

# 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()、semctl()。

## # 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。



- **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；

The image shows three terminal windows. The top-left window shows the output of a program where a reader begins to run and waits for the writer's output. The top-right window shows the output of a program where a writer begins to run, the id of share memory is 8, and the semaphore is 2. The bottom window shows the output of the `ipcs` command, displaying shared memory segments and semaphore arrays.

```
[bexholder@XuBinHan_六 5月 07_22:48_~/Documents/OsLabs/Lab_8/src]$ ./3reader_writer
reader begin to run, and the id of share memory is 8 **
wait for the writer's output information ...[]

[bexholder@XuBinHan_六 5月 07_22:48_~/Documents/OsLabs/Lab_8/src]$ ./3reader_writer 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: []

[bexholder@XuBinHan_六 5月 07_22:48_~/Desktop]$ ipcs -m
----- Shared Memory Segments -----
key      shmid    owner      perms      bytes      nattch     status
0x00000000 8          bexholder  600        1000       2          dest

[bexholder@XuBinHan_六 5月 07_22:48_~/Desktop]$ ipcs -s
----- Semaphore Arrays -----
key      semid    owner      perms      nsems
0x00000000 2        bexholder  600        2

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

[bexholder@XuBinHan_六 5月 07_22:49_~/Desktop]$ ipcs -s -c
----- Semaphore Arrays Creators/Owners -----
semid      perms      cuid      cgid      uid      gid
2          600       bexholder bexholder bexholder bexholder

[bexholder@XuBinHan_六 5月 07_22:55_~/Desktop]$ ipcs -s -t
----- Semaphore Operation/Change Times -----
semid      owner      last-op      last-changed
2          bexholder  Sat May 7 22:49:45 2022  Sat May 7 22:48:39 2022

[bexholder@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

```

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);
}

```

```

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

```

1  /* Our first program is a consumer. After the headers the shared memory segment
2     (the size of our shared memory structure) is created with a call to shmget,
3     with the IPC_CREAT bit specified. */
4
5  #include <unistd.h>
6  #include <stdlib.h>
7  #include <stdio.h>
8  #include <string.h>
9  #include <sys/types.h>
10 #include <sys/ipc.h>
11 #include <sys/shm.h>
12 #include <sys/sem.h>
13 #include <errno.h>
14 #include <signal.h>
15
16 #define TEXT_SZ 2048
17
18 /* The union for semctl may or may not be defined for us. This code, defined
19    in linux's semctl() manpage, is the proper way to attain it if necessary */
20 #if defined (__GNU_LIBRARY__) && !defined (_SEM_SEMUN_UNDEFINED)
21     /* union semun is defined by including <sys/sem.h> */
22 #else
23     /* according to X/OPEN we have to define it ourselves */
24     union semun{
25         int val; /* value for SETVAL */
26         struct semid_ds *buf; /* buffer for IPC_STAT,IPC_SET */

```

```

27     unsigned short int *array; /* array for GETALL,SETALL */
28     struct seminfo *__buf; /* buffer for IPC_INFO */
29 };
30 #endif
31 #define SHMDATASIZE 1000
32 #define BUFFERSIZE (SHMDATASIZE - sizeof(int))
33 #define SN_READ 0
34 #define SN_WRITE 1
35 int Semid = 0; /* 用于最后删除这个信号量 */
36 void delete(void);
37 void sigdelete(int signum);
38 void locksem(int semid,int semnum);
39 void unlocksem(int semid,int semnum);
40 void waitzero(int semid,int semnum);
41 int mysemget(key_t key,int nsems,int semflg);
42 int mysemctl(int semid,int semnum,int cmd,union semun arg);
43 int mysemop(int semid,struct sembuf *sops,unsigned nsops);
44 int myshmget(key_t key,int size,int shmflg);
45 void *myshmat(int shmid,const void *shmaddr,int shmflg);
46 int myshmctl(int shmid,int cmd,struct shmid_ds *buf);
47
48 int main()
49 {
50     union semun sunion;
51     int semid,shmid;
52     void *shmdata;
53     char *buffer;
54
55     /* 首先：我们要创建信号量 */
56     semid = mysemget(IPC_PRIVATE,2,SHM_R|SHM_W);
57     Semid = semid;
58
59     /* 在进程离开时，删除信号量 */
60     atexit(&delete); //进程退出后执行delete函数
61     signal(SIGINT,&sigdelete);
62
63     /* 信号量 SN_READ 初始化为 1(锁定)，SN_WRITE 初始化为 0 (未锁定) */
64     sunion.val = 1;
65     mysemctl(semid,SN_READ,SETVAL,sunion);
66
67     sunion.val = 0;
68     mysemctl(semid,SN_WRITE,SETVAL,sunion);

