# 实验三 生产者与消费者实验

| 姓名 | 学号 | 学院 | 日期 |
|---|---|---|---|
| 臧祝利 | 202011998088 | 人工智能学院 | 2022.10.18 |

## 实验目的

了解和熟悉linux系统下的信号量集和共享内存。

## 实验任务

使用linux系统提供的信号量集( `semget` 、 `semop` 、 `semctl` 等系统调用)和共享内存( `shmget` 、 `shmat` 、 `shmdt` 、 `shmctl` 等系统调用)实现生产者和消费者问题。

## 实验要求

1. 写两个程序，一个模拟生产者过程，一个模拟消费者过程；
2. 创建一个共享内存来模拟生产者-消费者问题中缓冲队列，该缓冲队列有N（例如N=10）个缓冲区，每个缓冲区的大小为1024B，每个生产者和消费者对缓冲区必须互斥访问；缓冲队列除了N个缓冲区外，还有一个指向第一个空缓冲区的指针 `in` ，一个指向第一个满缓冲区的指针 `out` ；
3. 由第一个生产者创建信号量集和共享内存，其他生产者和消费者可以使用该信号量集和共享内存；
4. 生产者程序：生产者生产产品（即是从键盘输入长度小于1024B的字符）放入空的缓冲区；
5. 消费者程序：消费者消费产品（即从满的缓冲区中取出内容在屏幕上打印出来），然后满的缓冲区变为空的缓冲区;
6. 多次运行生产者程序和消费者进程，同时产生多个生产者进程和多个消费者进程，这些进程共享这些信号量和共享内存，实现生产者和消费者问题；
7. 在生产者程序中，可以选择：

   (1) 生产产品；

   (2) 退出。退出进程，但信号量和共享内存仍然存在，其他生产者进程和消费者进程还可以继续使用；

   (3) 删除信号量和共享内存。显性删除信号量和共享内存，后续其他生产者进程和消费者进程都不能使用这些信号量和共享内存。
8. 在消费者程序中，可以选择：

 (1) 消费产品；

 (2) 退出。退出进程，但信号量和共享内存仍然存在，其他生产者进程和消费者进程还 可以继续使用；

 (3) 删除信号量和共享内存。显性删除信号量和共享内存，后续其他生产者进程和消费 者进程都不能使用这些信号量和共享内存。

---

# 实验报告

## 实验原理

### 理论原理

缓冲区为**临界资源**；

互斥关系：生产者访问缓冲区时，消费者不能访问；生产者之间不能访问同一个缓冲区；反之亦然；

同步关系：生产者必须在消费者之前进行；

由于存在互斥问题，因此设置互斥信号量 `mutex=1` ，用于控制两个进程互斥访问；

设置 `full` 信号量记录缓冲队列中已经满的缓冲区数，初值为0；

设置 `empty` 信号量记录当前空的缓冲区数，初始值为N；

伪代码如下：

`生产者`：

```
void producer(){
    while(true){
        生产一个数据;
        P(empty);//申请一个空白的区域用来存放生产的数据，此时empty - 1
        //申请完了之后，互斥进入缓冲区，使得其他进程不能访问该缓冲区
        p(mutex);//mutex = mutex - 1 = 0
        [将数据放入缓冲区;]
        V(mutex);//退出缓冲区,互斥信号量恢复为1
        V(full);//满的缓冲区 +1
    }
}
```

`消费者`：

```
void consumer(){
    while(true){
        P(full);//申请一块满的缓冲区
        P(mutex);//互斥进入
        [将数据从缓冲池中取出来;]
        V(mutex);//互斥退出
        V(empty);//释放缓冲区，此时缓冲区状态为空
                 //缓冲池中的空的缓冲区 +1
        消费取出的数据。
    }
}
```

