**实验 5 进程的同步**

1. **实验目的**

1.使用 EOS 的信号量，编程解决生产者—消费者问题，理解进程同步的意义。

2.调试跟踪 EOS 信号量的工作过程，理解进程同步的原理。

3.修改 EOS 的信号量算法，使之支持等待超时唤醒功能（有限等待），加深理解进程同的原理。

1. **实验内容**

**2.1准备实验**

请读者按照下面方法之一在本地创建项目，用于完成本次实验：

**方法一：从 CodeCode.net 平台领取任务**

读者需要首先登录 CodeCode.net 平台领取本次实验对应的两个任务，从而在 CodeCode.net 平台上创建了两个个人项目，然后使用 OS Lab 提供的“从 Git 远程库新建项目”功能将这两个个人项目克隆到本地磁盘中。其中一个任务需要读者修改 EOS 内核程序，另一个任务需要读者修改 EOS 应用程序。

**方法二：不从 CodeCode.net 平台领取任务**

如果读者不使用 CodeCode.net 平台，需要按照下面的步骤创建项目：

1. 使用 OS Lab 提供的“从 Git 远程库新建项目”功能直接将 EOS 内核实验模板克隆到本地磁盘中，创建一个 EOS 内核项目，实验模板的 URL 为

https://www.codecode.net/engintime/os-lab/Project-Template/ eos-kernel.git。

2. 使用 OS Lab 提供的“从 Git 远程库新建项目”功能直接将 EOS 应用程序实验模板克隆到本地磁盘中，创建一个 EOS 应用程序项目，实验模板的 URL 为

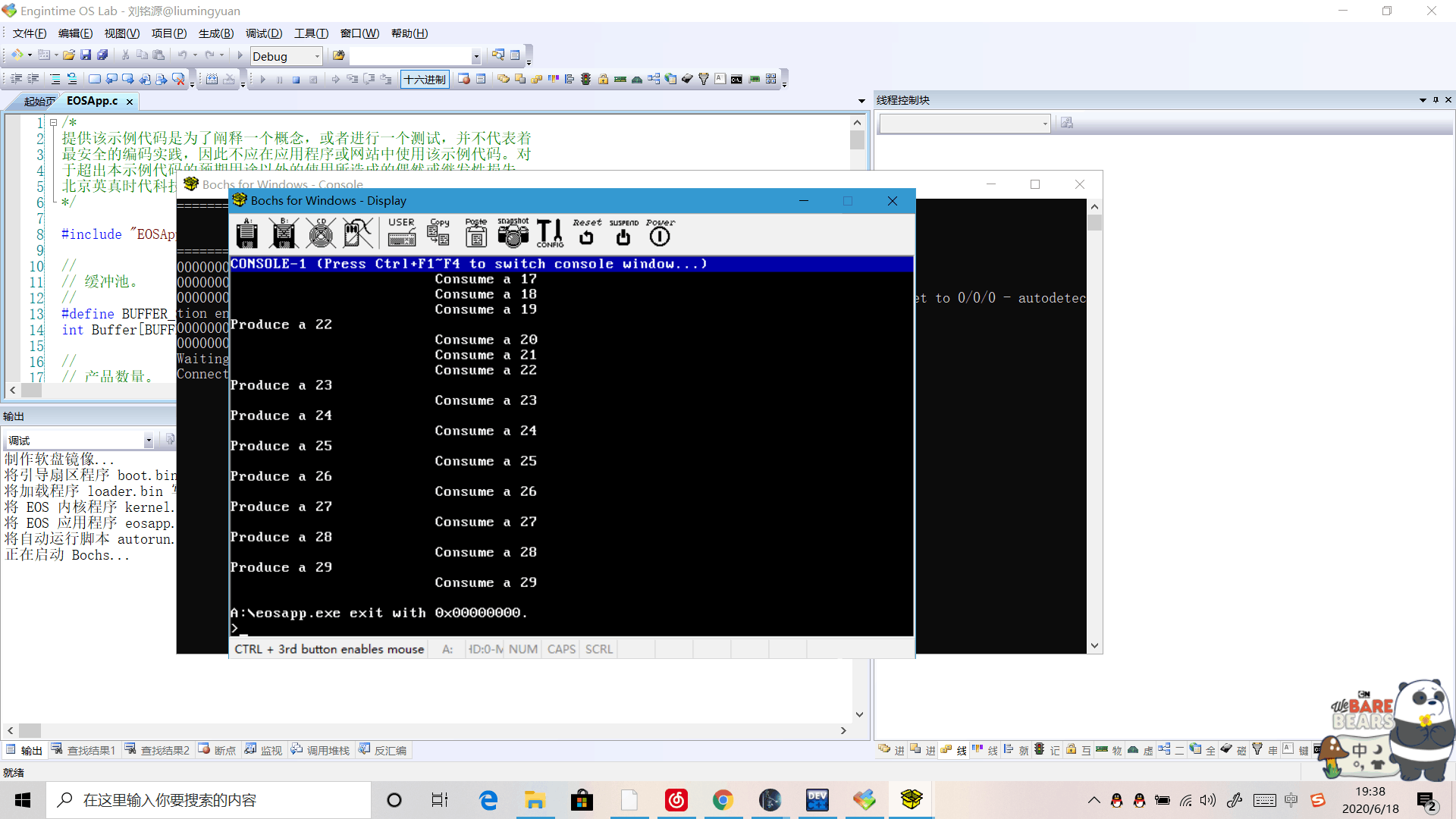
https://www.codecode.net/engintime/os-lab/Project-Template/eos-app.g

