实　验　报　告

课程名称　操作系统　**实验名称**　进程互斥实例（消费者生产者问题）

姓　名　李世旺　**学　号**　1307402068　**专业班级**　信息与计算科学

实验日期　2016　**年**　3　**月**　24　**日**　　**成绩**

实验目的

用信号量实现生产者消费者问题

实验方案

生产者和消费者问题是多个相互合作的进程之间的一种抽象。生产者和消费者之间的关系:

1. 对缓冲区的访问是互斥的。由于两者都会修改缓冲区，因此，一方修改缓冲区时，另一方不能修改，这就是互斥。

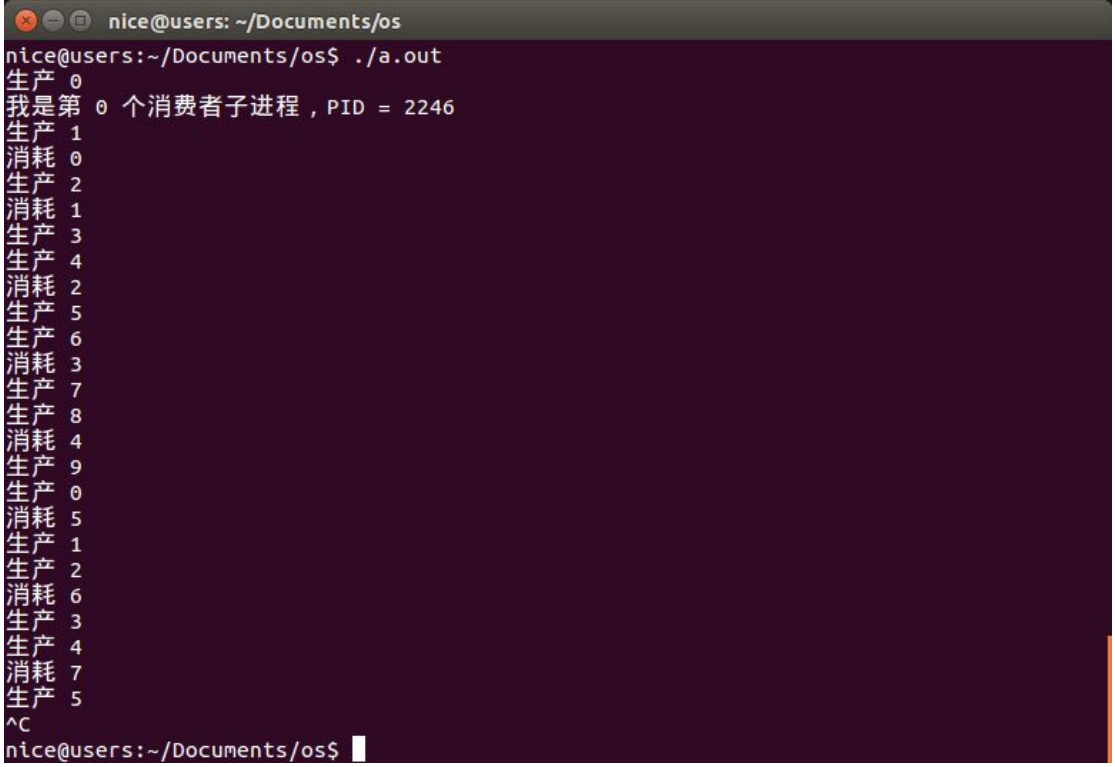
2. 一方的行为影响另一方。缓冲区不空，才能消费，何时不空？生产了就不空；缓冲区满，就不能生产，何时不满？消费了就不满。这是同步关系。

为了描述这种关系，一方面，使用共享内存代表缓冲区；另一方面，使用 互斥信号量 控制对缓冲区的访问，使用同步信号量描述两者的依赖关系。

共享存储是进程间通信的一种手段，通常，使用信号量同步或互斥访问共享存储。共享存储的原理是将进程的地址空间映射到一个共享存储段。在 LINUX 下，通过使用 shmget 函数创建或者获取共享内存。

