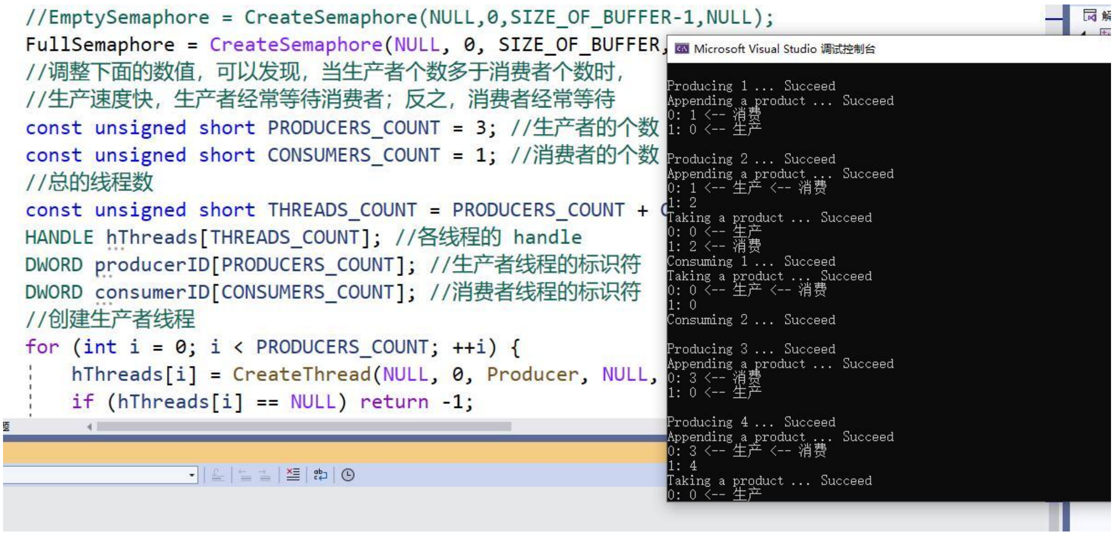
实验代码：

```
#include <windows.h>
#include <iostream>
const unsigned short SIZE_OF_BUFFER = 2; //缓冲区长度
unsigned short ProductID = 0; //产品号
unsigned short ConsumeID = 0; //将被消耗的产品号
unsigned short in = 0; //产品进缓冲区时的缓冲区下标
unsigned short out = 0; //产品出缓冲区时的缓冲区下标
int buffer[SIZE_OF_BUFFER]; //缓冲区是个循环队列
bool p_ccontinue = true; //控制程序结束
HANDLE Mutex; //用于线程间的互斥
HANDLE FullSemaphore; //当缓冲区满时迫使生产者等待
HANDLE EmptySemaphore; //当缓冲区空时迫使消费者等待
DWORD WINAPI Producer(LPVOID); //生产者线程
DWORD WINAPI Consumer(LPVOID); //消费者线程
int main()
{
    //创建各个互斥信号
   //注意，互斥信号量和同步信号量的定义方法不同，互斥信号量调用的是 CreateMutex 函数，同步信号量调用的是 CreateSemaphore 函数，函数的返回值都是句柄。
        Mutex = CreateMutex(NULL, FALSE, NULL);
          //Mutex = CreateSemaphore(NULL,1,1,NULL);// 将信号量 Mutex 完全用 CreateSemaphore 及相关函数实现
    EmptySemaphore = CreateSemaphore(NULL, SIZE_OF_BUFFER, SIZE_OF_BUFFER, NULL);
    //将上句做如下修改
    //EmptySemaphore = CreateSemaphore(NULL,0,SIZE_OF_BUFFER-1,NULL);
    FullSemaphore = CreateSemaphore(NULL, 0, SIZE_OF_BUFFER, NULL);
    //调整下面的数值，可以发现，当生产者个数多于消费者个数时，
    //生产速度快，生产者经常等待消费者；反之，消费者经常等待
    const unsigned short PRODUCERS_COUNT = 3; //生产者的个数
    const unsigned short CONSUMERS_COUNT = 1; //消费者的个数
    //总的线程数
    const unsigned short THREADS_COUNT = PRODUCERS_COUNT + CONSUMERS_COUNT;
    HANDLE hThreads[THREADS_COUNT]; //各线程的 handle
    DWORD producerID[PRODUCERS_COUNT]; //生产者线程的标识符
    DWORD consumerID[CONSUMERS_COUNT]; //消费者线程的标识符
    //创建生产者线程
    for (int i = 0; i < PRODUCERS_COUNT; ++i) {
        hThreads[i] = CreateThread(NULL, 0, Producer, NULL, 0, &producerID[i]);
        if (hThreads[i] == NULL) return -1;
    }
    //创建消费者线程
    for (int i = 0; i < CONSUMERS_COUNT; ++i) {

        hThreads[PRODUCERS_COUNT + i] = CreateThread(NULL, 0, Consumer, NULL, 0,
&consumerID[i]);
        if (hThreads[i] == NULL) return -1;
    }
    while (p_ccontinue) {
        if (getchar()) { //按回车后终止程序运行
            p_ccontinue = false;
        }
    }
    return 0;
}