```

```

69
70     /* 现在创建一块共享内存 */
71     shmids =
myshmget(IPC_PRIVATE,SHMDATASIZE,IPC_CREAT|SHM_R|SHM_W);
72
73     /* 将该共享内存映射到进程的虚存空间 */
74     shmdat = shmat(shmid,0,0);
75
76     /* 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*/
77     shmctl(shmid,IPC_RMID,NULL);
78
79     /* 将信号量的标识符写入共享内存，以通知其它的进程 */
80
81     *(int *)shmdat = semid;
82
83     buffer = shmdat + sizeof(int);
84
85     printf("\n consumer begin to run, and the id of share memory is %d **
\n",shmid);
86
87     /******
88     reader 的主循环
89     *****/
90     while(1){
91         locksem(semid,SN_WRITE);
92         printf("You wrote: %s \n",buffer);
93         //sleep( rand() % 4 );
94         unlocksem(semid,SN_READ);
95     }
96 }
97 void delete(void){
98     printf("\n quit; delete the semaphore %d \n",Semid);
99
100     /* 删除信号量 */
101     if(semctl(Semid,0,IPC_RMID,0) == -1){
102         printf("Error releasing semaphore.\n");
103     }
104 }
105 void sigdelete(int signum){
106     /* Calling exit will conveniently trigger the normal delete item. */
107     exit(0);
108 }

```

```
109 void locksem(int semid,int semnum){
110     struct sembuf sb;
111
112     sb.sem_num = semnum;
113     sb.sem_op = -1;
114     sb.sem_flg = SEM_UNDO;
115
116     mysemop(semid,&sb,1);
117 }
118 void unlocksem(int semid,int semnum){
119     struct sembuf sb;
120
121     sb.sem_num = semnum;
122     sb.sem_op = 1;
123     sb.sem_flg = SEM_UNDO;
124
125     mysemop(semid,&sb,1);
126 }
127 void waitzero(int semid,int semnum){
128     struct sembuf sb;
129
130     sb.sem_num = semnum;
131     sb.sem_op = 0;
132     sb.sem_flg = 0; /* No modification so no need to undo */
133     mysemop(semid,&sb,1);
134 }
135 int mysemget(key_t key,int nsems,int semflg){
136     int retval;
137
138     retval = semget(key,nsems,semflg);
139     if(retval == -1){
140         printf("semget key %d,nsems %d failed: %s ",key,nsems,strerror(errno));
141         exit(255);
142     }
143     return retval;
144 }
145 int mysemctl(int semid,int semnum,int cmd,union semun arg){
146     int retval;
147
148     retval = semctl(semid,semnum,cmd,arg);
149     if(retval == -1){
```

```

150     printf("semctl semid %d,semnum %d,cmd %d failed:
%s",semid,semnum,cmd,strerror(errno));
151     exit(255);
152 }
153 return retval;
154 }
155
156 int mysemop(int semid,struct sembuf *sops,unsigned nsops){
157     int retval;
158
159     retval = semop(semid,sops,nsops);
160     if(retval == -1){
161         printf("semop semid %d (%d operations) failed:
%s",semid,nsops,strerror(errno));
162         exit(255);
163     }
164     return retval;
165 }
166 int myshmget(key_t key,int size,int shmflg){
167     int retval;
168
169     retval = shmget(key,size,shmflg);
170     if(retval == -1){
171         printf("shmget key %d,size %d failed: %s",key,size,strerror(errno));
172         exit(255);
173     }
174     return retval;
175 }
176 void *myshmat(int shmid,const void *shmaddr,int shmflg){
177     void *retval;
178
179     retval = shmat(shmid,shmaddr,shmflg);
180     if(retval == (void*) -1){
181         printf("shmat shmid %d failed: %s",shmid,strerror(errno));
182         exit(255);
183     }
184     return retval;
185 }
186 int myshmctl(int shmid,int cmd,struct shmid_ds *buf){
187     int retval;
188
189     retval = shmctl(shmid,cmd,buf);