## 函数使用

### 信号量相关

`int semget(key_t key,int nsems,int flag);`

**功能**：创建新的 `信号量集` 或获取已存在的 `信号量集`

**参数**：

- `key` 通常为0，可以指定一个键值，或者使用 `ftok()` 函数获得一个唯一的键值；
- `nsems` 为信号量个数， `此实验为3个，分别为 mutex、full、empty` ；
- `flag` ：
    - `IPC_CREAT` ：如果信号量不存在，创建信号量，否则获取已存在的信号量；
    - `IPC_CREAT|IPC_EXCL` :不存在信号量时才会创建，否则产生错误；
    - `因此可以先使用 IPC_CREAT|IPC_EXCL 参数尝试创建，如果创建失败使用 IPC_CREAT 获取` ；

**返回值**：成功返回信号量的标识码，失败返回-1；

`int semctl(int semid,int semnum,int cmd,union semun arg);`

**功能**：控制信号量的信息；

**参数**：

- `semid` ：标识符，由 `semget()` 得到；
- `semnum` ：操作信号的第几个信号量；
- `cmd` 表示要进行的操作， `这里主要使用 SETVAL 来设置信号量的值`

**返回值**：成功返回0，失败返回-1；

`int semop(int semid,struct sembuf* sops.unsigned nsops);`

**功能：** 改变信号量的值；

`key_t ftok(const char *pathname,int proj_id)`；

**功能**：根据文件的路径和id值得到一个`key`值，其中两个参数都可以随意设置；

### 共享内存相关

`int shimd(key_t key, size_t size, int flag)`

**功能**：创建或获得一块共享内存

`int shmdt(const void *addr);`

**功能**：分离共享的内存区域；

`void *viraddr = shmat(int shmid, const void *addr, int flag)`

**功能**：调用共享区链接到虚地址空间；

`int shmctl(int shmid, int cmd, struct shimd_ds shmstatbuf)`

**功能**：查询共享存储区的状态和设置参数

## 实验设计

```
#define N 10   //缓冲区数目
#define BUFFERSIZE 1024   //缓冲区大小
#define SEM_NUM 3   //信号量数目
#define SEM_SIZE 1024*(N+1)   //共享存储区大小
```

缓冲队列定义如下：

```
struct Memory{
    char memory[N][BUFFERSIZE];   //缓冲区
    int in, out;   //指针
};
```

封装P、V操作

```
int P(int semid, int semnum){
    struct sembuf sops={semnum,-1,0};
    return (semop(semid,&sops,1));
}

int V(int semid, int semnum){
    struct sembuf sops={semnum,+1,0};
    return (semop(semid,&sops,1));
}
```

### 生产者

`producer.c`需要实现以下功能：

- 创建信号量集（首次运行）/获取信号量集
    - 创建信号量集若成功后,初始化信号量
        - 0660：百位6 -- 本用户有读写权；十位6 -- 同组用户有读写权；

```
    semid = semget(key,SEM_NUM,0660|IPC_CREAT|IPC_EXCL);   //尝试创建
    if (semid == -1){   //创建失败
        if (errno == EEXIST){
```

```
        printf("semget() warning: %s\n", strerror(errno));
        semid = semget(key,0,0660|IPC_CREAT);   //获取信号量集
        if (semid == -1){
            printf("semget() error: %s\n",strerror(errno));
            return -1;
        }
        printf("exist semget() success. semid=[%d]\n",semid);   //输出已存在
    }
    else{
        printf("semget() error: %s\n",strerror(errno));
        return -1;
    }
}else{   //进行初始化，初值mutex=1,full=0,empty=10;
    printf("create semget() success. semid=[%d]\n",semid);   //新建信号量集
    arg.val = 1;   //init mutex value
    ret = semctl(semid,0,SETVAL,arg);   //mutex
    if (ret < 0){
        perror("ctl sem error");
        semctl(semid,0,IPC_RMID,arg);
        return -1;
    }
    arg.val=0;   //init full value
    ret = semctl(semid,1,SETVAL,arg);   //full
    if (ret<0){
        perror("ctl sem error");
        semctl(semid,0,IPC_RMID,arg);
        return -1;
    }
    arg.val=N;   //init empty value
    ret = semctl(semid,2,SETVAL,arg);   //empty
    if (ret<0){
        perror("ctl sem error");
        semctl(semid,0,IPC_RMID,arg);
        return -1;
    }
}
```

