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：实验七 进程通信——共享内存
- 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：2022/04/29

实验目的

1. 了解 Linux 系统的进程间通信 (IPC)：
2. 理解 Linux 关于共享内存的概念；
3. 掌握 Linux 支持进程间内存共享的系统调用；
4. 巩固进程同步概念。

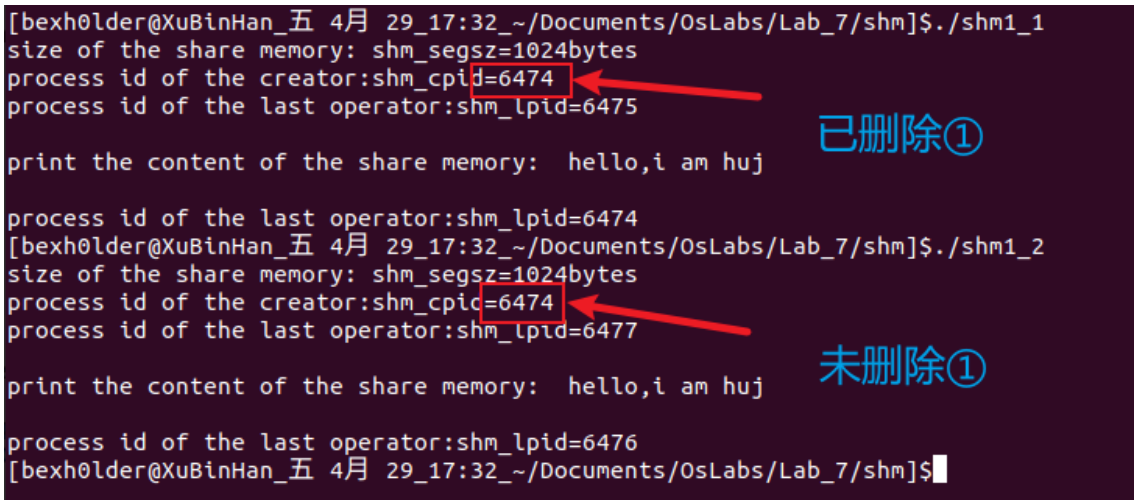
实验内容

1. 掌握进程通信 IC；
2. 使用和共享内存相关的系统调用：shmget、shmat、shmdt、shmctl；
3. 使用有关 IC 的 guanl 命令：ipcs 、ipcrm。

实验步骤

shm1

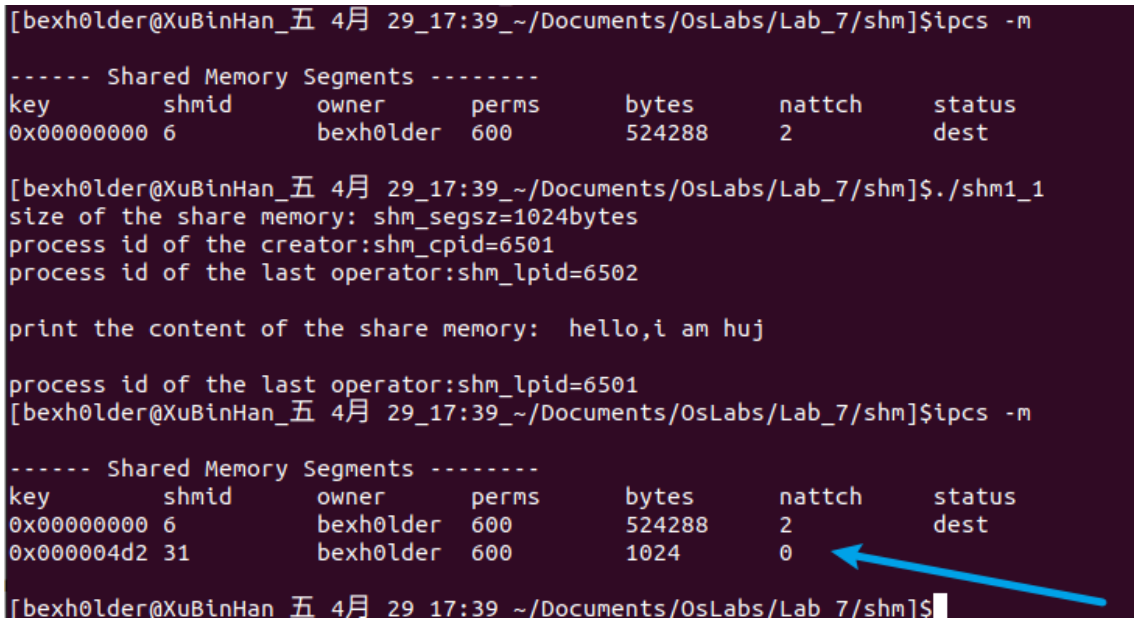
1. 删除①，重新编译运行，观察输出内容，理解输出不变的部分和改变的部分



- shm1_1为删除①后的程序
- shm1_2为未删除①的程序

可见在删除①前后，输出的创建共享区域的进程的进程ID号是不变的，原因在于语句①是用于删除共享内存空间的，删除①语句后，共享内存在程序运行完之后并没有被回收，导致再次运行程序时，由于key指向的共享内存区域已经存在，所以会直接调用之前创建的共享内存区域，而不是重新创建，从而使紧接着shm1_1运行的shm1_2使用的仍是6457进程创建的共享内存，输出相同的创建者进程号

