Oslab4 进程同步实验报告

学号：171860659 姓名：吴紫航　联系方式：401986905@qq.com

一．实验目的

1.学习基于信号量的进程同步机制

2.加深对生产者消费者问题的理解

3.加深对系统调用和框架代码内存分配的理解

二．实验设计

（一）　需要完成的任务:

（1）　实现 sem_post,sem_wait,sem_destroy 系统调用，通过框架代码测试用例

（2）　修改框架代码使其可以支持 8 个进程

（3）　完成用户态的生产者消费者问题（需要实现 getpid 系统调用）

（二）　代码思路

（1-1）

（1）syscallSemPost:

相当于 V 操作，先利用传入的 sf 得到信号量号，如果这个信号量的位置不是 use 状态则操作失败。

否则，将信号量与 0 比较

如果小于 0，则自增后要释放双向链表上的进程

如果大于等于 0，则自增即可，不用释放任何进程

注：这里双向链表的结构比较特殊，值得一提的是如何从双向链表的节点地址得到对应被释放进程的 pcb 的地址呢？

只要根据等式 被释放进程的（pid 地址-blocked 地址）==当前进程的（pid 地址-blocked 地址）

由于当前进程的 pid 与 blocked 是可以访问的，被释放进程的 blocked 通过信号量的链表节点可以查询到，因此，自然得到了被释放进程的 pid，即 pcb 数组的第 pid 项。更改其 state 为 runable

并在返回前更改父子进程返回值即 eax

（2）syscallSemWait

相当于 P 操作，先利用传入的 sf 得到信号量号，如果这个信号量的位置不是 use 状态则操作失败。

否则，将当前信号量自减，如果自减后小于 0，则阻塞当前进程，并把它加入信号量的双向链表中，然后切换进程。切换进程使用 asm volatile(“int $0x20”);即可

（3）SyscallSemDestroy

未定义的操作。先利用传入的 sf 得到信号量号，如果这个信号量的位置不是 use 状态则操作失败。否则只要设置信号量状态为未使用（0）即可

（1-2）

修改 memoory.h 的 NR_SEGMENTS 为 18 即可，因为原本支持 4 个进程，NR_SEGMENT 原本为 10，先要同时进行 8 个进程，则要增加 4 个进程，即 4*2=8 个段（data、code），因此 10 增加到 18 即可

（1-3）

用户态模拟生产者消费者程序，需要两个信号量：full、mutex

Full 限制消费者一定要在有产品的时候才能消费，实现同步，初始化为 0

Mutex 为共享区的锁，实现互斥，起始为 1。

父进程 fork 出 6 个子进程

pid 小的两个（2 和 3）作为生产者

pid 大的四个（4、5、6、7）作为消费者

getpid 只需要增加一条系统调用路线即可

函数中调用 syscall，定义宏 SYS_PID,在 syscallHandle 中的 switch 中补充一条 syscallPid 的系统调用，将当前进程 pid 赋值给 eax 即可返回

将 pid 和信号量传给对应 produce 和 consume 函数。

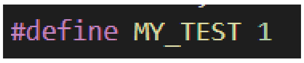
两个生产者各自 Produce 8 个产品，

四个消费者各自 consume 4 个产品

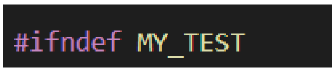
生产者步骤为：try lock，调用 sem_wait(mutex)锁共享区，locked，生产产品，调用 sem_post(full)产生产品，调用 sem_post(mutex)释放锁，unlock

消费者步骤为：try consume,调用 sem_wait(full)查看是否有商品，try lock，调用 sem_wait(mutex)锁共享区，locked，完成消费 consumed，调用 sem_post(mutex)释放锁，unlock

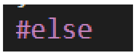
注 1：关于 1-1 和 1-3 测试代码的切换，定义了一个宏#define MY_TEST 1，使用了条件编译 ifndef，else，endif 结构，如果定义了 MY_TEST 则运行 1-3，如注释了 MY_TEST 的定义则运行 1-1



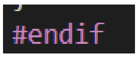
```
#define MY_TEST 1
```



```
#ifndef MY_TEST
```



```
#else
```



