**北京科技大学实验报告**

学院： 计算机与通信工程学院 专业： 计算机科学与技术 班级： 计184

姓名： 王丹琳 学号： 41824179 实验日期： 2020 年 10 月 26 日

**实验名称：操作系统实验3 进程同步（4分）**

**实验目的：**以一个教学型操作系统EOS为例，深入理解进程（线程）同步的原理、意义及信号量的含义和实现方法；能对核心源代码进行分析和修改，能运用信号量实现同步问题；训练分析问题、解决问题以及自主学习能力，逐步达到能独立对小型操作系统的功能进行分析、设计和实现。

**实验环境：**EOS操作系统及其实验环境。

**实验内容：**

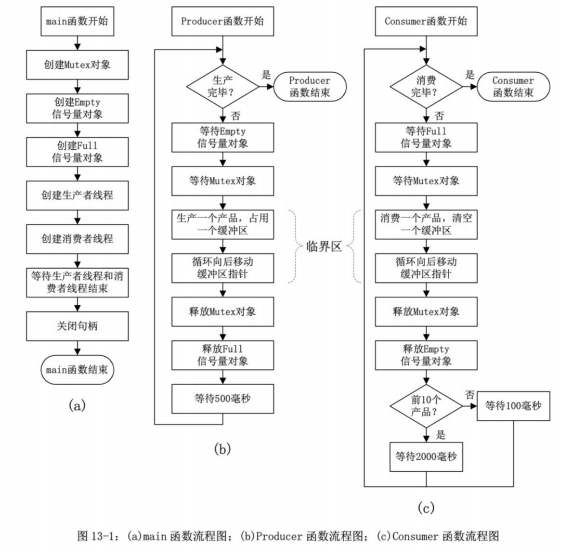
使用EOS的信号量，实现生产者-消费者问题；跟踪EOS信号量的工作过程，分析EOS信号量实现的源代码，理解并阐述其实现方法；修改EOS信号量的实现代码，使之支持等待超时唤醒和批量释放功能。

**实验步骤：**

**1）使用EOS的信号量实现生产者-消费者问题**

（给出使用EOS的信号量解决生产者-消费者问题的实现方法，包括实现方法的简要描述、源代码、测试及结果等）

1.1使用EOS的信号量解决生产者-消费者问题的实现方法



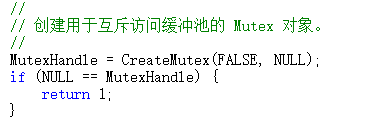
解决生产者-消费者问题需要三个信号量：empty 信号量对象和 Full 信号量对象，以及互斥信号量mutex。其中empty 信号量对象和 Full 信号量对象分别表示放产品的空缓冲区个数和已占用的缓冲区个数。初始时Empty=N，Full=0，Mutex=1，缓冲区无产品。

main 创建生产者线程和消费者线程。因为在生产者线程和消费者线程中都含有临界区，所以在访问临界资源时利用Mutex信号量实现互斥操作。

1.2源代码分析

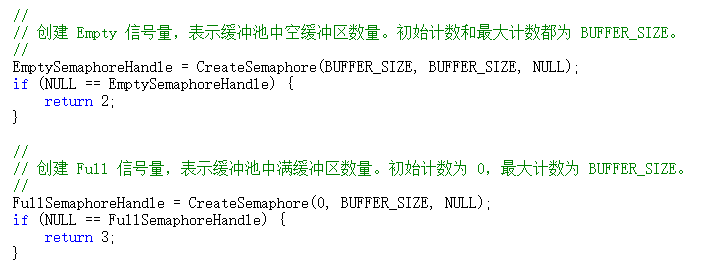
Main 函数源代码分析：

CreateMutex 创建互斥访问缓冲池的Mutex对象，实现互斥访问缓冲：



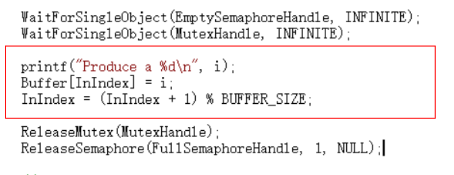
CreateSemaphore 创建信号量，并设置初值和最大计数：

创建 Empty 信号量，表示缓冲池中空缓冲区数量，初始计数和最大计数都为 BUFFER\_SIZE；创建 Full 信号量，表示缓冲池中满缓冲区数量，初始计数为 0，最大计数为 BUFFER\_SIZE：