2. 删除①，重新编译运行程序后，使用ipcs命令观察关于共享内存的相关信息；



```
[bexh0lder@XuBinHan_五 4月 29_17:40_~/Documents/OsLabs/Lab_7/shm]$ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 6          bexh0lder  600        524288     2          dest

[bexh0lder@XuBinHan_五 4月 29_17:40_~/Documents/OsLabs/Lab_7/shm]$./shm1_2
size of the share memory: shm_segsz=1024bytes
process id of the creator:shm_cpid=6509
process id of the last operator:shm_lpid=6510

print the content of the share memory:  hello,i am huj

process id of the last operator:shm_lpid=6509
[bexh0lder@XuBinHan_五 4月 29_17:40_~/Documents/OsLabs/Lab_7/shm]$ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 6          bexh0lder  600        524288     2          dest

[bexh0lder@XuBinHan_五 4月 29_17:40_~/Documents/OsLabs/Lab_7/shm]$
```

- shm1_1为删除①后的程序
- shm1_2为未删除①的程序

可见shm1_1运行前后多出一块共享内存，原因在于语句①就是用于删除共享内存空间的，删除之后即便程序结束共享内存空间也不会被删除仍然存在。

shm2

1. 两个并发进程的启动顺序对程序运行有没有影响？

   - 先启动shm2w.c，再启动shm2r.c

     ```
     [bexh0lder@XuBinHan_五 4月 29_19:11_~/Documents/OsLabs/Lab_7/shm]$./shm2w

     [bexh0lder@XuBinHan_五 4月 29_19:11_~/Documents/OsLabs/Lab_7/shm]$./shm2r
     abcdefghijklmnopqrstuvwxyz

     [bexh0lder@XuBinHan_五 4月 29_19:11_~/Documents/OsLabs/Lab_7/shm]$
     ```

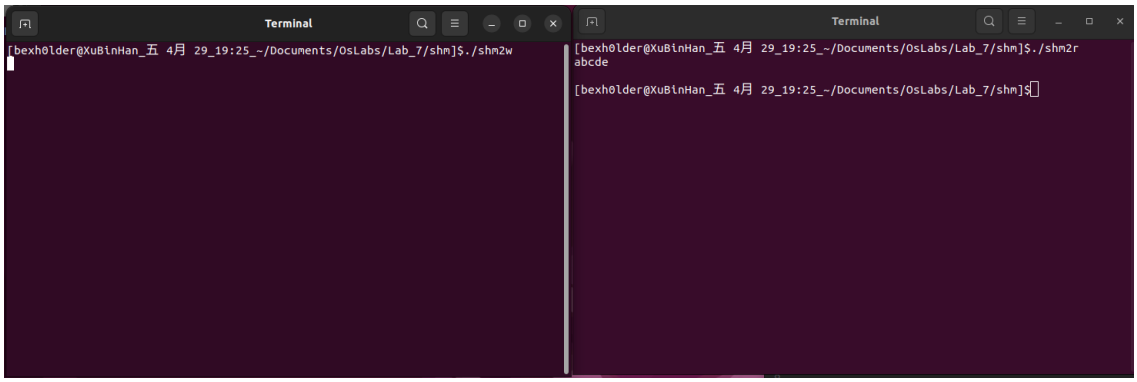
   - 先启动shm2r.c，再启动shm2w.c

     ```
     [bexh0lder@XuBinHan_五 4月 29_19:11_~/Documents/OsLabs/Lab_7/shm]$./shm2r
     shmget: No such file or directory
     [bexh0lder@XuBinHan_五 4月 29_19:12_~/Documents/OsLabs/Lab_7/shm]$./shm2w