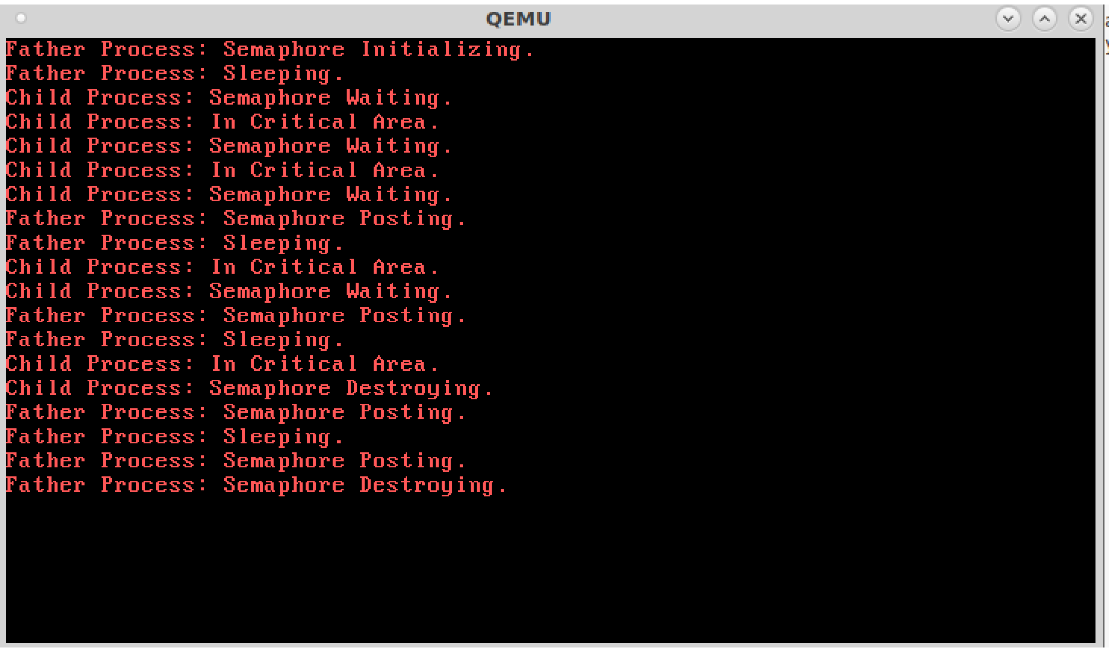
```
#endif
```

注 2：为方便核对实验 1-3 阶段结果，将输出中，在输出到 qemu 同时，内容输出到串口，因此终端可以方便的查看到输出！

注 3：在 produce 和 consume 的过程中，为突出 pv 操作的作用，用户函数每运行几行就会调用 sleep 切换进程，可以更加清晰的感受同步、互斥的机制！