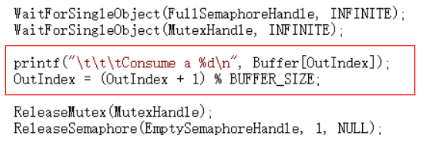
生产者线程：

生产者在每次生产时，得调用WaitForSingleObject函数判断是否有位置来存放产品，等待一个Empty信号量，之后使用Mutex实现互斥操作。完成生产后，释放Mutex信号量，并释放一个Full信号量，允许消费者进行消费。



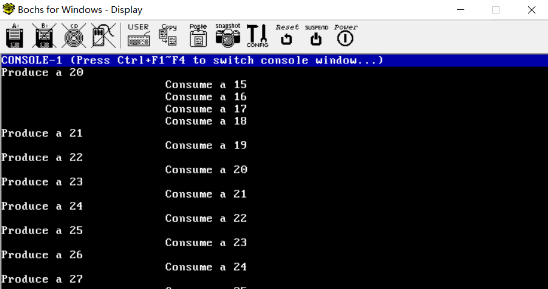
消费者进程：

消费者每次消费产品时，需要确认有产品来进行消费，因此先等待一个Full信号量，之后占用Mutex互斥操作，在完成消费后，释放Mutex信号量，然后释放一个Empty信号量，允许生产者进行生产。



测试结果：

使用 pc.c 文件中的源代码，替换之前创建的 EOS 应用程序亯目中 EOSApp.c 文件内的源代码之后运行程序：

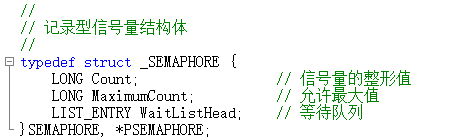


**2）EOS信号量工作过程的跟踪与源代码分析**

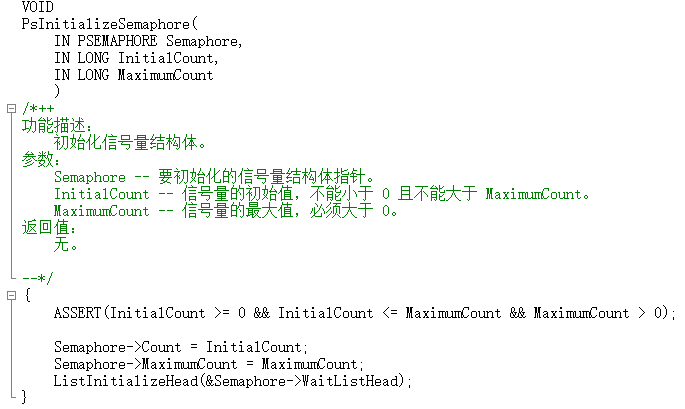
（分析EOS信号量实现的核心源代码，阐述其实现方法，包括数据结构和算法等；简要说明在本部分实验过程中完成的主要工作，包括对EOS信号量工作过程的跟踪等）

