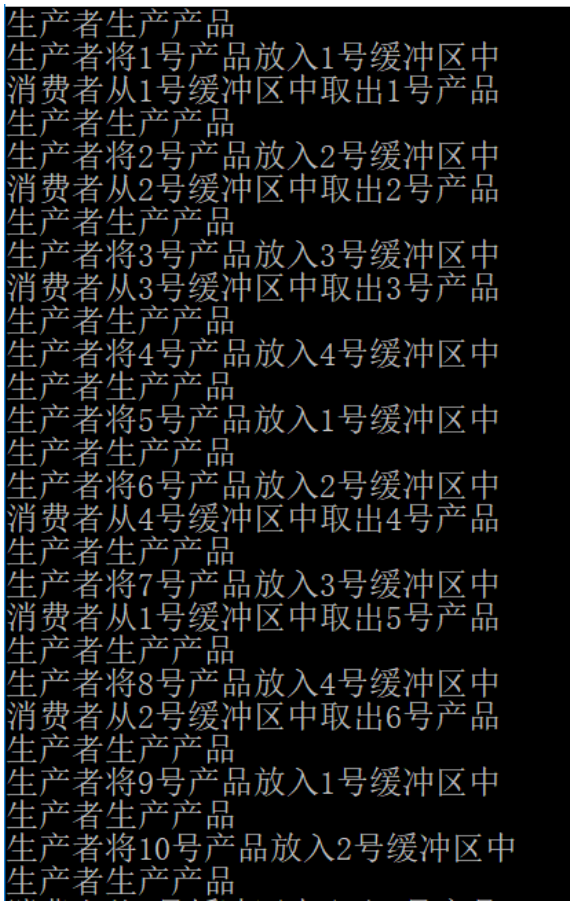
```
#include <stdio.h>
#include <process.h>
#include <windows.h>
#define Max 20
//生产者最多在缓冲区中加入15个产品
static int Count_P = 0;//用于记录生产者将产品加入缓冲区的次数
static int Count_C = 0;//用于记录消费者获取的资源个数
static int bufferNum_P = 0;//生产者的缓冲区编号
static int bufferNum_C = 0;//消费者的缓冲区编号
//信号量与关键段
HANDLE Empty, Full, Mutex;
//生产者
unsigned int __stdcall ProducerThread(PVOID pM)
{
    while (Count_P<Max)
    {
        printf("生产者生产产品\n");
        //等待有空的缓冲区出现
        WaitForSingleObject(Empty, INFINITE);
        //互斥地访问缓冲区
        WaitForSingleObject(Mutex,INFINITE);
        Count_P++;
        printf("生产者将%d号产品放入%d号缓冲区中\n",Count_P, bufferNum_P+1);
        bufferNum_P = (bufferNum_P + 1) % 4;//保证缓冲区的编号循环
```

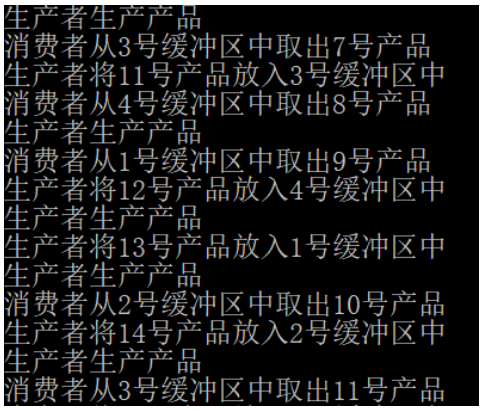
```
        //释放信号量
        ReleaseSemaphore(Mutex, 1, NULL);
        ReleaseSemaphore(Full, 1, NULL);
        Sleep(10);
    }
    printf("生产者完成任务，线程结束运行\n");
    return 0;
}
//消费者
unsigned int __stdcall ConsumerThread(PVOID pM)
{
    while (1)
    {
        //等待非空的缓冲区出现
        WaitForSingleObject(Full, INFINITE);
        WaitForSingleObject(Mutex, INFINITE);
        //互斥地访问缓冲区
        if (Count_C < Max) {//需要保证对Count_C的访问也互斥
            Count_C++;
            printf("消费者从%d号缓冲区中取出%d号产品\n", bufferNum_C + 1, Count_C);
            bufferNum_C = (bufferNum_C + 1) % 4;//保证缓冲区编号循环
            ReleaseSemaphore(Mutex, 1, NULL);
            ReleaseSemaphore(Empty, 1, NULL);
            Sleep(60);
        }
```

