浙大城市学院实验报告

- 课程名称：操作系统原理实验
- 实验项目名称：实验八 进程通信——通信量
- 学生姓名：徐彬涵
- 专业班级：软件工程2003
- 学号：32001272
- 实验成绩：
- 指导老师：胡隽
- 日期：2022/05/04

# 实验目的

1. 了解 Linux 系统的进程间通信 (IPC)；
2. 理解 Linux 关于信号量的概念；
3. 掌握 Linux 支持system V 信号量的系统调用；
4. 巩固进程同步概念。

# 实验内容

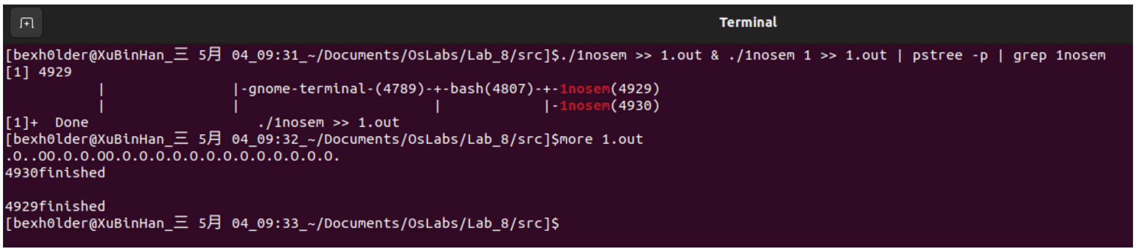
1. 在多个进程通过共享内存进行通信时，使用信号量进行同步控制
2. 使用系统调用：semget()、semop()、semtel()。

# 实验步骤

1nosem.c

两个进程并发执行时，交错输出内容

1. 观察进程并发执行结果，理解输出内容交替的原因。



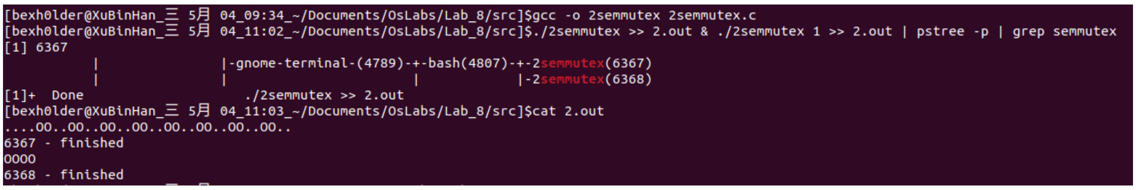
```
[bexh0lder@XuBinHan_三 5月 04_09:31_~/Documents/OsLabs/Lab_8/src]$./1nosem >> 1.out & ./1nosem 1 >> 1.out | pstree -p | grep 1nosem
[1] 4929
           |                |-gnome-terminal-(4789)-+-bash(4807)-+-1nosem(4929)
           |                |                        |            |-1nosem(4930)
[1]+  Done                   ./1nosem >> 1.out
[bexh0lder@XuBinHan_三 5月 04_09:32_~/Documents/OsLabs/Lab_8/src]$more 1.out
.O..OO.O.O.OO.O.O.O.O.O.O.O.O.O.O.O.O.O.
4930finished

4929finished
[bexh0lder@XuBinHan_三 5月 04_09:33_~/Documents/OsLabs/Lab_8/src]$
```

两个进程并发执行，同时往1.txt内写入内容，进程争取cpu资源,争取到就先输出，从而出现输出内容交替

2semmutex.c

修改例程1，使用System V 信号进行互斥控制

1. 和例程1比较，观察输出的变化，体会互斥信号量的作用



```
[bexh0lder@XuBinHan_三 5月 04_09:34_~/Documents/OsLabs/Lab_8/src]$gcc -o 2semmutex 2semmutex.c
[bexh0lder@XuBinHan_三 5月 04_11:02_~/Documents/OsLabs/Lab_8/src]$./2semmutex >> 2.out & ./2semmutex 1 >> 2.out | pstree -p | grep semmutex
[1] 6367
           |                |-gnome-terminal-(4789)-+-bash(4807)-+-2semmutex(6367)
           |                |                        |            |-2semmutex(6368)
[1]+  Done                   ./2semmutex >> 2.out
[bexh0lder@XuBinHan_三 5月 04_11:03_~/Documents/OsLabs/Lab_8/src]$cat 2.out
....OO..OO..OO..OO..OO..OO..OO..OO..
6367 - finished
OOOO
6368 - finished
```

