

实验 5 进程的同步

一、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），加深理解进程同步的原理。

二、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：

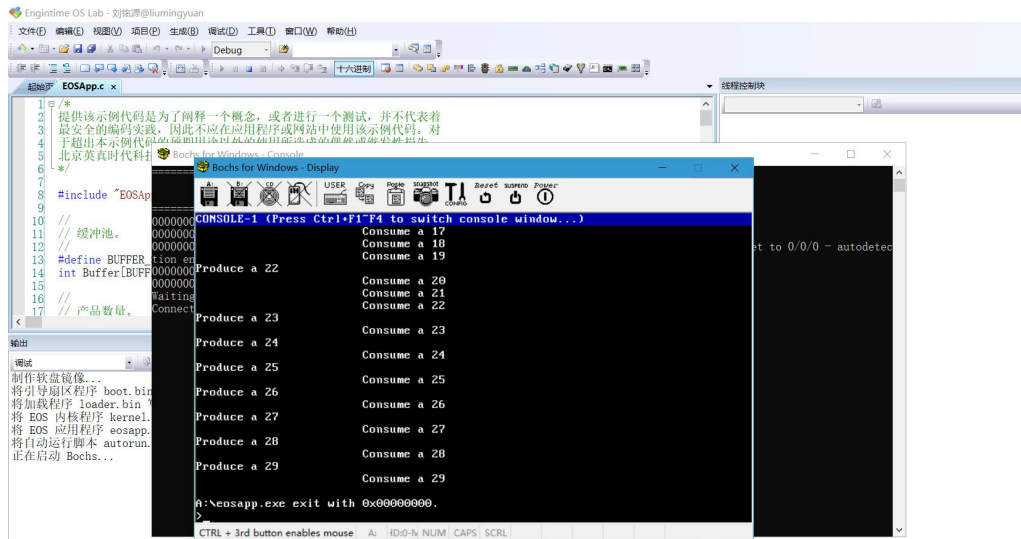
1. 使用 OS Lab 提供的“从 Git 远程库新建项目”功能直接将 EOS 内核实验模板克隆到本地磁盘中，创建一个 EOS 内核项目，实验模板的 URL 为
<https://www.codecode.net/engintime/os-lab/Project-Template/eos-kernel.git>。
2. 使用 OS Lab 提供的“从 Git 远程库新建项目”功能直接将 EOS 应用程序实验模板克隆到本地磁盘中，创建一个 EOS 应用程序项目，实验模板的 URL 为
<https://www.codecode.net/engintime/os-lab/Project-Template/eos-app.g>

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

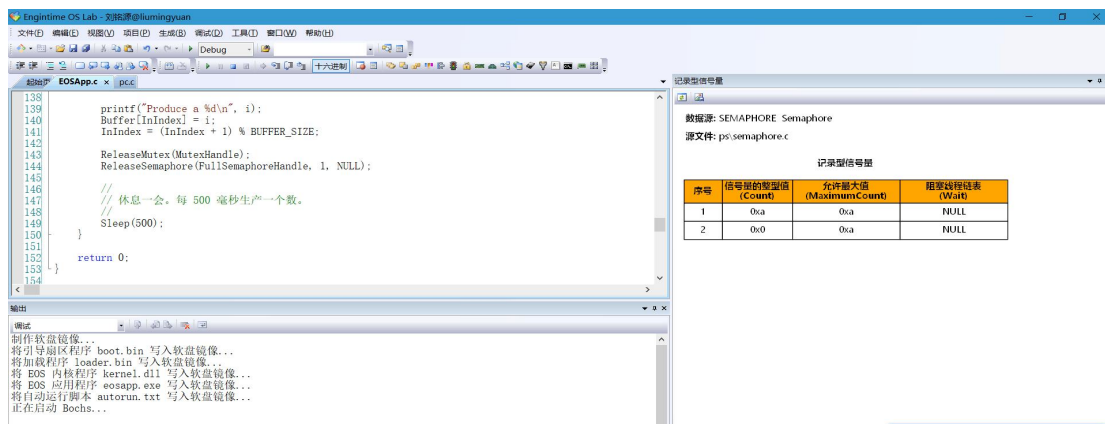
2.3.1 创建并初始化信号量

```

214 {
215     STATUS Status;
216     PVOID SemaphoreObject;
217     SEM_CREATE_PARAM CreateParam;
218
219     if (InitialCount < 0 || MaximumCount <= 0 || InitialCount > MaximumCount) {
220         return STATUS_INVALID_PARAMETER;
221     }
222
223     //
224     // 创建信号量对象。
225     //
226     CreateParam.InitialCount = InitialCount;
227     CreateParam.MaximumCount = MaximumCount;
228
229     Status = ObCreateObject( PspSemaphoreType,
230                             Name,
231                             sizeof(SEMAPHORE),
232                             (ULONG_PTR)&CreateParam,
233                             &SemaphoreObject);
234
235     if (!EOS_SUCCESS(Status)) {
236         return Status;
237     }
238
239     Status = ObCreateHandle(SemaphoreObject, SemaphoreHandle);
240
241     if (!EOS_SUCCESS(Status)) {
242         ObDereferenceObject(SemaphoreObject);
243     }
244
245     return Status;
246 }