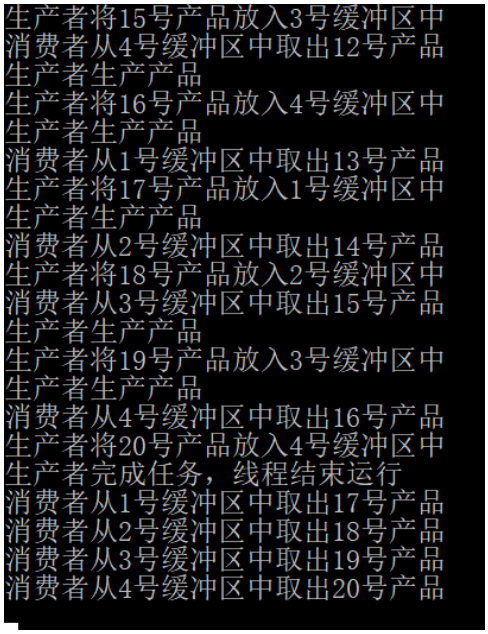
```
        }
        else
            break;
    }
    return 0;
}
int main()
{
    //初始化信号量，缓冲区大小为4
    Empty = CreateSemaphore(NULL, 4, 4, NULL);
    Full = CreateSemaphore(NULL, 0, 4, NULL);
    Mutex = CreateSemaphore(NULL, 1, 1, NULL);
    //生产者1个，消费者3个
    HANDLE hThread[4];
    hThread[0] = (HANDLE)_beginthreadex(NULL, 0, ProducerThread, NULL, 0, NULL);
    hThread[1] = (HANDLE)_beginthreadex(NULL, 0, ConsumerThread, NULL, 0, NULL);
    hThread[2] = (HANDLE)_beginthreadex(NULL, 0, ConsumerThread, NULL, 0, NULL);
    hThread[3] = (HANDLE)_beginthreadex(NULL, 0, ConsumerThread, NULL, 0, NULL);

    WaitForMultipleObjects(4, hThread, TRUE, INFINITE);//主线程等待所有生产者和消费者运行完
    for (int i = 0; i < 4; i++)
        CloseHandle(hThread[i]);
```

```
    //销毁信号量
    CloseHandle(Empty);
    CloseHandle(Full);
    return 0;
}
```

（2）运行结果







五、主要数据结构及其说明

1.linux 系统

sem_t empty,full,mutex;　　//定义全局同步信号量 empty，full，mutex 分别表示缓冲区剩余空间、剩余资源、访问缓冲区权限

static int Count_P=0;//用户记录生产者将产品加入缓冲区的次数

static int Count_C=0;//用户记录消费者将产品加入缓冲区的次数

static int bufferNum_P=0;//生产者的缓冲区编号

static int bufferNum_C=0;//消费者的缓冲区编号

编译.c 文件使用“gcc”

编译时需要在末尾加上 “-pthread”