2. 比较System V信号量和POSIX信号量的异同
   1. POSIX信号量常用于线程；system v信号量常用于进程的同步。
   2. 从使用的角度，System V 信号量的使用比较复杂，而 POSIX 信号量使用起来相对简单。
   3. 对 POSIX 来说，信号量是个非负整数。而 System V 信号量则是一个或多个信号量的集合，它对应的是一个信号量结构体，这个结构体是为 System V IPC 服务的，信号量只不过是它的一部分。
   4. Posix信号量是基于内存的，即信号量值是放在共享内存中的，它是由可能与文件系统中的路径名对应的名字来标识的。而System v信号量则是基于内核的，它放在内核里面。
   5. POSIX 信号量的头文件是 <semaphore.h>，而 System V 信号量的头文件是 <sys/sem.h>。
   6. Posix还有有名信号量，一般用于进程同步, 有名信号量是内核持续的。

3reader_writer.c

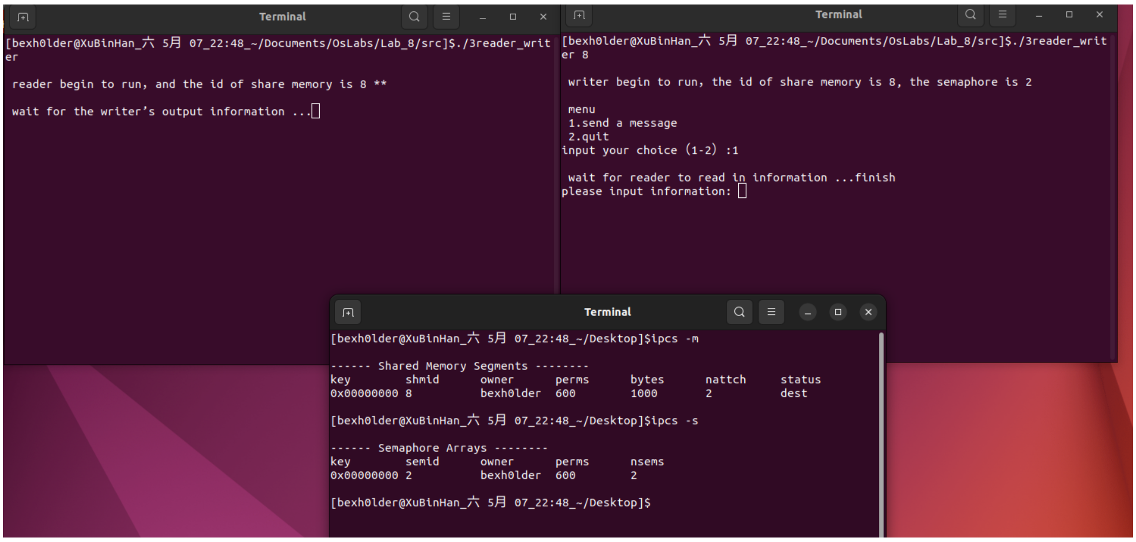
实现用System V信号量对两个（或多个）通过共享内存传递信息的并发进程来进行同步控制

1. 理解共享内存和信号量相关的系统调用；

   - `semget` 得到一个信号量集标识符或创建一个信号量集对象
   - `semop` 完成对信号量的P操作或V操作
   - `semctl` 得到一个信号量集标识符或创建一个信号量集对象
   - `shmget` 得到一个共享内存标识符或创建一个共享内存并返回共享内存标识符

- `shmat` 连接共享内存标识符为shmid的共享内存，连接成功后把共享内存区对象映射到调用进程的地址空间，随后像本地空间一样访问
- `shmctl` 完成对共享内存的控制