2.1.EOS信号量的数据结构与算法

\_SEMAPHORE信号量的数据结构，其中包含了信号量的值（整形）、允许的最大值和等待队列的指针

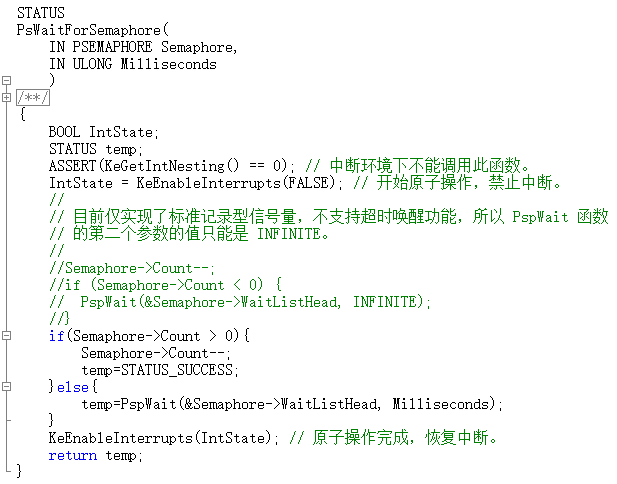


PsInitialSemaphore用来初始化信号量结构体，设置信号量结构体指针，初始值和最大值：



PsWaitForSemaphore 函数实现信号量的P操作

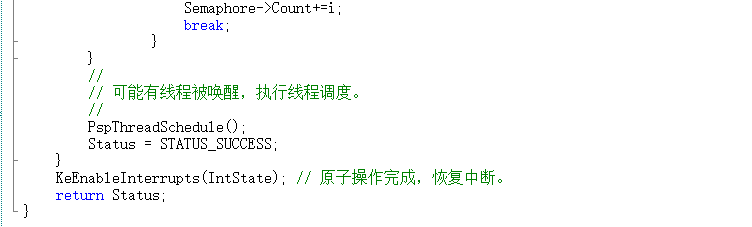
调用KeEnableInterrupts来开始原子操作，禁止中断，无法进行线程的切换。所以这里保证了下面操作的原子性，必须全部完成。在原子操作中，判断是否还有资源，若有，则信号量减一并返回“成功”，否则无资源可用，将其插入等待队列，等待信号的释放



PsReleaseSemaphore函数实现信号量的V操作

使用PsReleaseSemaphore来实现信号量的释放。与P操作一样，在访问信号量的时候要关中断，使得下面的操作成为原子操作。在原子操作中，在信号量没有超过最大值的情况下，如果为空信号量的等待队列为空，则信号量的值加一；如果不为空，则唤醒因缺资源而被阻塞的等待线程。



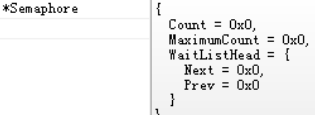


1.2.在本部分实验过程中完成的主要工作

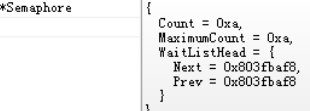
1. 使用CreateSemaphore 函数创建信号量

main函数调用CreateSemaphore的代码行加断点，进入信号量的创建过程

信号量结构体初始化之前：



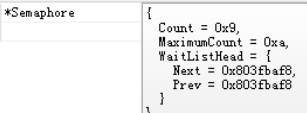
经过初始化之后，信号量已经被赋值，Count 等于 MaximumCount，等待队列为空。



2.使用WaitForSingleObject 函数等待信号量且不会阻塞

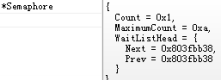
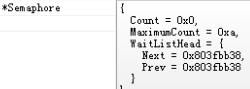
此时信号量计数大于 0 的过程：

从下图可以看出Count<MaximumCount，一部分资源被占用,但还有一些资源可以分配，所以等待信号量不会被阻塞，调用完后，信号量减一



3. 使用ReleaseSemaphore 函数释放信号量且无阻塞进程

此时信号量等于 0，但等待队列为空，没有阻塞进程，信号量加一



4. 使用WaitForSingleObject 函数等待信号量且会阻塞

信号量减1之后的信号量数值<0，则说明资源不足，调用PspWait使线程插入到等待队列队尾

5. 利用 ReleaseSemaphore 函数释放信号量且有阻塞进程

等待队列不为空，唤醒等待队列队头的进程，为其分配资源

**3）支持等待超时唤醒和批量释放功能的信号量实现**

（给出实现方法的简要描述、源代码、测试及结果等）

3.1等待超时唤醒

在目前 EOS Kernel 项目的 ps/semaphore.c 文件中，PsWaitForSemaphore 函数的 Milliseconds 参数只能是 INFINITE，不支持等待超时功能。

为了实现等待超时功能，我们可以把 PsWaitForSemaphore接收到的等待时间参数传递给 PspWait 函数，并设定等待时间（Millisecodes），当超过预定等待时间，Pspwait会返回超时状态（WAIT\_TIMEOUT），这样系统就会自动唤醒这个线程。

修改PsWaitForSemaphore 函数使计数值和 0 作比较，即判断是否有资源当计数值大于 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）。

STATUS

PsWaitForSemaphore(

IN PSEMAPHORE Semaphore,

IN ULONG Milliseconds

)

/\*++

功能描述：

信号量的 Wait 操作（P 操作）。

参数：

Semaphore -- Wait 操作的信号量对象。

Milliseconds -- 等待超时上限，单位毫秒。

返回值：

STATUS\_SUCCESS。

当你修改信号量使之支持超时唤醒功能后，如果等待超时，应该返回 STATUS\_TIMEOUT。

--\*/

{

BOOL IntState;

STATUS temp;