- 创建共享内存（首次运行）/获取共享内存
    - 若首次创建，赋初始值；

```
shmid = shmget((key_t)key,SEM_SIZE,0660|IPC_CREAT|IPC_EXCL);   //尝试创建
if (shmid == -1){
    shmid = shmget(key,,0660|IPC_CREAT);   //已经创建，则获取
    if (shmid==-1){
        printf("shmget() failed\n");
        exit(EXIT_FAILURE);
    }else{
        shm = shmat(shmid,NULL,0);
        sharing = (struct Memory *)shm;   //访问共享内存
    }
}else{
    shm = shmat(shmid,NULL,0);
    sharing = (struct Memory *)shm;
    for (int i=0;i<N;i++){   //初始化
        strcpy(sharing->memory[i],"");
    }
    sharing->in = 0;
    sharing->out = 0;
}
```

- 操作选择
  - 操作为1时，会输出生产物品之前的状态；待输入生产的产品后，程序会输出产品的名称，同时输出生产后的状态；

```
    if (option==1){
            mutex = semctl(semid,0,GETVAL,arg);  //获取mutex
            full = semctl(semid,1,GETVAL,arg);  //获取full
            empty = semctl(semid,2,GETVAL,arg);  //获取empty
            printf("product before. semval[0](mutex)=[%d] semval[1](full)=[%d] semval[2](empty)=
[%d]\n",mutex,full,empty);
            printf("in: %d, out: %d\n",sharing->in,sharing->out);
            if (full == N){  //如果满了，输出这句话，以等待消费者消费
                printf("Please wait consumer...\n");
            }
            while ((full=semctl(semid,1,GETVAL,arg))==10);  //等待ing
            printf("Input your message:");
            scanf("%s",msg);
            P(semid,2);  //wait(empty);
            P(semid,0);  //wait(mutex);
            strcpy(sharing->memory[sharing->in],msg);
            sharing->in = (sharing->in + 1)%N;
            printf("Products:%s\n",msg);
            V(semid,0);  //signal(mutex);
            V(semid,1);  //signal(full);
            mutex = semctl(semid,0,GETVAL,arg);  //再获取，以输出生产完后的值
            full = semctl(semid,1,GETVAL,arg);
            empty = semctl(semid,2,GETVAL,arg);
            printf("product finish. semval[0](mutex)=[%d] semval[1](full)=[%d] semval[2](empty)=
[%d]\n",mutex,full,empty);
            printf("product in=[%d]\n",sharing->in);
```

  - 操作为2时，会退出

```
    else if (option==2){
            shmdt(sharing);  //释放内存
            printf("Exit from the process\n");
            exit(0);
        }
```

  - 操作为3时，调用函数删除信号量集和共享内存；

```
    else{
            shmdt(sharing);  //释放内存
            semctl(semid,IPC_RMID,0);  //删除信号量集
            shmctl(shmid,IPC_RMID,0);  //删除共享内存
            printf("Delete the sharing memory\n");
            exit(1);
        }
```

## 消费者

`consumer.c` 需要实现的功能：

- 获取信号量集
  - `consumer.c` 不需要创建新的信号量集；

```
    semid = semget(key,SEM_NUM,0660|IPC_CREAT);  //获取信号量集
    if (semid == -1){
        perror("create semget error");
        return -1;

    }else{
        printf("semget() success. semid=[%d]\n",semid);
    }
```

- 如果第一个 `producer` 没执行，即未建立共享内存，退出程序；

```
    if ((shmid = shmget(key, SEM_SIZE, 0666 | IPC_CREAT | IPC_EXCL)) ≠ -1)
    {
        printf("Please create shared memory\n");
        semctl(semid, 0, IPC_RMID, 0);
        shmctl(shmid, IPC_RMID, 0);
        return -1;
    }
```