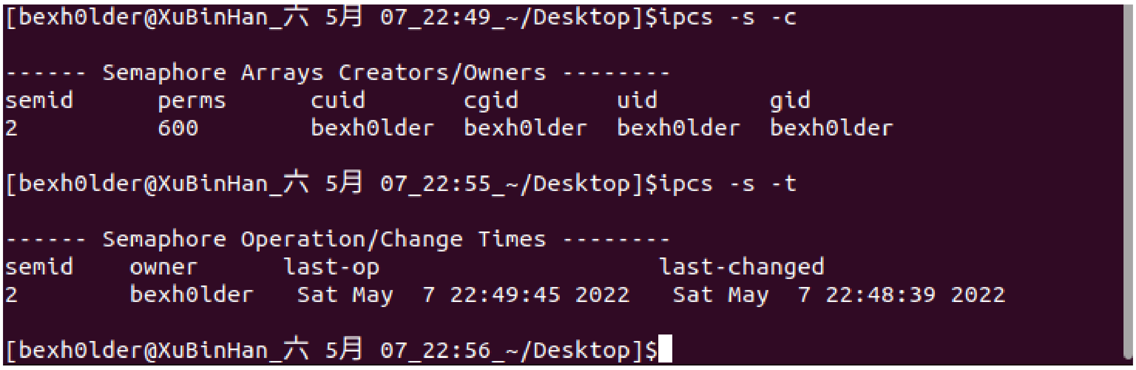
2. 用ipcs观察和理解共享内存的信号量信息；



```
[bexh0lder@XuBinHan_六 5月 07_22:48_~/Documents/OsLabs/Lab_8/src]$./3reader_writ
er

 reader begin to run, and the id of share memory is 8 **

 wait for the writer’s output information ...
```

```
[bexh0lder@XuBinHan_六 5月 07_22:48_~/Documents/OsLabs/Lab_8/src]$./3reader_writ
er 8

 writer begin to run, the id of share memory is 8, the semaphore is 2

 menu
 1.send a message
 2.quit
input your choice (1-2) :1

 wait for reader to read in information ...finish
please input information: 
```

```
[bexh0lder@XuBinHan_六 5月 07_22:48_~/Desktop]$ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 8          bexh0lder  600        1000       2          dest

[bexh0lder@XuBinHan_六 5月 07_22:48_~/Desktop]$ipcs -s

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0x00000000 2          bexh0lder  600        2

[bexh0lder@XuBinHan_六 5月 07_22:48_~/Desktop]$
```

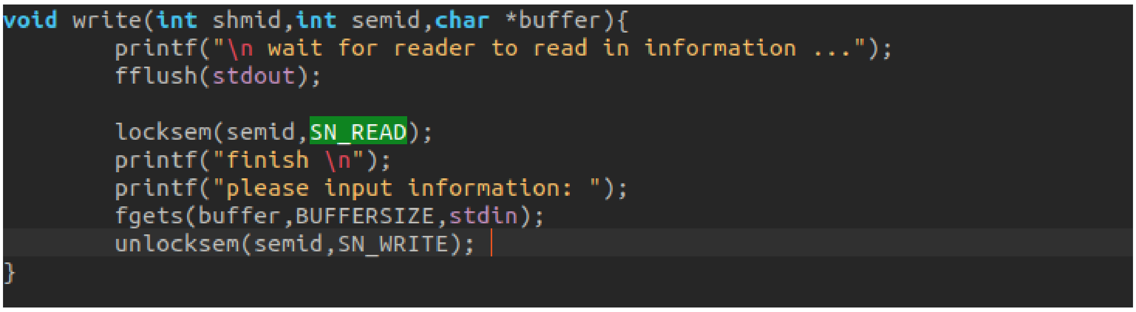


```
[bexh0lder@XuBinHan_六 5月 07_22:49_~/Desktop]$ipcs -s -c

------ Semaphore Arrays Creators/Owners --------
semid      perms      cuid       cgid       uid        gid
2          600        bexh0lder  bexh0lder  bexh0lder  bexh0lder

[bexh0lder@XuBinHan_六 5月 07_22:55_~/Desktop]$ipcs -s -t

------ Semaphore Operation/Change Times --------
semid    owner      last-op                    last-changed
2        bexh0lder   Sat May  7 22:49:45 2022   Sat May  7 22:48:39 2022

[bexh0lder@XuBinHan_六 5月 07_22:56_~/Desktop]$
```

通过ipcs命令我们可以知道在 `3reader_writer` 运行时在系统中存在一个共享内存和两个共享信号量， `ipcs -s` 运行后显示 `nesms` 为2表示有两个信号量， `perms` 为信号量集的权限