//生产一个产品。简单模拟了一下，仅输出新产品的 ID 号
void Produce()
{
    std::cout << std::endl << "Producing " << ++ProductID << " ... ";
    std::cout << "Succeed" << std::endl;
}
//把新生产的产品放入缓冲区
void Append()
{
    std::cerr << "Appending a product ... ";
    buffer[in] = ProductID;
    in = (in + 1) % SIZE_OF_BUFFER;
    std::cerr << "Succeed" << std::endl;
    //输出缓冲区当前的状态
    for (int i = 0; i < SIZE_OF_BUFFER; ++i) {
        std::cout << i << ": " << buffer[i];
        if (i == in) std::cout << " <-- 生产";
        if (i == out) std::cout << " <-- 消费";
        std::cout << std::endl;
    }
}
//从缓冲区中取出一个产品
void Take()
{
    std::cerr << "Taking a product ... ";
    ConsumeID = buffer[out];
    buffer[out] = 0;
    out = (out + 1) % SIZE_OF_BUFFER;
    std::cerr << "Succeed" << std::endl;
    //输出缓冲区当前的状态
    for (int i = 0; i < SIZE_OF_BUFFER; ++i) {
        std::cout << i << ": " << buffer[i];
        if (i == in) std::cout << " <-- 生产";
        if (i == out) std::cout << " <-- 消费";
        std::cout << std::endl;
    }
}
//消耗一个产品
void Consume()
{
    std::cout << "Consuming " << ConsumeID << " ... ";
    std::cout << "Succeed" << std::endl;
}
//生产者
DWORD WINAPI Producer(LPVOID lpPara)
{
    while (p_ccontinue) {
        WaitForSingleObject(EmptySemaphore, INFINITE); //p(empty);
        WaitForSingleObject(Mutex, INFINITE); //p(mutex);
        Produce();
        Append();
        Sleep(1500);
        ReleaseMutex(Mutex); //V(mutex);
        //ReleaseSemaphore(Mutex, 1, NULL); //V(mutex);
        ReleaseSemaphore(FullSemaphore, 1, NULL); //V(full);
    }
    return 0;
}
//消费者
DWORD WINAPI Consumer(LPVOID lpPara)
{
    while (p_ccontinue) {
        WaitForSingleObject(FullSemaphore, INFINITE);//P(full);
        WaitForSingleObject(Mutex, INFINITE); //P(mutex);
        Take();
        Consume();
        Sleep(1500);
        ReleaseMutex(Mutex); //V(mutex);
        //ReleaseSemaphore(Mutex, 1, NULL); //V(mutex);
        ReleaseSemaphore(EmptySemaphore, 1, NULL); //V(empty);
    }
    return 0;
}
```