执行结果与分析

如下图所示：
先生产的产品总是先被消费。



详细代码

```
#include <stdio.h>

#include <unistd.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <stdlib.h>

#define SHM_SIZE (1024*1024)

#define SHM_MODE 0600

#define SEM_MODE 0600


#if defined(__GNU_LIBRARY__) && !defined(_SEM_SEMUN_UNDEFINED)

/* union semun is defined by including <sys/sem.h> */

#else
```

```
/* according to X/OPEN we have to define it ourselves */
union semun{
    int val;
    struct semid_ds *buf;
    unsigned short *array;
};
#endif

struct ShM{
    int start;
    int end;
}* pSM;

const int N_CONSUMER = 1//消费者数量
const int N_BUFFER = 10;//缓冲区容量
int shmId = -1,semSetId=-1;
union semun su;//sem union，用于初始化信号量

//semSetId 表示信号量集合的 id
//semNum 表示要处理的信号量在信号量集合中的索引
void waitSem(int semSetId,int semNum)
{
```

```
    struct sembuf sb;
    sb.sem_num = semNum;
    sb.sem_op = -1;//表示要把信号量减一
    sb.sem_flg = SEM_UNDO;//
    //第二个参数是  sembuf []  类型的，表示数组
    //第三个参数表示  第二个参数代表的数组的大小
    if(semop(semSetId,&sb,1) < 0){
        perror("waitSem failed");
        exit(1);
    }
}
void sigSem(int semSetId,int semNum)
{
    struct sembuf sb;
    sb.sem_num = semNum;
    sb.sem_op = 1;
    sb.sem_flg = SEM_UNDO;
    //第二个参数是  sembuf []  类型的，表示数组
    //第三个参数表示  第二个参数代表的数组的大小
    if(semop(semSetId,&sb,1) < 0){
        perror("waitSem failed");
        exit(1);
```

```
        }
}
//必须在保证互斥以及缓冲区不满的情况下调用
void produce()
{
        int last = pSM->end;
        pSM->end = (pSM->end+1) % N_BUFFER;
        printf("生产  %d\n",last);
}
//必须在保证互斥以及缓冲区不空的情况下调用
void consume()
{
        int last = pSM->start;
        pSM->start = (pSM->start + 1)%N_BUFFER;
        printf("消耗  %d\n",last);
}

void init()
{
        //缓冲区分配以及初始化
        if((shmId = shmget(IPC_PRIVATE,SHM_SIZE,SHM_MODE)) < 0)
        {
```

```
        perror("create shared memory failed");
        exit(1);
}
pSM = (struct ShM *)shmat(shmId,0,0);
pSM->start = 0;
pSM->end = 0;

//信号量创建
//第一个:同步信号量,表示先后顺序,必须有空间才能生产
//第二个:同步信号量,表示先后顺序,必须有产品才能消费
//第三个:互斥信号量,生产者和每个消费者不能同时进入缓冲区

if((semSetId = semget(IPC_PRIVATE,3,SEM_MODE)) < 0)
{
        perror("create semaphore failed");
        exit(1);
}
//信号量初始化,其中  su  表示  union semun
su.val = N_BUFFER;//当前库房还可以接收多少产品
if(semctl(semSetId,0,SETVAL, su) < 0){
        perror("semctl failed");
        exit(1);
```

```
        }
        su.val = 0;//当前没有产品
        if(semctl(semSetId,1,SETVAL,su) < 0){
                perror("semctl failed");
                exit(1);
        }
        su.val = 1;//为 1 时可以进入缓冲区
        if(semctl(semSetId,2,SETVAL,su) < 0){
                perror("semctl failed");
                exit(1);
        }
}
int main()
{
        int i = 0,child = -1;
        init();
        //创建 多个（N_CONSUMER）消费者子进程
        for(i = 0; i < N_CONSUMER; i++)
        {
                if((child = fork()) < 0)//调用 fork 失败
                {
                        perror("the fork failed");
```

```
            exit(1);
        }
        else if(child == 0)//子进程
        {
            printf("我是第 %d 个消费者子进程，PID = %d\n",i,getpid());
            while(1)
            {
                waitSem(semSetId,1);//必须有产品才能消费
                waitSem(semSetId,2);//锁定缓冲区
                consume();//获得产品,需要修改缓冲区
                sigSem(semSetId,2);//释放缓冲区
                sigSem(semSetId,0);//告知生产者,有空间了
                sleep(2);//消费频率
            }
            break;//务必有
        }
    }


    //父进程开始生产
    if(child > 0)
    {
```

```
        while(1)
        {
            waitSem(semSetId,0);//获取一个空间用于存放产品
            waitSem(semSetId,2);//占有产品缓冲区
            produce();
            sigSem(semSetId,2);//释放产品缓冲区
            sleep(1);//每两秒生产一个
            sigSem(semSetId,1);//告知消费者有产品了
        }
    }
    return 0;
}
```