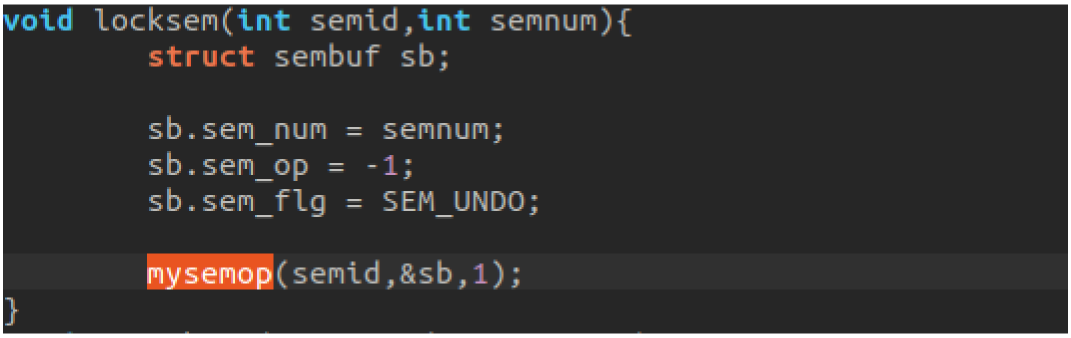
3. 尝试运行多个writer，能否正确同步？请分析代码解释你的判断。

可以正确同步

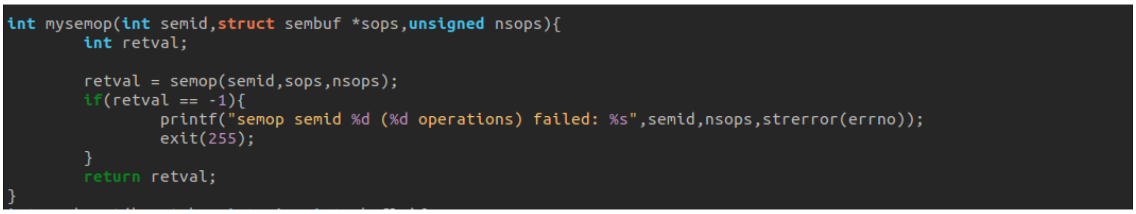


```
void write(int shmid,int semid,char *buffer){
        printf("\n wait for reader to read in information ...");
        fflush(stdout);

        locksem(semid,SN_READ);
        printf("finish \n");
        printf("please input information: ");
        fgets(buffer,BUFFERSIZE,stdin);
        unlocksem(semid,SN_WRITE);
}
```

`writer` 进程执行后会调用 `locksem` 函数,而 `locksem` 包括了 `mysemop` 函数控制了对 `SN_WRITE` 信号量的PV操作，从而实现进程间的并发互斥，实现正确同步



```c
void locksem(int semid,int semnum){
        struct sembuf sb;

        sb.sem_num = semnum;
        sb.sem_op = -1;
        sb.sem_flg = SEM_UNDO;

        mysemop(semid,&sb,1);
}
```



```c
int mysemop(int semid,struct sembuf *sops,unsigned nsops){
        int retval;

        retval = semop(semid,sops,nsops);
        if(retval == -1){
                printf("semop semid %d (%d operations) failed: %s",semid,nsops,strerror(errno));
                exit(255);
        }
        return retval;
}
```