ASSERT(KeGetIntNesting() == 0); // 中断环境下不能调用此函数。

IntState = KeEnableInterrupts(FALSE); // 开始原子操作，禁止中断。

//

// 目前仅实现了标准记录型信号量，不支持超时唤醒功能，所以 PspWait 函数

// 的第二个参数的值只能是 INFINITE。

//

//Semaphore->Count--;

//if (Semaphore->Count < 0) {

// PspWait(&Semaphore->WaitListHead, INFINITE);

//}

if(Semaphore->Count > 0){

Semaphore->Count--;

temp=STATUS\_SUCCESS;

}else{

temp=PspWait(&Semaphore->WaitListHead, Milliseconds);

}

KeEnableInterrupts(IntState); // 原子操作完成，恢复中断。

return temp;

}

3.2批量释放的实现

在实现批量释放功能时，利用了EOS为每个信号量定义的阻塞队列，当有信号量被释放的时候，检测阻塞队列中是否为空，若不为空，则唤醒队首的线程

STATUS

PsReleaseSemaphore(

IN PSEMAPHORE Semaphore,

IN LONG ReleaseCount,

OUT PLONG PreviousCount

)

/\*++

功能描述：

信号量的 Signal 操作（V 操作）。

参数：

Semaphore -- Wait 操作的信号量对象。

ReleaseCount -- 信号量计数增加的数量。当前只能为 1。当你修改信号量使之支持

超时唤醒功能后，此参数的值能够大于等于 1。

PreviousCount -- 返回信号量计数在增加之前的值。

返回值：

如果成功释放信号量，返回 STATUS\_SUCCESS。

--\*/

{

STATUS Status;

BOOL IntState;

IntState = KeEnableInterrupts(FALSE); // 开始原子操作，禁止中断。

if (Semaphore->Count + ReleaseCount > Semaphore->MaximumCount) {

Status = STATUS\_SEMAPHORE\_LIMIT\_EXCEEDED;

} else {

//

// 记录当前的信号量的值。

//

if (NULL != PreviousCount) {

\*PreviousCount = Semaphore->Count;

}

//

// 目前仅实现了标准记录型信号量，每执行一次信号量的释放操作

// 只能使信号量的值增加 1。

//

//Semaphore->Count++;

//if (Semaphore->Count <= 0) {

// PspWakeThread(&Semaphore->WaitListHead, STATUS\_SUCCESS);

//}

int i;

for(i=ReleaseCount;i>0;i--){

if(!ListIsEmpty(&Semaphore->WaitListHead)){//等待队列不为空，

PspWakeThread(&Semaphore->WaitListHead,STATUS\_SUCCESS);//唤醒

}else{

Semaphore->Count+=i;

break;

}

}

//

// 可能有线程被唤醒，执行线程调度。

//

PspThreadSchedule();

Status = STATUS\_SUCCESS;

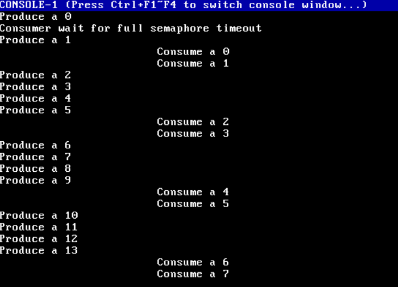
}

KeEnableInterrupts(IntState); // 原子操作完成，恢复中断。

return Status;

}

测试结果：

****

**结果分析：**

（对本实验所做工作及结果进行分析，包括EOS信号量实现方法的特点、不足及改进意见；结合EOS对信号量实现的相关问题提出自己的思考；分析支持等待超时唤醒和批量释放功能的信号量实现方法的有效性、不足和改进意见，如果同时采用了多种实现方法，则进行对比分析；其他需要说明的问题）

通过这次实验，我深刻直观的了解了EOS信号量的实现方法与生产者——消费者问题的实现思路。在这次实验完成后，我了解到了，信号量的等待和释放都是原子操作，所以在单核处理器上可以利用开关中断来保证操作的原子性，但是在多处理机的情况下，开关中断可能就无法保证原子性，需要采用其他的方法，例如使用HLOCK pin引线来实现原子操作。

在等待超时唤醒和批量释放功能的信号量实现方法中。使用了PspWait函数，若等待时间超过预定时间，则返回WAIT\_TIMEOUT以此实现等待超时。在批量释放功能的信号量实现时，我的思路是：逐个释放信号量的时候，检测该信号量的等待队列，如果等待队列不为空，就唤醒被阻塞的线程,这两个功能经过测试，都运行正常。