2.Windows 系统

使用 vs2015 进行编译

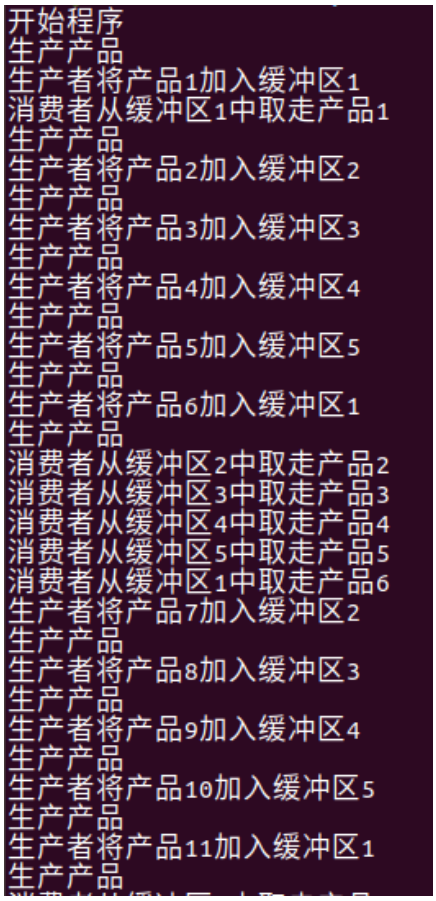
使用信号量：HANDLE Empty, Full, Mutex;分别表示缓冲区剩余空间、剩余资源、访问缓冲区权限

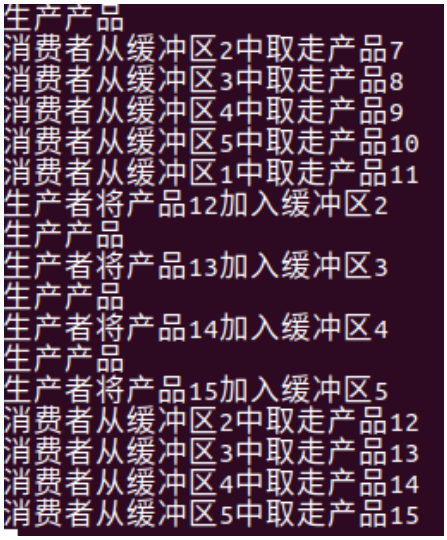
```
static int Count_P = 0;//用于记录生产者将产品加入缓冲区的次数
static int Count_C = 0;//用于记录消费者获取的资源个数
static int bufferNum_P = 0;//生产者的缓冲区编号
```

```
static int bufferNum_C = 0;//消费者的缓冲区编号
```

六、程序运行时的初值和运行结果

1.linux 系统





2.Windows 系统

```
生产者生产产品
生产者将1号产品放入1号缓冲区中
消费者从1号缓冲区中取出1号产品
生产者生产产品
生产者将2号产品放入2号缓冲区中
消费者从2号缓冲区中取出2号产品
生产者生产产品
生产者将3号产品放入3号缓冲区中
消费者从3号缓冲区中取出3号产品
生产者生产产品
生产者将4号产品放入4号缓冲区中
生产者生产产品
生产者将5号产品放入1号缓冲区中
生产者生产产品
生产者将6号产品放入2号缓冲区中
消费者从4号缓冲区中取出4号产品
生产者生产产品
生产者将7号产品放入3号缓冲区中
消费者从1号缓冲区中取出5号产品
生产者生产产品
生产者将8号产品放入4号缓冲区中
消费者从2号缓冲区中取出6号产品
生产者生产产品
生产者将9号产品放入1号缓冲区中
生产者生产产品
生产者将10号产品放入2号缓冲区中
生产者生产产品
```

```
生产者生产产品
消费者从3号缓冲区中取出7号产品
生产者将11号产品放入3号缓冲区中
消费者从4号缓冲区中取出8号产品
生产者生产产品
消费者从1号缓冲区中取出9号产品
生产者将12号产品放入4号缓冲区中
生产者生产产品
生产者将13号产品放入1号缓冲区中
生产者生产产品
消费者从2号缓冲区中取出10号产品
生产者将14号产品放入2号缓冲区中
生产者生产产品
消费者从3号缓冲区中取出11号产品
```

```
生产者将15号产品放入3号缓冲区中
消费者从4号缓冲区中取出12号产品
生产者生产产品
生产者将16号产品放入4号缓冲区中
生产者生产产品
消费者从1号缓冲区中取出13号产品
生产者将17号产品放入1号缓冲区中
生产者生产产品
消费者从2号缓冲区中取出14号产品
生产者将18号产品放入2号缓冲区中
消费者从3号缓冲区中取出15号产品
生产者生产产品
生产者将19号产品放入3号缓冲区中
生产者生产产品
消费者从4号缓冲区中取出16号产品
生产者将20号产品放入4号缓冲区中
生产者完成任务，线程结束运行
消费者从1号缓冲区中取出17号产品
消费者从2号缓冲区中取出18号产品
消费者从3号缓冲区中取出19号产品
消费者从4号缓冲区中取出20号产品
```