编程题

4con.c

```c
/* Our first program is a consumer. After the headers the shared memory segment
 (the size of our shared memory structure) is created with a call to shmget,
 with the IPC_CREAT bit specified. */

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <errno.h>
#include <signal.h>

#define TEXT_SZ 2048

/* The union for semctl may or may not be defined for us.This code,defined
 in linux's semctl() manpage,is the proper way to attain it if necessary */
#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; /* value for SETVAL */
    struct semid_ds *buf; /* buffer for IPC_STAT,IPC_SET */
    unsigned short int *array; /* array for GETALL,SETALL */
    struct seminfo *__buf; /* buffer for IPC_INFO */
};
#endif
#define SHMDATASIZE 1000
#define BUFFERSIZE (SHMDATASIZE - sizeof(int))
#define SN_READ 0
#define SN_WRITE 1
int Semid = 0; /* 用于最后删除这个信号量 */
void delete(void);
void sigdelete(int signum);
void locksem(int semid,int semnum);
void unlocksem(int semid,int semnum);
void waitzero(int semid,int semnum);
int mysemget(key_t key,int nsems,int semflg);
int mysemctl(int semid,int semnum,int cmd,union semun arg);
int mysemop(int semid,struct sembuf *sops,unsigned nsops);
int myshmget(key_t key,int size,int shmflg);
void *myshmat(int shmid,const void *shmaddr,int shmflg);
int myshmctl(int shmid,int cmd,struct shmid_ds *buf);

int main()
{
    union semun sunion;
    int semid,shmid;
    void *shmdata;
    char *buffer;

    /* 首先：我们要创建信号量 */
    semid = mysemget(IPC_PRIVATE,2,SHM_R|SHM_W);
    Semid = semid;

    /* 在进程离开时，删除信号量 */
    atexit(&delete); //进程退出后执行delete函数
    signal(SIGINT,&sigdelete);

    /* 信号量 SN_READ 初始化为 1(锁定)，SN_WRITE 初始化为 0（未锁定）*/
    sunion.val = 1;
    mysemctl(semid,SN_READ,SETVAL,sunion);

    sunion.val = 0;
    mysemctl(semid,SN_WRITE,SETVAL,sunion);

    /* 现在创建一块共享内存 */
    shmid = myshmget(IPC_PRIVATE,SHMDATASIZE,IPC_CREAT|SHM_R|SHM_W);

    /* 将该共享内存映射到进程的虚存空间 */
    shmdata = shmat(shmid,0,0);

    /* 将该共享内存标志为已销毁的，这样在使用完毕后，将被自动销毁*/
    shmctl(shmid,IPC_RMID,NULL);

    /* 将信号量的标识符写入共享内存，以通知其它的进程 */

    *(int *)shmdata = semid;

    buffer = shmdata + sizeof(int);

    printf("\n consumer begin to run， and the id of share memory is %d ** \n",shmid);

    /************************************************************
    reader 的主循环
    ************************************************************/
    while(1){
        locksem(semid,SN_WRITE);
        printf("You wrote：%s \n",buffer);
        //sleep( rand() % 4 );
        unlocksem(semid,SN_READ);
    }
}
void delete(void){
    printf("\n quit; delete the semaphore %d \n",Semid);

    /* 删除信号量 */
    if(semctl(Semid,0,IPC_RMID,0) == -1){
        printf("Error releasing semaphore.\n");
    }
}
void sigdelete(int signum){
    /* Calling exit will conveniently trigger the normal delete item. */
    exit(0);
}
```