实验结果：

（1）



```
//EmptySemaphore = CreateSemaphore(NULL,0,SIZE_OF_BUFFER-1,NULL);
FullSemaphore = CreateSemaphore(NULL, 0, SIZE_OF_BUFFER,
//调整下面的数值，可以发现，当生产者个数多于消费者个数时，
//生产速度快，生产者经常等待消费者；反之，消费者经常等待
const unsigned short PRODUCERS_COUNT = 3; //生产者的个数
const unsigned short CONSUMERS_COUNT = 1; //消费者的个数
//总的线程数
const unsigned short THREADS_COUNT = PRODUCERS_COUNT + C
HANDLE hThreads[THREADS_COUNT]; //各线程的 handle
DWORD producerID[PRODUCERS_COUNT]; //生产者线程的标识符
DWORD consumerID[CONSUMERS_COUNT]; //消费者线程的标识符
//创建生产者线程
for (int i = 0; i < PRODUCERS_COUNT; ++i) {
    hThreads[i] = CreateThread(NULL, 0, Producer, NULL,
    if (hThreads[i] == NULL) return -1;
```

Microsoft Visual Studio 调试控制台

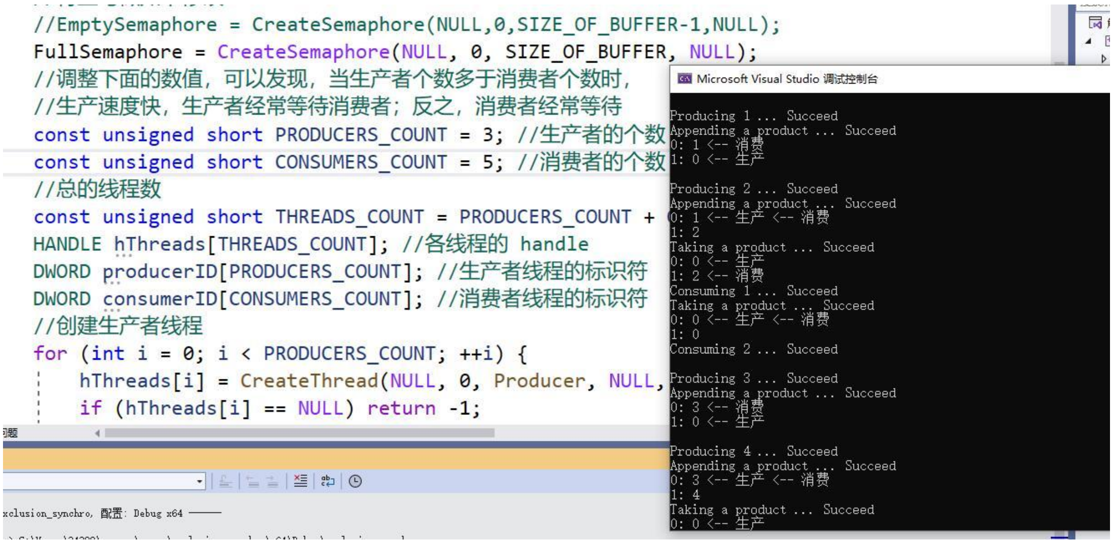
```
Producing 1 ... Succeed
Appending a product ... Succeed
0: 1 <-- 消费
1: 0 <-- 生产

Producing 2 ... Succeed
Appending a product ... Succeed
0: 1 <-- 生产 <-- 消费
1: 2
Taking a product ... Succeed
0: 0 <-- 生产
1: 2 <-- 消费
Consuming 1 ... Succeed
Taking a product ... Succeed
0: 0 <-- 生产 <-- 消费
1: 0
Consuming 2 ... Succeed

Producing 3 ... Succeed
Appending a product ... Succeed
0: 3 <-- 消费
1: 0 <-- 生产

Producing 4 ... Succeed
Appending a product ... Succeed
0: 3 <-- 生产 <-- 消费
1: 4
Taking a product ... Succeed
0: 0 <-- 生产
```

（2）修改消费者大于生产者结果



```
//EmptySemaphore = CreateSemaphore(NULL,0,SIZE_OF_BUFFER-1,NULL);
FullSemaphore = CreateSemaphore(NULL, 0, SIZE_OF_BUFFER, NULL);
//调整下面的数值，可以发现，当生产者个数多于消费者个数时，
//生产速度快，生产者经常等待消费者；反之，消费者经常等待
const unsigned short PRODUCERS_COUNT = 3; //生产者的个数
const unsigned short CONSUMERS_COUNT = 5; //消费者的个数
//总的线程数
const unsigned short THREADS_COUNT = PRODUCERS_COUNT +
HANDLE hThreads[THREADS_COUNT]; //各线程的 handle
DWORD producerID[PRODUCERS_COUNT]; //生产者线程的标识符
DWORD consumerID[CONSUMERS_COUNT]; //消费者线程的标识符
//创建生产者线程
for (int i = 0; i < PRODUCERS_COUNT; ++i) {
    hThreads[i] = CreateThread(NULL, 0, Producer, NULL,
    if (hThreads[i] == NULL) return -1;
```

xclusion_synchro, 配置: Debug x64

Microsoft Visual Studio 调试控制台

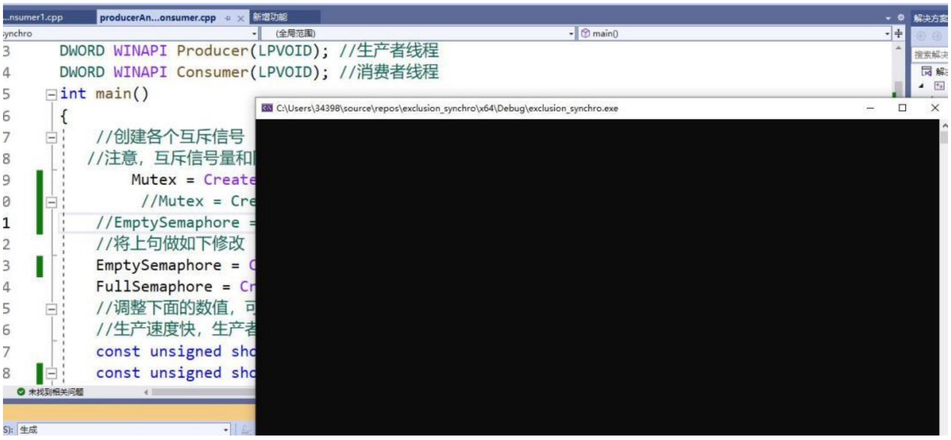
```
Producing 1 ... Succeed
Appending a product ... Succeed
0: 1 <-- 消费
1: 0 <-- 生产

Producing 2 ... Succeed
Appending a product ... Succeed
0: 1 <-- 生产 <-- 消费
1: 2
Taking a product ... Succeed
0: 0 <-- 生产
1: 2 <-- 消费
Consuming 1 ... Succeed
Taking a product ... Succeed
0: 0 <-- 生产 <-- 消费
1: 0
Consuming 2 ... Succeed

Producing 3 ... Succeed
Appending a product ... Succeed
0: 3 <-- 消费
1: 0 <-- 生产

Producing 4 ... Succeed
Appending a product ... Succeed
0: 3 <-- 生产 <-- 消费
1: 4
Taking a product ... Succeed
0: 0 <-- 生产
```