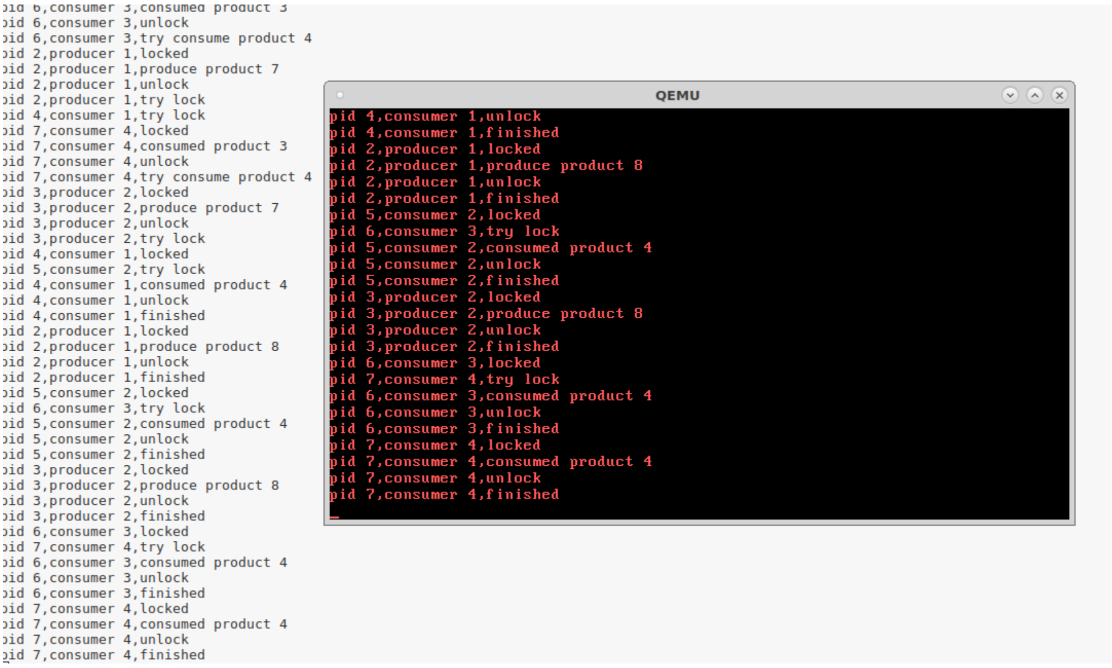
三． 实验结果

（1-1）



```
Father Process: Semaphore Initializing.
Father Process: Sleeping.
Child Process: Semaphore Waiting.
Child Process: In Critical Area.
Child Process: Semaphore Waiting.
Child Process: In Critical Area.
Child Process: Semaphore Waiting.
Father Process: Semaphore Posting.
Father Process: Sleeping.
Child Process: In Critical Area.
Child Process: Semaphore Waiting.
Father Process: Semaphore Posting.
Father Process: Sleeping.
Child Process: In Critical Area.
Child Process: Semaphore Destroying.
Father Process: Semaphore Posting.
Father Process: Sleeping.
Father Process: Semaphore Posting.
Father Process: Semaphore Destroying.
```