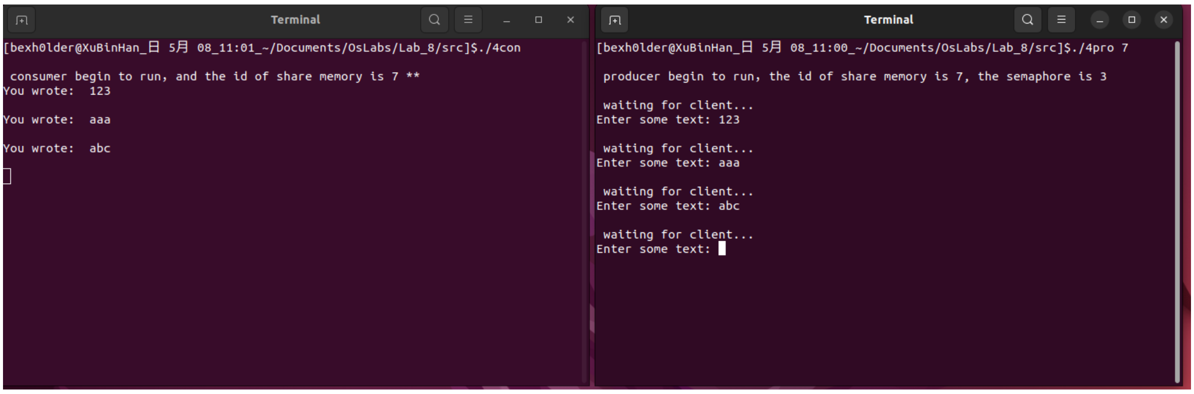
```c
void locksem(int semid,int semnum){
    struct sembuf sb;

    sb.sem_num = semnum;
    sb.sem_op = -1;
    sb.sem_flg = SEM_UNDO;

    mysemop(semid,&sb,1);
}
void unlocksem(int semid,int semnum){
    struct sembuf sb;

    sb.sem_num = semnum;
    sb.sem_op = 1;
    sb.sem_flg = SEM_UNDO;

    mysemop(semid,&sb,1);
}
void waitzero(int semid,int semnum){
    struct sembuf sb;

    sb.sem_num = semnum;
    sb.sem_op = 0;
    sb.sem_flg = 0; /* No modification so no need to undo */
    mysemop(semid,&sb,1);
}
int mysemget(key_t key,int nsems,int semflg){
    int retval;

    retval = semget(key,nsems,semflg);
    if(retval == -1){
        printf("semget key %d,nsems %d failed: %s ",key,nsems,strerror(errno));
        exit(255);
    }
    return retval;
}
int mysemctl(int semid,int semnum,int cmd,union semun arg){
    int retval;

    retval = semctl(semid,semnum,cmd,arg);
    if(retval == -1){
        printf("semctl semid %d,semnum %d,cmd %d failed: %s",semid,semnum,cmd,strerror(errno));
        exit(255);
    }
    return retval;
}

int mysemop(int semid,struct sembuf *sops,unsigned nsops){
    int retval;

    retval = semop(semid,sops,nsops);
    if(retval == -1){
        printf("semop semid %d (%d operations) failed: %s",semid,nsops,strerror(errno));
        exit(255);
    }
    return retval;
}
int myshmget(key_t key,int size,int shmflg){
    int retval;

    retval = shmget(key,size,shmflg);
    if(retval == -1){
        printf("shmget key %d,size %d failed: %s",key,size,strerror(errno));
        exit(255);
    }
    return retval;
}
void *myshmat(int shmid,const void *shmaddr,int shmflg){
    void *retval;

    retval = shmat(shmid,shmaddr,shmflg);
    if(retval == (void*) -1){
        printf("shmat shmid %d failed: %s",shmid,strerror(errno));
        exit(255);
    }
    return retval;
}
int myshmctl(int shmid,int cmd,struct shmid_ds *buf){
    int retval;

    retval = shmctl(shmid,cmd,buf);
    if(retval == -1){
        printf("shmctl shmid %d,cmd %d failed: %s",shmid,cmd,strerror(errno));
        exit(255);
    }
    return retval;
}
```