- 获取共享内存

```
    shmid = shmget(key, SEM_SIZE, 0660 | IPC_CREAT);
    if (shmid == -1)
    {
        printf("shmget() failed\n");
        exit(EXIT_FAILURE);
    }
    else
    {
        printf("shmget() success. shmid=[%d]\n", shmid);
        shm = shmat(shmid, NULL, 0);
        sharing = (struct Memory *)shm;
    }
```

- 操作选择
    - 选择为1时，会输出消费前的共享内存的状态，然后消费后会输出之后的状态；

```
    if (option == 1)
        {
            mutex = semctl(semid, 0, GETVAL, arg);
            full = semctl(semid, 1, GETVAL, arg);
            empty = semctl(semid, 2, GETVAL, arg);
            printf("consume before. semval[0](mutex)=[%d] semval[1](full)=[%d] semval[2](empty)=
[%d]\n", mutex, full, empty);
            printf("before———in: %d, out: %d\n", sharing→in, sharing→out);
            if (full == 0)
            {
                printf("Please wait producer ... \n");
            }
            while ((full = semctl(semid, 1, GETVAL, arg)) == 0)
                ;

            P(semid, 1); // wait(full);
            P(semid, 0); // wait(mutex);

            strcpy(sharing→memory[sharing→out], message);
            sharing→out = (sharing→out + 1) % N;

            V(semid, 0); // signal(mutex);
            V(semid, 2); // signal(empty);
            mutex = semctl(semid, 0, GETVAL, arg);
            full = semctl(semid, 1, GETVAL, arg);
            empty = semctl(semid, 2, GETVAL, arg);
            printf("consume finish. semval[0](mutex)=[%d] semval[1](full)=[%d] semval[2](empty)=
[%d]\n", mutex, full, empty);
            printf("product out=[%d]\n",sharing->out);
            printf("after———in: %d, out: %d\n", sharing->in, sharing->out);
        }
```

- 操作为2，退出

```
if (option == 1)
        {
            mutex = semctl(semid, 0, GETVAL, arg);
            full = semctl(semid, 1, GETVAL, arg);
            empty = semctl(semid, 2, GETVAL, arg);
            printf("consume before. semval[0](mutex)=[%d] semval[1](full)=[%d] semval[2](empty)=
[%d]\n", mutex, full, empty);
            printf("before———in: %d, out: %d\n", sharing->in, sharing->out);
            if (full == 0)
            {
                printf("Please wait producer ... \n");
            }
            while ((full = semctl(semid, 1, GETVAL, arg)) == 0)
                ;

            P(semid, 1); // wait(full);
            P(semid, 0); // wait(mutex);

            strcpy(sharing->memory[sharing->out], message);
            sharing->out = (sharing->out + 1) % N;

            V(semid, 0); // signal(mutex);
            V(semid, 2); // signal(empty);

            mutex = semctl(semid, 0, GETVAL, arg);
            full = semctl(semid, 1, GETVAL, arg);
            empty = semctl(semid, 2, GETVAL, arg);
            printf("consume finish. semval[0](mutex)=[%d] semval[1](full)=[%d] semval[2](empty)=
[%d]\n", mutex, full, empty);
            printf("after———in: %d, out: %d\n", sharing->in, sharing->out);
        }
```