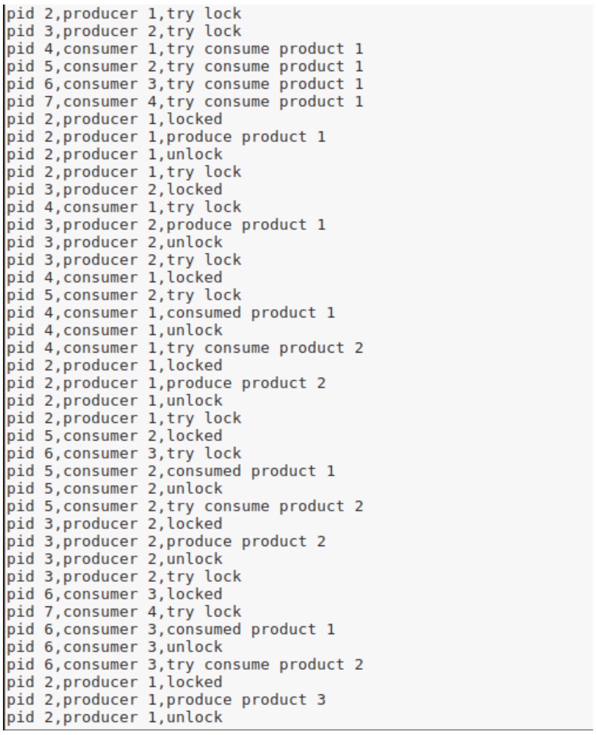
（1-3）qemu 输出



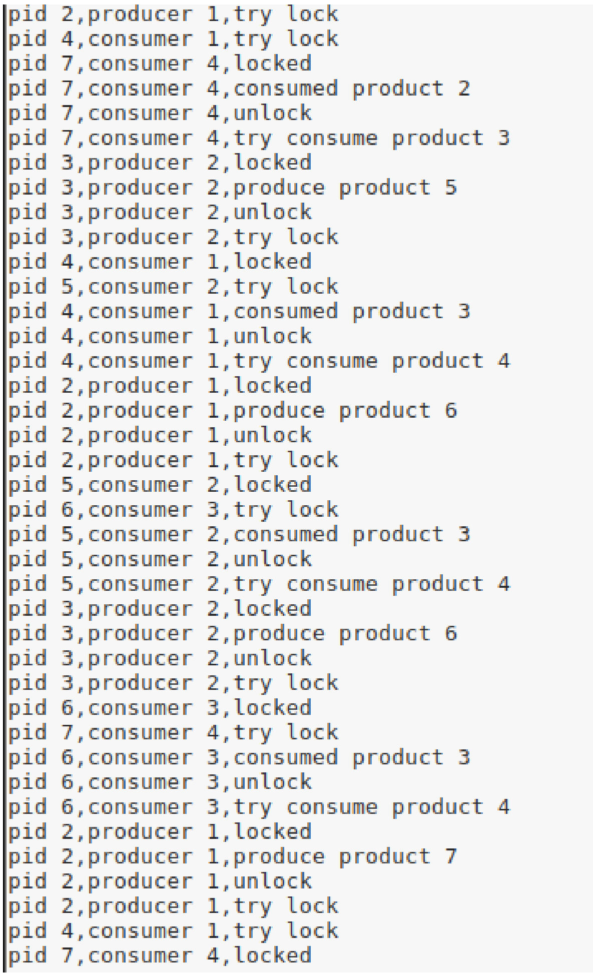
```
pid 6,consumer 3,consumed product 3
pid 6,consumer 3,unlock
pid 6,consumer 3,try consume product 4
pid 2,producer 1,locked
pid 2,producer 1,produce product 7
pid 2,producer 1,unlock
pid 2,producer 1,try lock
pid 4,consumer 1,try lock
pid 7,consumer 4,locked
pid 7,consumer 4,consumed product 3
pid 7,consumer 4,unlock
pid 7,consumer 4,try consume product 4
pid 3,producer 2,locked
pid 3,producer 2,produce product 7
pid 3,producer 2,unlock
pid 3,producer 2,try lock
pid 4,consumer 1,locked
pid 5,consumer 2,try lock
pid 4,consumer 1,consumed product 4
pid 4,consumer 1,unlock
pid 4,consumer 1,finished
pid 2,producer 1,locked
pid 2,producer 1,produce product 8
pid 2,producer 1,unlock
pid 2,producer 1,finished
pid 5,consumer 2,locked
pid 6,consumer 3,try lock
pid 5,consumer 2,consumed product 4
pid 5,consumer 2,unlock
pid 5,consumer 2,finished
pid 3,producer 2,locked
pid 3,producer 2,produce product 8
pid 3,producer 2,unlock
pid 3,producer 2,finished
pid 6,consumer 3,locked
pid 7,consumer 4,try lock
pid 6,consumer 3,consumed product 4
pid 6,consumer 3,unlock
pid 6,consumer 3,finished
pid 7,consumer 4,locked
pid 7,consumer 4,consumed product 4
pid 7,consumer 4,unlock
pid 7,consumer 4,finished
```

```
pid 4,consumer 1,unlock
pid 4,consumer 1,finished
pid 2,producer 1,locked
pid 2,producer 1,produce product 8
pid 2,producer 1,unlock
pid 2,producer 1,finished
pid 5,consumer 2,locked
pid 6,consumer 3,try lock
pid 5,consumer 2,consumed product 4
pid 5,consumer 2,unlock
pid 5,consumer 2,finished
pid 3,producer 2,locked
pid 3,producer 2,produce product 8
pid 3,producer 2,unlock
pid 3,producer 2,finished
pid 6,consumer 3,locked
pid 7,consumer 4,try lock
pid 6,consumer 3,consumed product 4
pid 6,consumer 3,unlock
pid 6,consumer 3,finished
pid 7,consumer 4,locked
pid 7,consumer 4,consumed product 4
pid 7,consumer 4,unlock
pid 7,consumer 4,finished
```