```

创建信号量结果如下：



数据源: SEMAPHORE Semaphore

源文件: ps\semaphore.c

记录型信号量

序号	信号量的整型值 (Count)	允许最大值 (MaximumCount)	阻塞线程链表 (Wait)
1	0xa	0xa	NULL
2	0x0	0xa	NULL

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

数据源: SEMAPHORE Semaphore

源文件: ps\semaphore.c

记录型信号量

序号	信号量的整型值 (Count)	允许最大值 (MaximumCount)	阻塞线程链表 (Wait)
1	0x9	0xa	NULL
2	0x1	0xa	NULL

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）

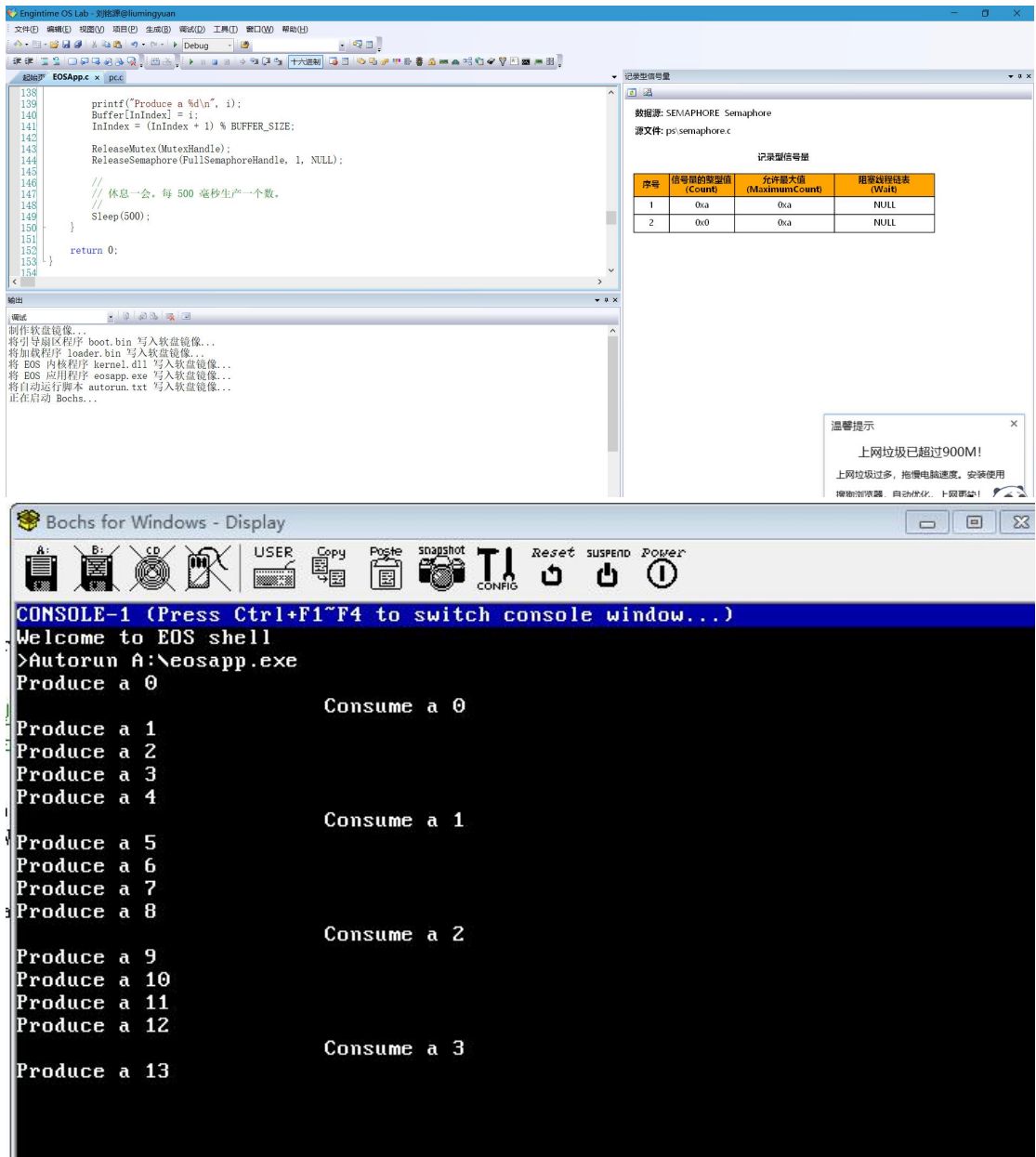
运行结果如下

数据源: SEMAPHORE Semaphore

源文件: ps\semaphore.c

记录型信号量

序号	信号量的整型值 (Count)	允许最大值 (MaximumCount)	阻塞线程链表 (Wait)
1	0xa	0xa	NULL
2	0x0	0xa	NULL