```



```

190     if(retval == -1){
191         printf("shmctl shmctl %d,cmd %d failed: %s",shmctl,cmd,strerror(errno));
192         exit(255);
193     }
194     return retval;
195 }

```

## 4pro.c

```

1  /* The second program is the producer and allows us to enter data for
2     consumers.*/
3
4  #include <stdlib.h>
5  #include <stdio.h>
6  #include <string.h>
7  #include <sys/types.h>
8  #include <sys/ipc.h>
9  #include <sys/shm.h>
10 #include <sys/sem.h>
11 #include <errno.h>
12 #include <signal.h>
13
14 #define TEXT_SZ 2048
15
16 /* The union for semctl may or may not be defined for us.This code,defined
17    in linux's semctl() manpage,is the proper way to attain it if necessary */
18 #if defined (__GNU_LIBRARY__)&& !defined (_SEM_SEMUN_UNDEFINED)
19     /* union semun is defined by including <sys/sem.h> */
20 #else
21     /* according to X/OPEN we have to define it ourselves */
22     union semun{
23         int val; /* value for SETVAL */
24         struct semid_ds *buf; /* buffer for IPC_STAT,IPC_SET */
25         unsigned short int *array; /* array for GETALL,SETALL */
26         struct seminfo *__buf; /* buffer for IPC_INFO */
27     };
28 #endif
29
30 #define SHMDATASIZE 1000
31 #define BUFFERSIZE (SHMDATASIZE - sizeof(int))
32 #define SN_READ 0
33 #define SN_WRITE 1

```

```

32 int Semid = 0; /* 用于最后删除这个信号量 */
33 void delete(void);
34 void sigdelete(int signum);
35 void locksem(int semid,int semnum);
36 void unlocksem(int semid,int semnum);
37 void waitzero(int semid,int semnum);
38 void write(int shmid,int semid,char *buffer);
39 int mysemget(key_t key,int nsems,int semflg);
40 int mysemctl(int semid,int semnum,int cmd,union semun arg);
41 int mysemop(int semid,struct sembuf *sops,unsigned nsops);
42 int myshmget(key_t key,int size,int shmflg);
43 void *myshmat(int shmid,const void *shmaddr,int shmflg);
44 int myshmctl(int shmid,int cmd,struct shmid_ds *buf);
45
46 int main(int argc,char *argv[])
47 {
48     int shmid;
49     if(argc < 2){
50         printf("Plz use .\\4pro [shmid]\n");
51     }else{
52         shmid = atoi(argv[1]);
53     }
54     int semid;
55     void *shmdata;
56     char *buffer;
57
58     /* 将该共享内存映射到进程的虚存空间 */
59     shmdata = myshmat(shmid,0,0);
60
61     semid = *(int *)shmdata;
62     buffer = shmdata + sizeof(int);
63
64     printf("\n producer begin to run , the id of share memory is %d, the semaphore
is %d \n",shmid,semid);
65     /******
66     writer 的主循环
67     *****/
68     while(1){
69         /*char input[3];
70
71         printf("\n menu \n 1.send a message \n");
72         printf(" 2.quit \n");