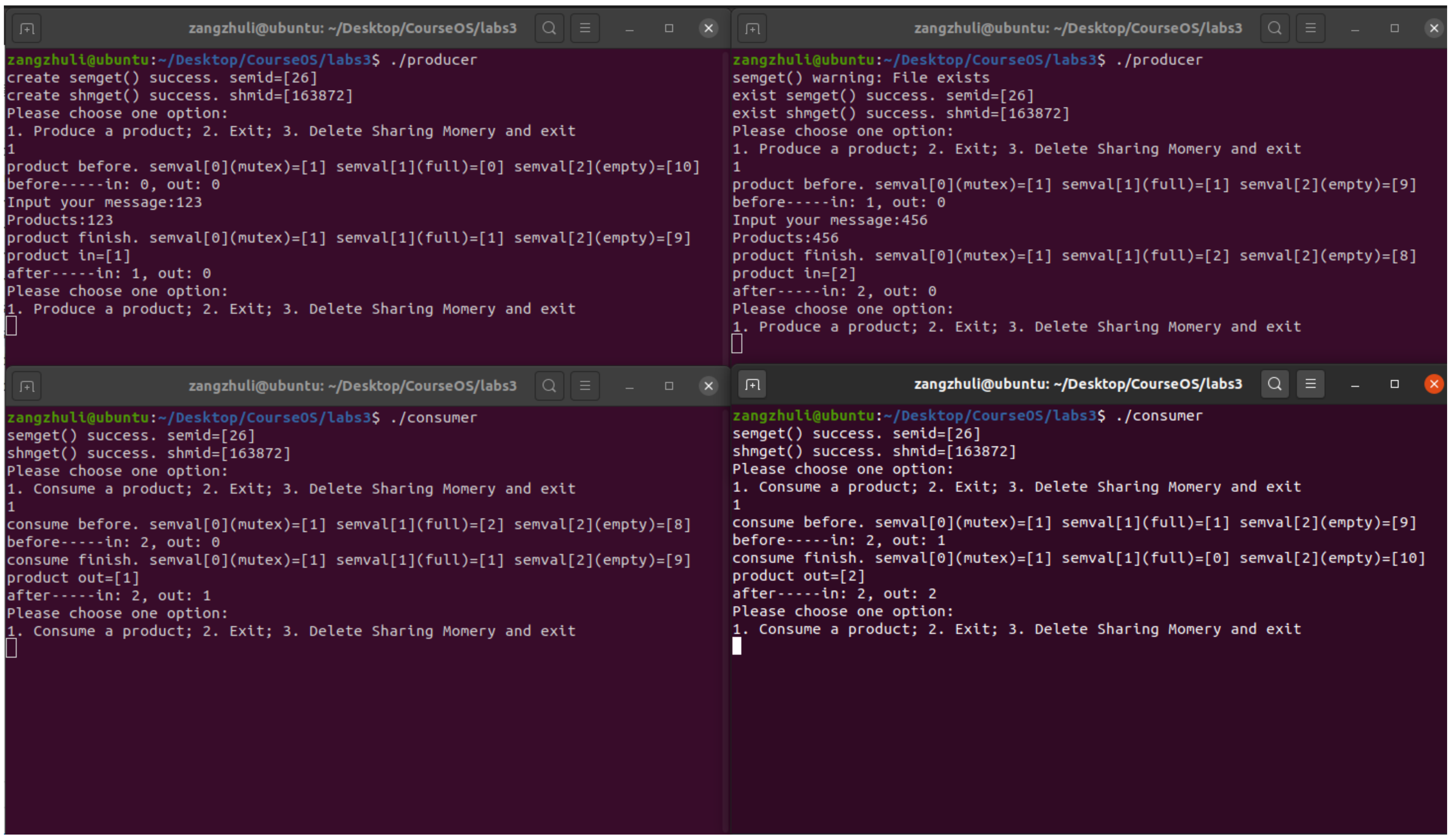
- 操作为3，删除信号量集和共享内存

```
    else
        {
            shmdt(sharing);
            semctl(semid, IPC_RMID, 0);
            shmctl(shmid, IPC_RMID, 0);
            printf("Delete the sharing memory\n");
            exit(1);
        }
    }
```