**2.2使用 EOS 的信号量解决生产者－消费者问题**

1、 通过执行指导书 3.2 的 1-4 步骤，了解到了进程的同步是通过 mutex、empty 和 full 三个信号量实现的，通过函数 wait()和 release()控制进程进入临界区；三个同步操作不能颠倒，否则可能会出现死锁现象；

2、 生产者在生产了 13 号产品后本来要继续生产 14 号产品，可此时生产者为什么必须等待消费者消 费了 4 号产品后，才能生产 14 号产品呢？生产者和消费者是怎样使用同 步对象来实现该同步过程的呢？

答：这是因为临界资源的限制。临界资源是存放产品的缓冲池，只有等到缓冲池空闲的时候生产者才能生产东西，才能有权存放产品，因此只有等到消费者消费完产品，临界资源才会空前，才能继续生产 14 号产品。

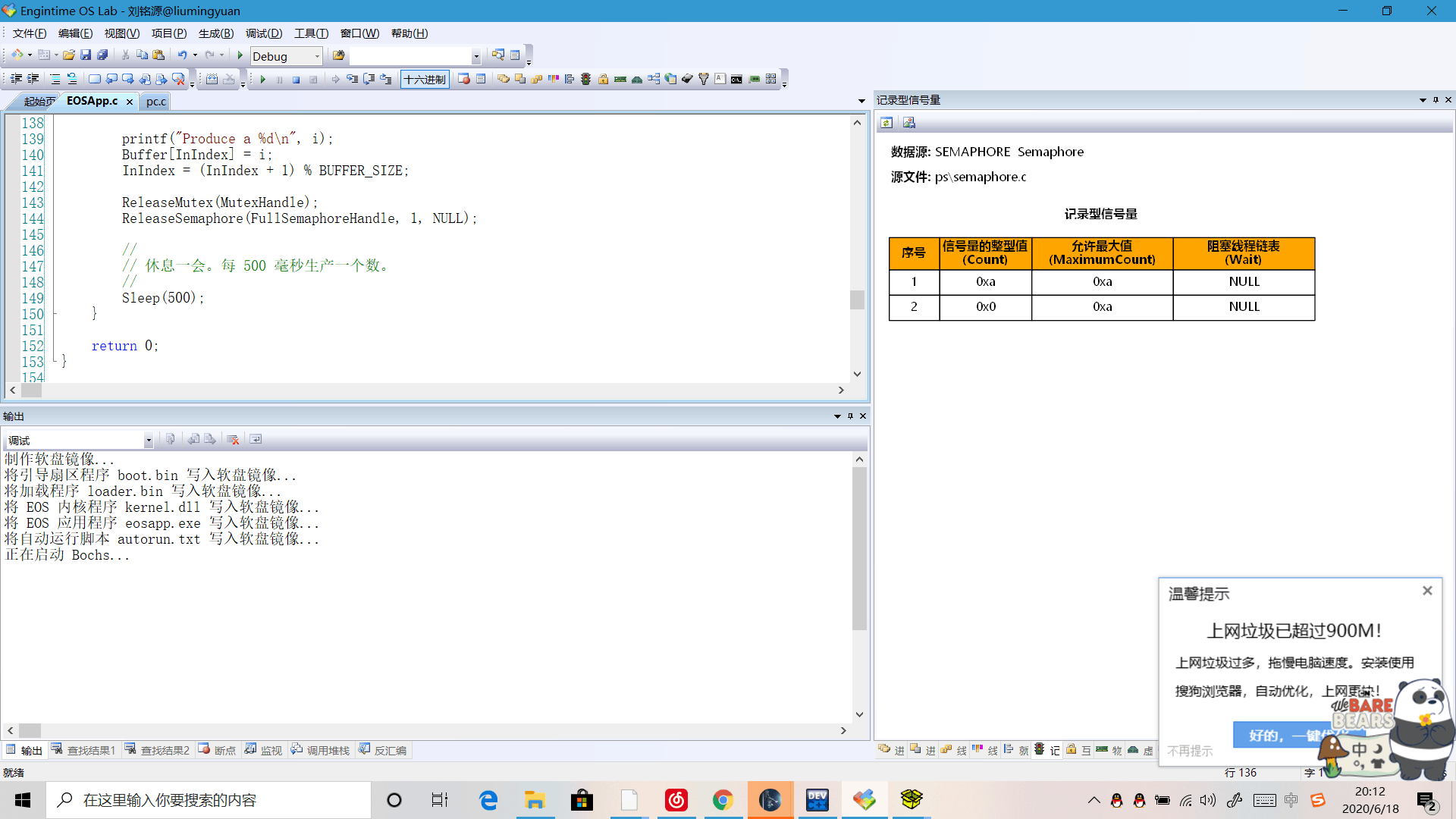