七、实验体会（实验中遇到的问题及解决方法、实验中产生的错误及原因分析、实验的体会及收获、对做好今后实验提出建设性建议等）

1.linux 系统编译时未在结尾加入“-pthread”，导致出现“对信号量操作以及线程操作的引用未定义”。加入后即可解决问题

```
vana@vana-VirtualBox:~/osEX2$ gcc PAndC.c
/tmp/ccbpRtUR.o：在函数‘main’中：
PAndC.c:(.text+0x3c)：对‘sem_init’未定义的引用
PAndC.c:(.text+0x52)：对‘sem_init’未定义的引用
PAndC.c:(.text+0x68)：对‘sem_init’未定义的引用
```

2.linux 在 C 编程中 printf 不加‘\n’不输出

Linux下C编程中printf不加'\n'不输出

2018年04月29日 17:56:47 Akeron 阅读数：736

调试linux下的socket程序时，发现服务器端收到的信息只有在客户端结束后才会显示收到的信息，但是如果在printf中加入换行符，就会立刻输出。原因是因为Unix系统一般有行缓存。而‘\n’可视为行刷新标志。
只要把printf（“1”）；改成printf（”1\n”）；

下面情况下会刷新缓存：
1 强制刷新标准输出缓存fflush(stdout);
2 放到缓冲区的内容中包含/n;
3 缓冲区已满;
4 需要从缓冲区拿东西到时候，如执行scanf;

3. Main 函数中需要加入 WaitForMultipleObjects(4, hThread, TRUE, INFINITE);等待所有子线程执行完毕后再继续执行 main 函数，否则主进程结束后会释放所有线程，生产者和消费者线程将不能执行

八、源程序并附上注释

1.Linux 系统

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define Max 15
sem_t empty,full,mutex;                    //定义全局同步信号量empty，full，mutex
static int Count_P=0;//用户记录生产者将产品加入缓冲区的次数
static int Count_C=0;//用户记录消费者将产品加入缓冲区的次数
static int bufferNum_P=0;//生产者的缓冲区编号
static int bufferNum_C=0;//消费者的缓冲区编号
void *producer( void *arg );    //生产者线程
void *consumer( void *arg );    //消费者线程

int main(int argc , char *argv[]){
    printf("开始程序\n");
    pthread_t thrd_prod , thrd_cons;

    sem_init (&mutex, 0, 1);    //初始化mutex
    sem_init (&empty, 0, 5);            //初始化empty信号量
    sem_init (&full, 0, 0);         //初始化full信号量
    //创建生产者和消费者线程
    if( pthread_create( &thrd_prod , NULL, producer ,
                NULL ) != 0 )
        printf( "thread create failed.\n" );

    if( pthread_create( &thrd_cons , NULL, consumer ,
                NULL ) != 0 )
        printf( "thread create failed.\n" );

    if( pthread_create( &thrd_cons , NULL, consumer ,
                NULL ) != 0 )
        printf( "thread create failed.\n" );
```

```
    //等待线程结束
    if( pthread_join( thrd_prod , NULL ) != 0 )
        printf( " wait thread failed.\n");
    if( pthread_join( thrd_cons , NULL ) != 0 )
        printf( " wait thread failed.\n");
    sem_destroy (&full);            //释放信号量
    sem_destroy(&empty);            //释放信号量
    sem_destroy(&mutex);            //释放信号量
    return 0;
}