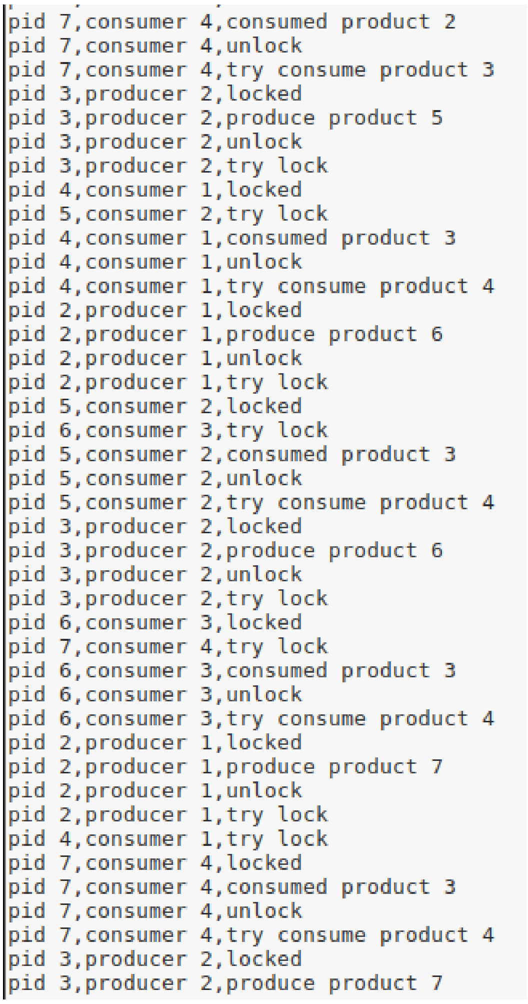
（1-3）串口输出



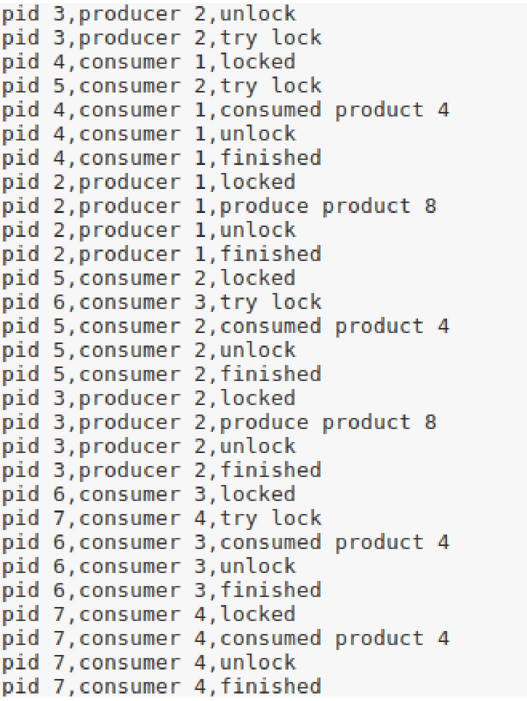
```
pid 2,producer 1,try lock
pid 3,producer 2,try lock
pid 4,consumer 1,try consume product 1
pid 5,consumer 2,try consume product 1
pid 6,consumer 3,try consume product 1
pid 7,consumer 4,try consume product 1
pid 2,producer 1,locked
pid 2,producer 1,produce product 1
pid 2,producer 1,unlock
pid 2,producer 1,try lock
pid 3,producer 2,locked
pid 4,consumer 1,try lock
pid 3,producer 2,produce product 1
pid 3,producer 2,unlock
pid 3,producer 2,try lock
pid 4,consumer 1,locked
pid 5,consumer 2,try lock
pid 4,consumer 1,consumed product 1
pid 4,consumer 1,unlock
pid 4,consumer 1,try consume product 2
pid 2,producer 1,locked
pid 2,producer 1,produce product 2
pid 2,producer 1,unlock
pid 2,producer 1,try lock
pid 5,consumer 2,locked
pid 6,consumer 3,try lock
pid 5,consumer 2,consumed product 1
pid 5,consumer 2,unlock
pid 5,consumer 2,try consume product 2
pid 3,producer 2,locked
pid 3,producer 2,produce product 2
pid 3,producer 2,unlock
pid 3,producer 2,try lock
pid 6,consumer 3,locked
pid 7,consumer 4,try lock
pid 6,consumer 3,consumed product 1
pid 6,consumer 3,unlock
pid 6,consumer 3,try consume product 2
pid 2,producer 1,locked
pid 2,producer 1,produce product 3
pid 2,producer 1,unlock
```



```
pid 2,producer 1,try lock
pid 4,consumer 1,try lock
pid 7,consumer 4,locked
pid 7,consumer 4,consumed product 2
pid 7,consumer 4,unlock
pid 7,consumer 4,try consume product 3
pid 3,producer 2,locked
pid 3,producer 2,produce product 5
pid 3,producer 2,unlock
pid 3,producer 2,try lock
pid 4,consumer 1,locked
pid 5,consumer 2,try lock
pid 4,consumer 1,consumed product 3
pid 4,consumer 1,unlock
pid 4,consumer 1,try consume product 4
pid 2,producer 1,locked
pid 2,producer 1,produce product 6
pid 2,producer 1,unlock
pid 2,producer 1,try lock
pid 5,consumer 2,locked
pid 6,consumer 3,try lock
pid 5,consumer 2,consumed product 3
pid 5,consumer 2,unlock
pid 5,consumer 2,try consume product 4
pid 3,producer 2,locked
pid 3,producer 2,produce product 6
pid 3,producer 2,unlock
pid 3,producer 2,try lock
pid 6,consumer 3,locked
pid 7,consumer 4,try lock
pid 6,consumer 3,consumed product 3
pid 6,consumer 3,unlock