4pro.c

```
/* The second program is the producer and allows us to enter data for consumers.*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <errno.h>
#include <signal.h>

#define TEXT_SZ 2048

/* The union for semctl may or may not be defined for us.This code,defined
 in linux's semctl() manpage,is the proper way to attain it if necessary */
#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; /* value for SETVAL */
    struct semid_ds *buf; /* buffer for IPC_STAT,IPC_SET */
    unsigned short int *array; /* array for GETALL,SETALL */
    struct seminfo *__buf; /* buffer for IPC_INFO */
};
#endif
#define SHMDATASIZE 1000
#define BUFFERSIZE (SHMDATASIZE - sizeof(int))
#define SN_READ 0
#define SN_WRITE 1
```

```c
int Semid = 0; /* 用于最后删除这个信号量 */
void delete(void);
void sigdelete(int signum);
void locksem(int semid,int semnum);
void unlocksem(int semid,int semnum);
void waitzero(int semid,int semnum);
void write(int shmid,int semid,char *buffer);
int mysemget(key_t key,int nsems,int semflg);
int mysemctl(int semid,int semnum,int cmd,union semun arg);
int mysemop(int semid,struct sembuf *sops,unsigned nsops);
int myshmget(key_t key,int size,int shmflg);
void *myshmat(int shmid,const void *shmaddr,int shmflg);
int myshmctl(int shmid,int cmd,struct shmid_ds *buf);

int main(int argc,char *argv[])
{
    int shmid;
    if(argc < 2){
        printf("Plz use .\\4pro [shmid]\n");
    }else{
        shmid = atoi(argv[1]);
    }
        int semid;
    void *shmdata;
    char *buffer;

    /* 将该共享内存映射到进程的虚存空间 */
    shmdata = myshmat(shmid,0,0);

    semid = *(int *)shmdata;
    buffer = shmdata + sizeof(int);

    printf(" \n producer begin to run， the id of share memory is %d, the semaphore is %d \n",shmid,semid);
    /*************************************************************
    writer 的主循环
    *************************************************************/
    while(1){
        /*char input[3];

        printf("\n menu \n 1.send a message \n");
        printf(" 2.quit \n");
        printf("input your choice （1-2）:");

        fgets(input,sizeof(input),stdin);

        switch(input[0]){
            case '1':write(shmid,semid,buffer);break;
            case '2':exit(0);break;
        }*/
        write(shmid,semid,buffer);
    }
}
void locksem(int semid,int semnum){
    struct sembuf sb;

    sb.sem_num = semnum;
    sb.sem_op = -1;
    sb.sem_flg = SEM_UNDO;

    mysemop(semid,&sb,1);
}
void unlocksem(int semid,int semnum){
    struct sembuf sb;

    sb.sem_num = semnum;
    sb.sem_op = 1;
    sb.sem_flg = SEM_UNDO;

    mysemop(semid,&sb,1);
}
void waitzero(int semid,int semnum){
    struct sembuf sb;

    sb.sem_num = semnum;
    sb.sem_op = 0;
    sb.sem_flg = 0; /* No modification so no need to undo */
    mysemop(semid,&sb,1);
}
void write(int shmid,int semid,char *buffer){
    printf("\n waiting for client...\n");
    fflush(stdout);

    locksem(semid,SN_READ);
```

```c
    //printf("finish \n");
    printf("Enter some text: ");
    fgets(buffer,BUFFERSIZE,stdin);
    unlocksem(semid,SN_WRITE);
}
int mysemget(key_t key,int nsems,int semflg){
    int retval;

    retval = semget(key,nsems,semflg);
    if(retval == -1){
        printf("semget key %d,nsems %d failed: %s ",key,nsems,strerror(errno));
        exit(255);
    }
    return retval;
}
int mysemctl(int semid,int semnum,int cmd,union semun arg){
    int retval;

    retval = semctl(semid,semnum,cmd,arg);
    if(retval == -1){
        printf("semctl semid %d,semnum %d,cmd %d failed: %s",semid,semnum,cmd,strerror(errno));
        exit(255);
    }
    return retval;
}

int mysemop(int semid,struct sembuf *sops,unsigned nsops){
    int retval;

    retval = semop(semid,sops,nsops);
    if(retval == -1){
        printf("semop semid %d (%d operations) failed: %s",semid,nsops,strerror(errno));
        exit(255);
    }
    return retval;
}
int myshmget(key_t key,int size,int shmflg){
    int retval;

    retval = shmget(key,size,shmflg);
    if(retval == -1){
        printf("shmget key %d,size %d failed: %s",key,size,strerror(errno));
        exit(255);
    }
    return retval;
}
void *myshmat(int shmid,const void *shmaddr,int shmflg){
    void *retval;

    retval = shmat(shmid,shmaddr,shmflg);
    if(retval == (void*) -1){
        printf("shmat shmid %d failed: %s",shmid,strerror(errno));
        exit(255);
    }
    return retval;
}
int myshmctl(int shmid,int cmd,struct shmid_ds *buf){
    int retval;

    retval = shmctl(shmid,cmd,buf);
    if(retval == -1){
        printf("shmctl shmid %d,cmd %d failed: %s",shmid,cmd,strerror(errno));
        exit(255);
    }
    return retval;
}
```



```
[bexh0lder@XuBinHan_日 5月 08_11:01_~/Documents/OsLabs/Lab_8/src]$./4con

 consumer begin to run, and the id of share memory is 7 **
You wrote:  123

You wrote:  aaa

You wrote:  abc

```

```
[bexh0lder@XuBinHan_日 5月 08_11:00_~/Documents/OsLabs/Lab_8/src]$./4pro 7

 producer begin to run, the id of share memory is 7, the semaphore is 3

 waiting for client...
Enter some text: 123

 waiting for client...
Enter some text: aaa

 waiting for client...
Enter some text: abc

 waiting for client...
Enter some text: 
```