void *producer( void *arg){
    while(Count_P<Max){
        printf("生产产品\n");
        sem_wait(&empty);  //判断缓冲区是否有剩余空间
        sem_wait(&mutex);  //等待使用缓冲区
        //成功占有互斥量，接下来可以对缓冲区进行操作
        Count_P++;
        printf( "生产者将产品%d加入缓冲区%d\n",Count_P,bufferNum_P+1);
        bufferNum_P=(bufferNum_P+1)%5;
        sem_post(&mutex); //释放缓冲区
        sem_post(&full);        //full+1
    }
}
void *consumer( void *arg ){
    while(Count_C<Max)
        {
        sem_wait(&full); //判断缓冲区是否有资源
        sem_wait(&mutex); //等待进入缓冲区
        //成功占有互斥量，接下来可以对缓冲区进行操作
        Count_C++;
        printf( "消费者从缓冲区%d中取走产品%d\n", bufferNum_C+1,Count_C);
        bufferNum_C=(bufferNum_C+1)%5;
        sem_post(&mutex);       //释放缓冲区
        sem_post(&empty);        //empty-1
    }
}
```

2.Windows 系统

```c
#include <stdio.h>
#include <process.h>
#include <windows.h>
#define Max 20
//生产者最多在缓冲区中加入15个产品
static int Count_P = 0;//用于记录生产者将产品加入缓冲区的次数
static int Count_C = 0;//用于记录消费者获取的资源个数
static int bufferNum_P = 0;//生产者的缓冲区编号
static int bufferNum_C = 0;//消费者的缓冲区编号
//信号量与关键段
HANDLE Empty, Full, Mutex;
//生产者
unsigned int __stdcall ProducerThread(PVOID pM)
{
    while (Count_P<Max)
    {
        printf("生产者生产产品\n");
        //等待有空的缓冲区出现
        WaitForSingleObject(Empty, INFINITE);
        //互斥地访问缓冲区
        WaitForSingleObject(Mutex,INFINITE);
        Count_P++;
        printf("生产者将%d号产品放入%d号缓冲区中\n",Count_P, bufferNum_P+1);
        bufferNum_P = (bufferNum_P + 1) % 4;//保证缓冲区的编号循环
        //释放信号量
        ReleaseSemaphore(Mutex, 1, NULL);
        ReleaseSemaphore(Full, 1, NULL);
        Sleep(10);
    }
    printf("生产者完成任务，线程结束运行\n");
    return 0;
}
//消费者
unsigned int __stdcall ConsumerThread(PVOID pM)
{
    while (1)
    {
        //等待非空的缓冲区出现
        WaitForSingleObject(Full, INFINITE);
        WaitForSingleObject(Mutex, INFINITE);
        //互斥地访问缓冲区
        if (Count_C < Max) {//需要保证对Count_C的访问也互斥
            Count_C++;
            printf("消费者从%d号缓冲区中取出%d号产品\n", bufferNum_C + 1, Count_C);
            bufferNum_C = (bufferNum_C + 1) % 4;//保证缓冲区编号循环
            ReleaseSemaphore(Mutex, 1, NULL);
            ReleaseSemaphore(Empty, 1, NULL);
            Sleep(60);
        }
```

```
        }
        else
            break;
    }
    return 0;
}
int main()
{
    //初始化信号量，缓冲区大小为4
    Empty = CreateSemaphore(NULL, 4, 4, NULL);
    Full = CreateSemaphore(NULL, 0, 4, NULL);
    Mutex = CreateSemaphore(NULL, 1, 1, NULL);
    //生产者1个，消费者3个
    HANDLE hThread[4];
    hThread[0] = (HANDLE)_beginthreadex(NULL, 0, ProducerThread, NULL, 0, NULL);
    hThread[1] = (HANDLE)_beginthreadex(NULL, 0, ConsumerThread, NULL, 0, NULL);
    hThread[2] = (HANDLE)_beginthreadex(NULL, 0, ConsumerThread, NULL, 0, NULL);
    hThread[3] = (HANDLE)_beginthreadex(NULL, 0, ConsumerThread, NULL, 0, NULL);

    WaitForMultipleObjects(4, hThread, TRUE, INFINITE);//主线程等待所有生产者和消费者运行完
    for (int i = 0; i < 4; i++)
        CloseHandle(hThread[i]);
```

```
    //销毁信号量
    CloseHandle(Empty);
    CloseHandle(Full);
    return 0;
}
```