## 实验结果

如图，第一行为两个生产者，第二行为两个消费者；

操作为第一个生产者建立信号量集和共享内存，并生产产品 123 ，第二个生产者生产产品 456 ，第一个消费者和第二个消费者分别消费一个产品；



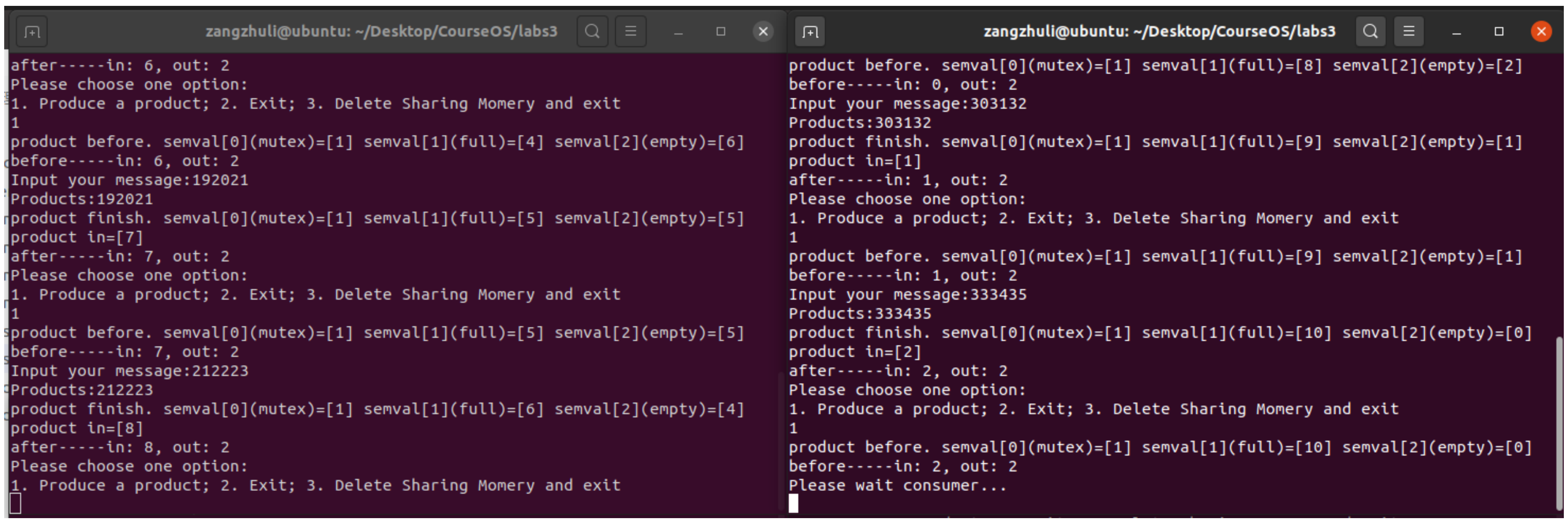
```
zangzhuli@ubuntu:~/Desktop/CourseOS/labs3$ ./producer
create semget() success. semid=[26]
create shmget() success. shmid=[163872]
Please choose one option:
1. Produce a product; 2. Exit; 3. Delete Sharing Momery and exit
1
product before. semval[0](mutex)=[1] semval[1](full)=[0] semval[2](empty)=[10]
before-----in: 0, out: 0
Input your message:123
Products:123
product finish. semval[0](mutex)=[1] semval[1](full)=[1] semval[2](empty)=[9]
product in=[1]
after-----in: 1, out: 0
Please choose one option:
1. Produce a product; 2. Exit; 3. Delete Sharing Momery and exit
```

```
zangzhuli@ubuntu:~/Desktop/CourseOS/labs3$ ./producer
semget() warning: File exists
exist semget() success. semid=[26]
exist shmget() success. shmid=[163872]
Please choose one option:
1. Produce a product; 2. Exit; 3. Delete Sharing Momery and exit
1
product before. semval[0](mutex)=[1] semval[1](full)=[1] semval[2](empty)=[9]
before-----in: 1, out: 0
Input your message:456
Products:456
product finish. semval[0](mutex)=[1] semval[1](full)=[2] semval[2](empty)=[8]
product in=[2]
after-----in: 2, out: 0
Please choose one option:
1. Produce a product; 2. Exit; 3. Delete Sharing Momery and exit
```

```
zangzhuli@ubuntu:~/Desktop/CourseOS/labs3$ ./consumer
semget() success. semid=[26]
shmget() success. shmid=[163872]
Please choose one option:
1. Consume a product; 2. Exit; 3. Delete Sharing Momery and exit
1
consume before. semval[0](mutex)=[1] semval[1](full)=[2] semval[2](empty)=[8]
before-----in: 2, out: 0
consume finish. semval[0](mutex)=[1] semval[1](full)=[1] semval[2](empty)=[9]
product out=[1]
after-----in: 2, out: 1
Please choose one option:
1. Consume a product; 2. Exit; 3. Delete Sharing Momery and exit
```

```
zangzhuli@ubuntu:~/Desktop/CourseOS/labs3$ ./consumer
semget() success. semid=[26]
shmget() success. shmid=[163872]
Please choose one option:
1. Consume a product; 2. Exit; 3. Delete Sharing Momery and exit
1
consume before. semval[0](mutex)=[1] semval[1](full)=[1] semval[2](empty)=[9]
before-----in: 2, out: 1
consume finish. semval[0](mutex)=[1] semval[1](full)=[0] semval[2](empty)=[10]
product out=[2]
after-----in: 2, out: 2
Please choose one option:
1. Consume a product; 2. Exit; 3. Delete Sharing Momery and exit
```