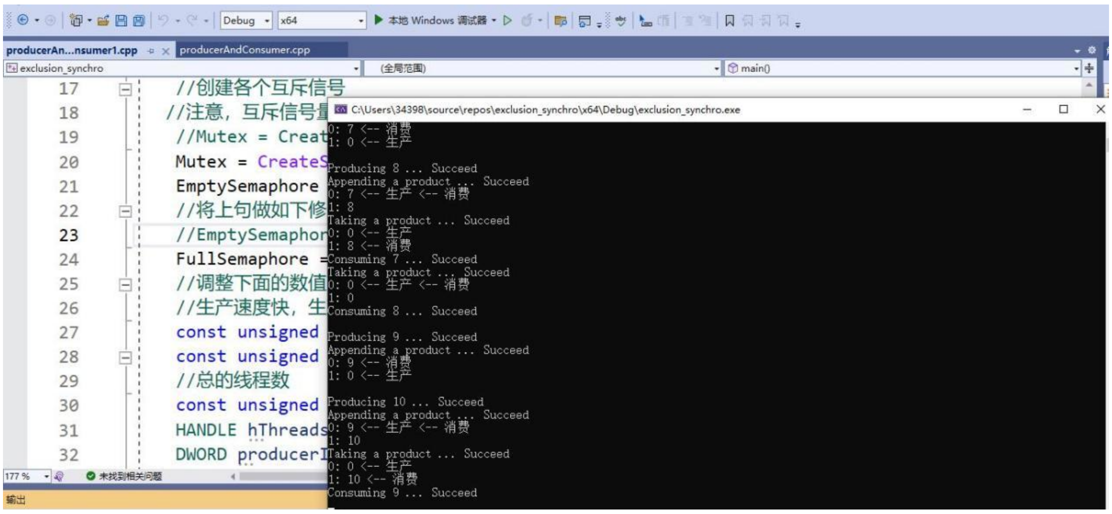
（3）修改 EmptySemaphore 后结果



```
    DWORD WINAPI Producer(LPVOID); //生产者线程
    DWORD WINAPI Consumer(LPVOID); //消费者线程
int main()
{
    //创建各个互斥信号
   //注意，互斥信号量和
        Mutex = Create
         //Mutex = Cre
    //EmptySemaphore =
    //将上句做如下修改
    EmptySemaphore = C
    FullSemaphore = Cr
    //调整下面的数值，可
    //生产速度快，生产者
    const unsigned sho
    const unsigned sho
```

C:\Users\34398\source\repos\exclusion_synchro\x64\Debug\exclusion_synchro.exe

（4）将信号量 mutex 用 CreatSemaphore 及相关函数代替



```
    //创建各个互斥信号
   //注意，互斥信号量
    //Mutex = Creat
    Mutex = CreateS
    EmptySemaphore 
    //将上句做如下修
    //EmptySemaphor
    FullSemaphore =
    //调整下面的数值
    //生产速度快，生
    const unsigned 
    const unsigned 
    //总的线程数
    const unsigned 
    HANDLE hThreads
    DWORD producerI
```

C:\Users\34398\source\repos\exclusion_synchro\x64\Debug\exclusion_synchro.exe

```
0: 7 <-- 消费
1: 0 <-- 生产

Producing 8 ... Succeed
Appending a product ... Succeed
0: 7 <-- 生产 <-- 消费
1: 8
Taking a product ... Succeed
0: 0 <-- 生产
1: 8 <-- 消费
Consuming 7 ... Succeed
Taking a product ... Succeed
0: 0 <-- 生产 <-- 消费
1: 0
Consuming 8 ... Succeed

Producing 9 ... Succeed
Appending a product ... Succeed
0: 9 <-- 消费
1: 0 <-- 生产

Producing 10 ... Succeed
Appending a product ... Succeed
0: 9 <-- 生产 <-- 消费
1: 10
Taking a product ... Succeed
0: 0 <-- 生产
1: 10 <-- 消费
Consuming 9 ... Succeed
```