pid 6,consumer 3,try consume product 4
pid 2,producer 1,locked
pid 2,producer 1,produce product 7
pid 2,producer 1,unlock
pid 2,producer 1,try lock
pid 4,consumer 1,try lock
pid 7,consumer 4,locked
```



```
pid 7,consumer 4,consumed product 2
pid 7,consumer 4,unlock
pid 7,consumer 4,try consume product 3
pid 3,producer 2,locked
pid 3,producer 2,produce product 5
pid 3,producer 2,unlock
pid 3,producer 2,try lock
pid 4,consumer 1,locked
pid 5,consumer 2,try lock
pid 4,consumer 1,consumed product 3
pid 4,consumer 1,unlock
pid 4,consumer 1,try consume product 4
pid 2,producer 1,locked
pid 2,producer 1,produce product 6
pid 2,producer 1,unlock
pid 2,producer 1,try lock
pid 5,consumer 2,locked
pid 6,consumer 3,try lock
pid 5,consumer 2,consumed product 3
pid 5,consumer 2,unlock
pid 5,consumer 2,try consume product 4
pid 3,producer 2,locked
pid 3,producer 2,produce product 6
pid 3,producer 2,unlock
pid 3,producer 2,try lock
pid 6,consumer 3,locked
pid 7,consumer 4,try lock
pid 6,consumer 3,consumed product 3
pid 6,consumer 3,unlock
pid 6,consumer 3,try consume product 4
pid 2,producer 1,locked
pid 2,producer 1,produce product 7
pid 2,producer 1,unlock
pid 2,producer 1,try lock
pid 4,consumer 1,try lock
pid 7,consumer 4,locked
pid 7,consumer 4,consumed product 3
pid 7,consumer 4,unlock
pid 7,consumer 4,try consume product 4
pid 3,producer 2,locked
pid 3,producer 2,produce product 7
```



```
pid 3,producer 2,unlock
pid 3,producer 2,try lock
pid 4,consumer 1,locked
pid 5,consumer 2,try lock
pid 4,consumer 1,consumed product 4
pid 4,consumer 1,unlock
pid 4,consumer 1,finished
pid 2,producer 1,locked
pid 2,producer 1,produce product 8
pid 2,producer 1,unlock
pid 2,producer 1,finished
pid 5,consumer 2,locked
pid 6,consumer 3,try lock
pid 5,consumer 2,consumed product 4
pid 5,consumer 2,unlock
pid 5,consumer 2,finished
pid 3,producer 2,locked
pid 3,producer 2,produce product 8
pid 3,producer 2,unlock
pid 3,producer 2,finished
pid 6,consumer 3,locked
pid 7,consumer 4,try lock
pid 6,consumer 3,consumed product 4
pid 6,consumer 3,unlock
pid 6,consumer 3,finished
pid 7,consumer 4,locked
pid 7,consumer 4,consumed product 4
pid 7,consumer 4,unlock
pid 7,consumer 4,finished
```

四. 实验收获

加深了对同步、互斥机制、共享区的理解

五. 实验建议

暂无