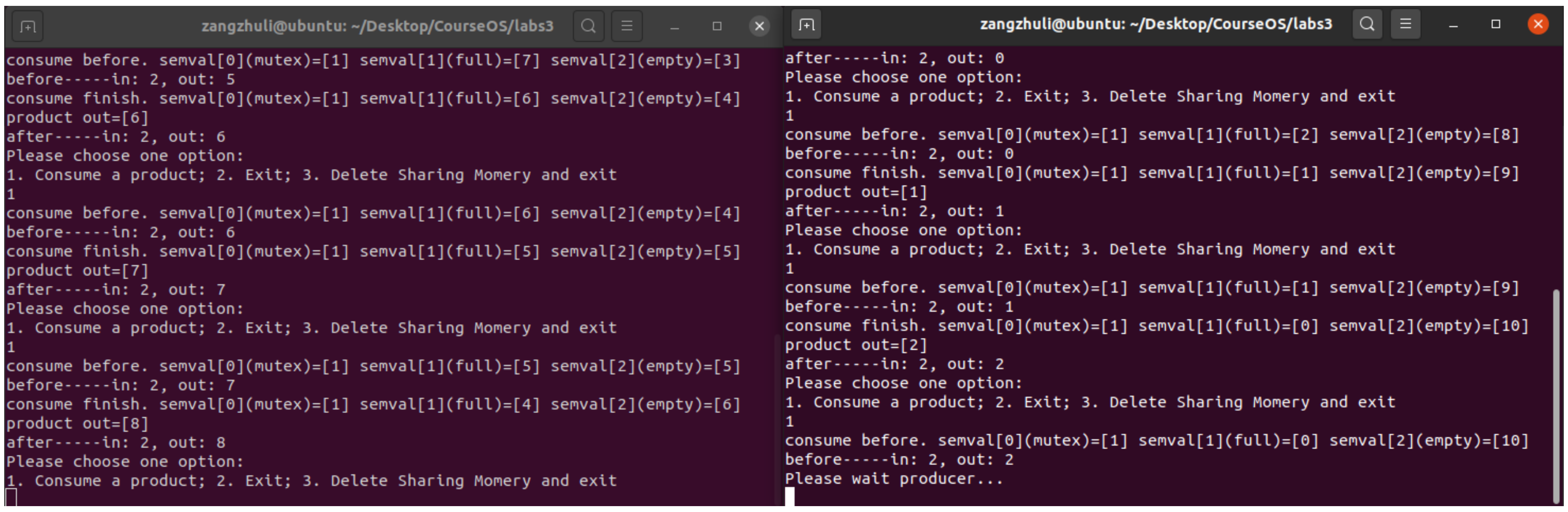
如果生产满了，会提示等待消费者消费



```
after-----in: 6, out: 2
Please choose one option:
1. Produce a product; 2. Exit; 3. Delete Sharing Momery and exit
1
product before. semval[0](mutex)=[1] semval[1](full)=[4] semval[2](empty)=[6]
before-----in: 6, out: 2
Input your message:192021
Products:192021
product finish. semval[0](mutex)=[1] semval[1](full)=[5] semval[2](empty)=[5]
product in=[7]
after-----in: 7, out: 2
Please choose one option:
1. Produce a product; 2. Exit; 3. Delete Sharing Momery and exit
1
product before. semval[0](mutex)=[1] semval[1](full)=[5] semval[2](empty)=[5]
before-----in: 7, out: 2
Input your message:212223
Products:212223
product finish. semval[0](mutex)=[1] semval[1](full)=[6] semval[2](empty)=[4]
product in=[8]
after-----in: 8, out: 2
Please choose one option:
1. Produce a product; 2. Exit; 3. Delete Sharing Momery and exit
```

```
product before. semval[0](mutex)=[1] semval[1](full)=[8] semval[2](empty)=[2]
before-----in: 0, out: 2
Input your message:303132
Products:303132
product finish. semval[0](mutex)=[1] semval[1](full)=[9] semval[2](empty)=[1]
product in=[1]
after-----in: 1, out: 2
Please choose one option:
1. Produce a product; 2. Exit; 3. Delete Sharing Momery and exit
1
product before. semval[0](mutex)=[1] semval[1](full)=[9] semval[2](empty)=[1]
before-----in: 1, out: 2
Input your message:333435
Products:333435
product finish. semval[0](mutex)=[1] semval[1](full)=[10] semval[2](empty)=[0]
product in=[2]
after-----in: 2, out: 2
Please choose one option:
1. Produce a product; 2. Exit; 3. Delete Sharing Momery and exit
1
product before. semval[0](mutex)=[1] semval[1](full)=[10] semval[2](empty)=[0]
before-----in: 2, out: 2
Please wait consumer...
```