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

名称	值	类型
i	0x4	int
i	0x4	int

执行指导书的步骤 7 得到如下结果
Empty 增至为 0

记录型信号量

序号	信号量的整型值 (Count)	允许最大值 (MaximumCount)	阻塞线程链表 (Wait)
1	0x0	0xa	NULL
2	0xa	0xa	NULL

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：

PspWaitForSemaphore 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(fullSemaphoreHandle,

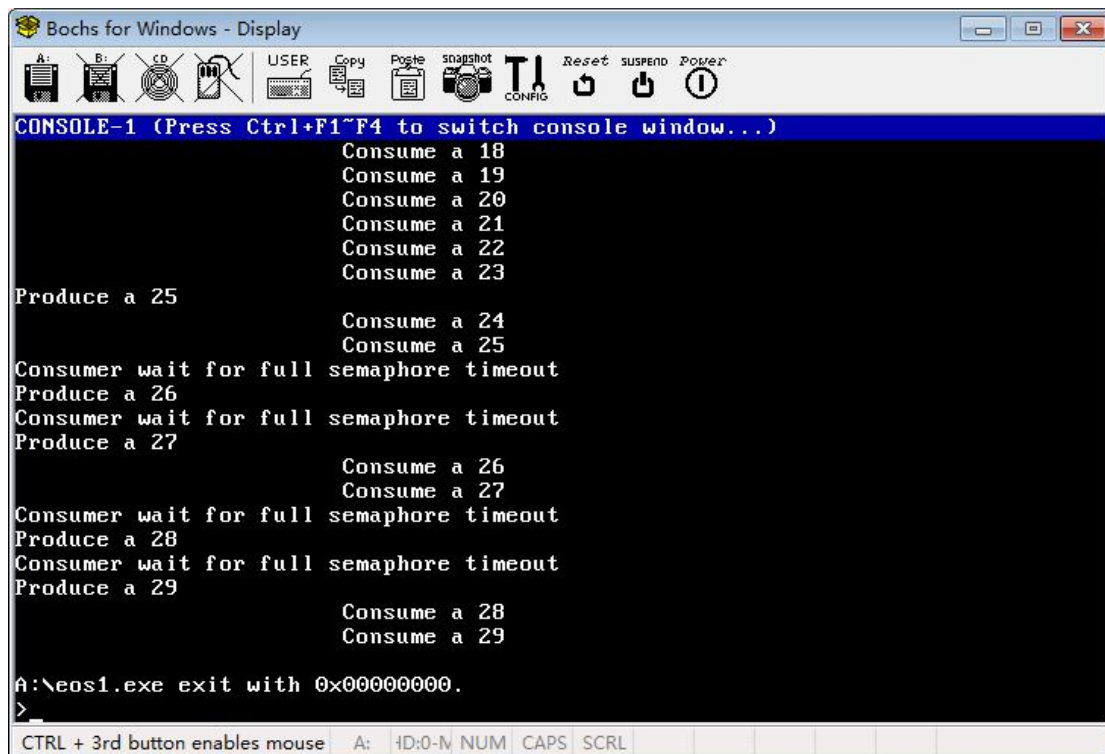
INFINITE);

替换为

while(WAIT_ _TIMEOUT == WaitForSingleObject(FullSemaphoreHandle,

300)) {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：



```
Bochs for Windows - Display
A: B: CD USER Copy Paste Snapshot T1 Reset SUSPEND POWER
CONFIG
CONSOLE-1 (Press Ctrl+F1~F4 to switch console window...)
Consume a 18
Consume a 19
Consume a 20
Consume a 21
Consume a 22
Consume a 23
Produce a 25
Consume a 24
Consume a 25
Consumer wait for full semaphore timeout
Produce a 26
Consumer wait for full semaphore timeout
Produce a 27
Consume a 26
Consume a 27
Consumer wait for full semaphore timeout
Produce a 28
Consumer wait for full semaphore timeout
Produce a 29
Consume a 28
Consume a 29
A:\eos1.exe exit with 0x00000000.
>
CTRL + 3rd button enables mouse A: 4D:0-N NUM CAPS SCRL
```

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。

三、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

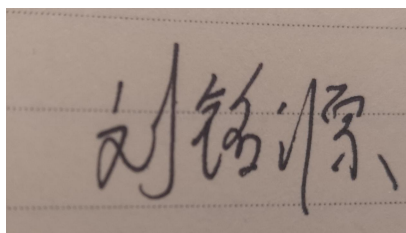
- ①函数开始
- ②开始操作
- ③判断 Semaphore->Count+ReleaseCount>Semaphore->Max immcount
(若 YES 则继续下一步操作 若 NO 则返回 Status=STATUS_SEMOPHORE
LIMIT_EXCEEDED 再跳到函数结束)
- ④记录当前信号量的值
- ⑤当前信号量的值加 1
- ⑥判断 Semaphore>Count<=0? 若 YES 则继续下一步，若 NO 则跳到第⑦步
- ⑦从阻塞队列唤醒线程
- ⑧返回 Status=STATUS_SUCCESS
- ⑨函数结束

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：

Handwritten signature in black ink on a piece of paper with horizontal lines. The signature appears to be '刘辉源' (Liu Huiyuan).