```

```
73     printf("input your choice (1-2) :");
74
75     fgets(input,sizeof(input),stdin);
76
77     switch(input[0]){
78         case '1':write(shmid,semid,buffer);break;
79         case '2':exit(0);break;
80     }*/
81     write(shmid,semid,buffer);
82 }
83 }
84 void locksem(int semid,int semnum){
85     struct sembuf sb;
86
87     sb.sem_num = semnum;
88     sb.sem_op = -1;
89     sb.sem_flg = SEM_UNDO;
90
91     mysemop(semid,&sb,1);
92 }
93 void unlocksem(int semid,int semnum){
94     struct sembuf sb;
95
96     sb.sem_num = semnum;
97     sb.sem_op = 1;
98     sb.sem_flg = SEM_UNDO;
99
100     mysemop(semid,&sb,1);
101 }
102 void waitzero(int semid,int semnum){
103     struct sembuf sb;
104
105     sb.sem_num = semnum;
106     sb.sem_op = 0;
107     sb.sem_flg = 0; /* No modification so no need to undo */
108     mysemop(semid,&sb,1);
109 }
110 void write(int shmid,int semid,char *buffer){
111     printf("\n waiting for client...\n");
112     fflush(stdout);
113
114     locksem(semid,SN_READ);
```

```

115     //printf("finish \n");
116     printf("Enter some text: ");
117     fgets(buffer,BUFFERSIZE,stdin);
118     unlocksem(semid,SN_WRITE);
119 }
120 int mysemget(key_t key,int nsems,int semflg){
121     int retval;
122
123     retval = semget(key,nsems,semflg);
124     if(retval == -1){
125         printf("semget key %d,nsems %d failed: %s ",key,nsems,strerror(errno));
126         exit(255);
127     }
128     return retval;
129 }
130 int mysemctl(int semid,int semnum,int cmd,union semun arg){
131     int retval;
132
133     retval = semctl(semid,semnum,cmd,arg);
134     if(retval == -1){
135         printf("semctl semid %d,semnum %d,cmd %d failed:
136 %s",semid,semnum,cmd,strerror(errno));
137         exit(255);
138     }
139     return retval;
140 }
141 int mysemop(int semid,struct sembuf *sops,unsigned nsops){
142     int retval;
143
144     retval = semop(semid,sops,nsops);
145     if(retval == -1){
146         printf("semop semid %d (%d operations) failed:
147 %s",semid,nsops,strerror(errno));
148         exit(255);
149     }
150     return retval;
151 }
152 int myshmget(key_t key,int size,int shmflg){
153     int retval;
154
155     retval = shmget(key,size,shmflg);

```

```

155     if(retval == -1){
156         printf("shmget key %d,size %d failed: %s",key,size,strerror(errno));
157         exit(255);
158     }
159     return retval;
160 }
161 void *myshmat(int shmid,const void *shmaddr,int shmflg){
162     void *retval;
163
164     retval = shmat(shmid,shmaddr,shmflg);
165     if(retval == (void*) -1){
166         printf("shmat shmid %d failed: %s",shmid,strerror(errno));
167         exit(255);
168     }
169     return retval;
170 }
171 int myshmctl(int shmid,int cmd,struct shmid_ds *buf){
172     int retval;
173
174     retval = shmctl(shmid,cmd,buf);
175     if(retval == -1){
176         printf("shmctl shmid %d,cmd %d failed: %s",shmid,cmd,strerror(errno));
177         exit(255);
178     }
179     return retval;
180 }

```

The image shows two terminal windows side-by-side, both with a dark purple background. The left terminal window shows the output of a consumer process, and the right terminal window shows the output of a producer process.

**Left Terminal Window:**

```

[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

```

**Right Terminal Window:**

```

[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:

```