若消费者全部消费，会提示等待生产者生产；



```
consume before. semval[0](mutex)=[1] semval[1](full)=[7] semval[2](empty)=[3]
before-----in: 2, out: 5
consume finish. semval[0](mutex)=[1] semval[1](full)=[6] semval[2](empty)=[4]
product out=[6]
after-----in: 2, out: 6
Please choose one option:
1. Consume a product; 2. Exit; 3. Delete Sharing Momery and exit
1
consume before. semval[0](mutex)=[1] semval[1](full)=[6] semval[2](empty)=[4]
before-----in: 2, out: 6
consume finish. semval[0](mutex)=[1] semval[1](full)=[5] semval[2](empty)=[5]
product out=[7]
after-----in: 2, out: 7
Please choose one option:
1. Consume a product; 2. Exit; 3. Delete Sharing Momery and exit
1
consume before. semval[0](mutex)=[1] semval[1](full)=[5] semval[2](empty)=[5]
before-----in: 2, out: 7
consume finish. semval[0](mutex)=[1] semval[1](full)=[4] semval[2](empty)=[6]
product out=[8]
after-----in: 2, out: 8
Please choose one option:
1. Consume a product; 2. Exit; 3. Delete Sharing Momery and exit
```

```
after-----in: 2, out: 0
Please choose one option:
1. Consume a product; 2. Exit; 3. Delete Sharing Momery and exit
1
consume before. semval[0](mutex)=[1] semval[1](full)=[2] semval[2](empty)=[8]
before-----in: 2, out: 0
consume finish. semval[0](mutex)=[1] semval[1](full)=[1] semval[2](empty)=[9]
product out=[1]
after-----in: 2, out: 1
Please choose one option:
1. Consume a product; 2. Exit; 3. Delete Sharing Momery and exit
1
consume before. semval[0](mutex)=[1] semval[1](full)=[1] semval[2](empty)=[9]
before-----in: 2, out: 1
consume finish. semval[0](mutex)=[1] semval[1](full)=[0] semval[2](empty)=[10]
product out=[2]
after-----in: 2, out: 2
Please choose one option:
1. Consume a product; 2. Exit; 3. Delete Sharing Momery and exit
1
consume before. semval[0](mutex)=[1] semval[1](full)=[0] semval[2](empty)=[10]
before-----in: 2, out: 2
Please wait producer...
```

## 实验思考