****

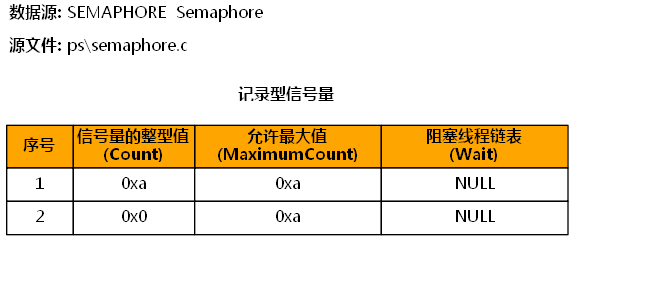
**2.3调试 EOS 信号量的工作过程**

**2.3.1创建并初始化信号量**



**创建信号量结果如下：**

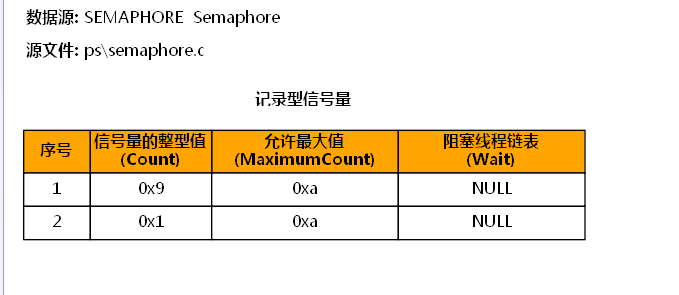
****



第一行为 empty 信号量，初始值 10，最大值 10，第二个为 full 信号量，初始值为 0，最大 值为 10，说明信号量创建成功。

**2.3.2.2 释放信号量（不唤醒）**

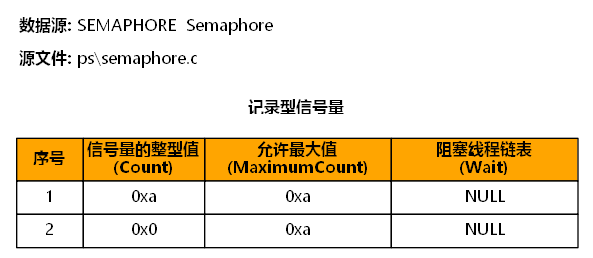
释放信号量的结果

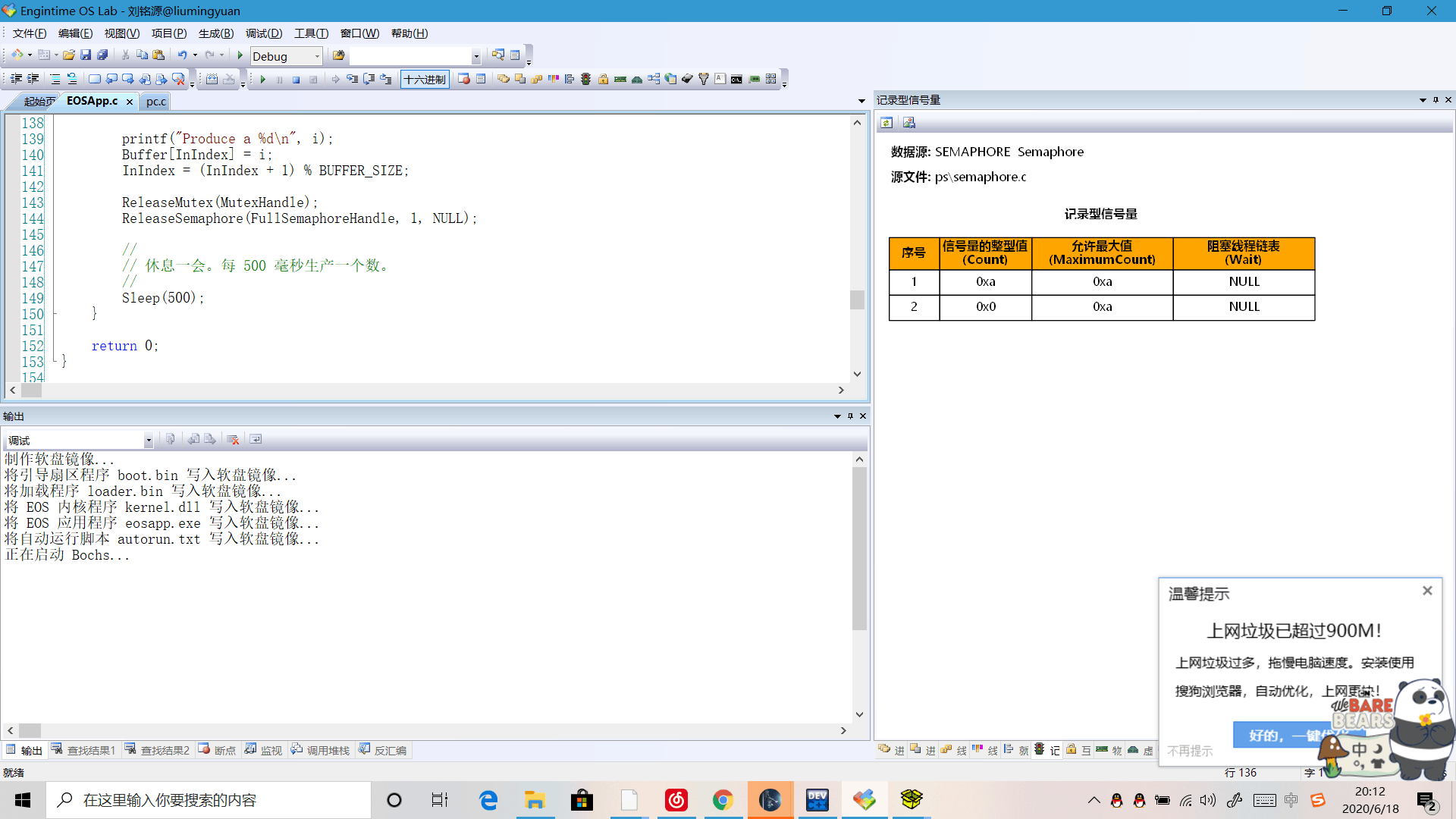


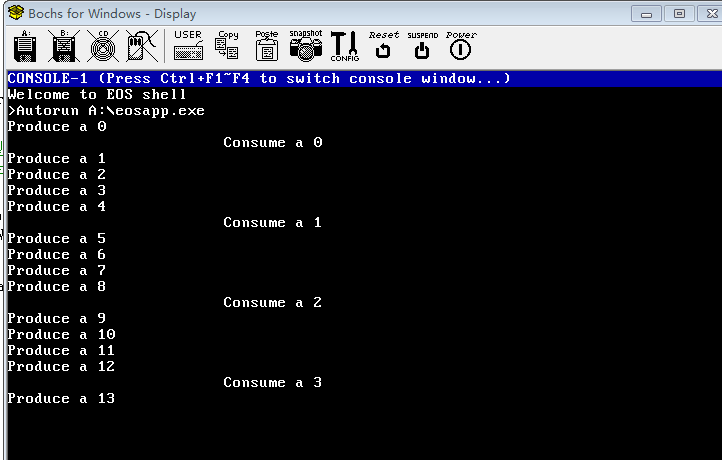
因为生产者产出了一个产品，所以 empty 释放 1，full 增加 1，说明进程的运行过程中会影响相应的信号量的值。

**2.3.2.3 等待信号量（阻塞）**

运行结果如下







**2.3.2.4 释放信号量（唤醒）**

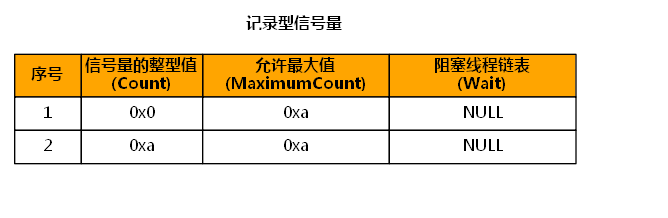
执行指导书的步骤 4 得到如下结果

消费数



执行指导书的步骤 7 得到如下结果

Empty 增至为 0



此时生产者进程被唤醒，并开始生产第 14 号（第 15 个）产品。说明信号量 empty 通过是否为负数实现对进程的同步。

**2.4 修改 EOS 的信号量算法**

修改的具体内容如下：

PsWaitForSemaphore 函数中原有的代码段 Semaphore->Count-;

If (Semaphore->Count < 0) {

PspWait(&Semaphore->WaitListHead, INFINITE);

应被修改为:先用计数值和 0 比较，当计数值大于 0 时,将计数值减 1 后直接返回成功; 当计数值等于 0 时，调用 PspWait 函数阻塞线程的执行(将参数Milliseconds 做为 PspWait 函数的第二个参数，并使用 PspWait 函数的返回值做为返回值)。

编写一个使用 ReleaseCount 做为计数器的循环体，来替换 PsReleaseSemaphore 函数中原有的代码段 Semaphore->Count++;

if (Semaphore->Count <= 0) {

PspWakeThread(&Semaphore- >WaitListHead, STATUS\_ SUCCESS);

将 Producer 函数中等待 Empty 信号量的代码行

WaitForSingleObject(EmptySemaphoreHandle, INFINITE);

替换为 while(WAIT\_ \_TIMEOUT == WaitForSingleObject(EmptySemaphoreHandle,

300)){printf("Producer wait for empty semaphore timeout\n");

将Consumer函数中等待Full 信号量的代码行WaitForSingleObject(ullemaphoreHandle,

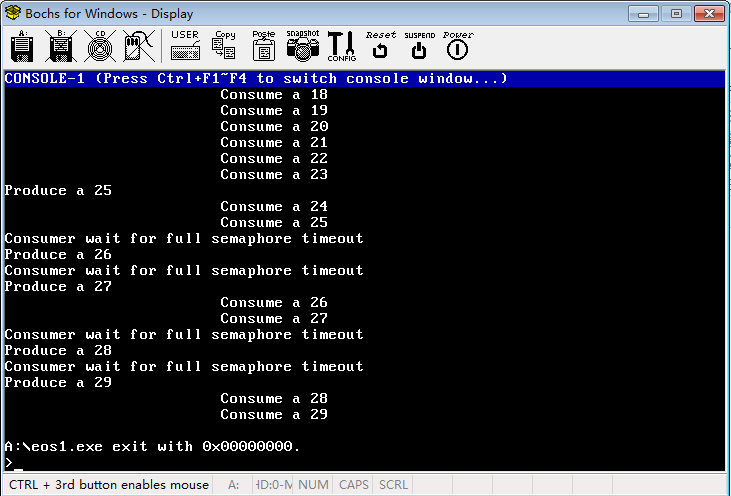
INFINITE);

替换为

while(WAIT\_ \_TIMEOUT == WaitForSingleObject(FullSemaphoreHandle,

1. ){printf("Consumer wait for full semaphore timeout\n");

执行指导书的步骤 6，一次消费两个产品的结果图如下：



说明当消费者消耗产品的速度略快时，生产者不会出现阻塞情况。

1. **思考题**

1、（思考题二）

**PsWaitForSemaphore 函数的流程图**：

①函数开始

②开始操作

③判断信号量是否小于 0 若 YES 则进行下一步，若 NO 则跳到第⑤步

④线程进入阻塞队列

⑤操作完成

⑥函数结束

**PsReleaseSemaphore 函数的流程图**

①函数开始

②开始操作

③判断 Semaphore->Count+ReleaseCount>Semaphore->Max immcount

（若 YES 则继续下一步操作 若 NO 则返回 Status=STATLS SEMPHORE LIMIT\_EXEEDED 再跳到函数结束）

④记录当前信号量的值

⑤当前信号量的值加 1

⑥判断 Semaphore>Count<=0? 若 YES 则继续下一步，若 N0 则跳到第⑦步

⑦从阻塞队列唤醒线程

⑧返回 Status=STATUS\_SECCESS

⑨函数结束

2、（思考题五）

答：申请两个资源信号量 empty 和 full，控制生产者线程和消费者线程之间的同步；只有当 empty>0 时，表示缓冲区有空闲，生产者线程可以进入临界区，每次线程结束后 empty-1；full+1。Empty<=0 时缓冲区已满，生产者线程阻塞。只有当 full>0 时，表示缓冲区内有产品，消费者可以进入临界区，每次线程结束后empty+1；full-1.full<=0 时缓冲区为空，消费者线程阻塞。

时间：2020.6.20

签名：