     [bexh0lder@XuBinHan_五 4月 29_19:12_~/Documents/OsLabs/Lab_7/shm]$
     ```

   需要先运行shm2w.c开辟共享内存空间并写入数据之后才能运行shm2r.c获取共享内存空间里的字符串，否则shm2r.c先运行就会因为找不到共享内存空间而报错

2. 将①替换成`{*s++ = c;sleep(1);}`，降低写入进程写数据的速度，观察对读出进程的影响？这说明共享内存没有提供同步机制，可能导致数据丢失。

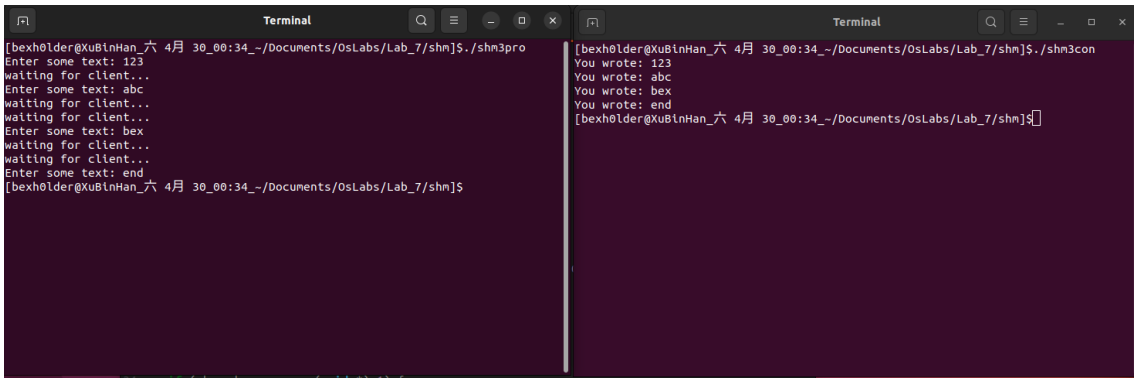


会导致数据的丢失，并且shm2r.c运行完之后共享内存空间已关闭，从而shm2w.c无法在shm2r.c运行完之后继续写入数据

shm3

1. 请确定两个并发进程的启动顺序

   先运行shm3con.c，后运行shm3pro.c



2. 删除蓝色标记的代码，对并发进程有什么影响

   不知道蓝色代码是哪一段，直接审计一下关键部分的代码

   - shm3pro.c

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

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

#define TEXT_SZ 2048

struct shared_use_st {
    int written_by_you;
```

```
    char some_text[TEXT_SZ];
}; //总字节数为 4 + 2048 = 2052 字节

int main()
{
    int running = 1;
    void *shared_memory = (void *)0;
    struct shared_use_st *shared_stuff;
    char buffer[BUFSIZ];
    int shmid;

    shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 |
IPC_CREAT); //新建消息队列，如果key值对应的消息队列已存在则返回此共享内存的标识符,但需要注意如果仅仅只是key值相同而应的消息队列的各参数(如内存大小、内存权限等)不同则会报错

    if (shmid == -1) {
        fprintf(stderr, "shmget failed\n");
        exit(EXIT_FAILURE);
    }

    shared_memory = shmat(shmid, (void *)0, 0);
    if (shared_memory == (void *)-1) {
        fprintf(stderr, "shmat failed\n");
        exit(EXIT_FAILURE);
    }

    shared_stuff = (struct shared_use_st *)shared_memory;
    while(running) { //循环写入内容
        while(shared_stuff->written_by_you == 1) { //相当于互斥锁的效果，等待shm3con将written_by_you重置为0后再将下一个字符串写入共享内存
            sleep(1);
            printf("waiting for client...\n");
        }
        printf("Enter some text: ");
        fgets(buffer, BUFSIZ, stdin);

        strncpy(shared_stuff->some_text, buffer, TEXT_SZ);
        shared_stuff->written_by_you = 1; //将written_by_you置为1使shm3con运行输出共享内存字符串

        if (strncmp(buffer, "end", 3) == 0) {  //如果检查到输入的为end则退出循环
```

```
            running = 0;
        }
    }

    if (shmdt(shared_memory) == -1) { //断开与共享地址的链接
        fprintf(stderr, "shmdt failed\n");
        exit(EXIT_FAILURE);
    }
    exit(EXIT_SUCCESS);
}
```

- shm3con.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>

#define TEXT_SZ 2048

struct shared_use_st {
    int written_by_you;
    char some_text[TEXT_SZ];
}; //总字节数为 4 + 2048 = 2052 字节

int main()
{
    int running = 1;
    void *shared_memory = (void *)0;
    struct shared_use_st *shared_stuff;
    int shmid;

    srand((unsigned int)getpid());
```

```
28
29      shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 |
        IPC_CREAT); //新建消息队列，如果key值对应的消息队列已存在则返回此共享内
        存的标识符,但需要注意如果仅仅只是key值相同而应的消息队列的各参数(如内存
        大小、内存权限等)不同则会报错
30
31      if (shmid == -1) {
32          fprintf(stderr, "shmget failed\n");
33          exit(EXIT_FAILURE);
34      }
35
36  /* We now make the shared memory accessible to the program. */
37
38      shared_memory = shmat(shmid, (void *)0, 0);
39      if (shared_memory == (void *)-1) {
40          fprintf(stderr, "shmat failed\n");
41          exit(EXIT_FAILURE);
42      }
43
44  /* The next portion of the program assigns the shared_memory segment to
        shared_stuff,
45   which then prints out any text in written_by_you. The loop continues until end is
        found
46   in written_by_you. The call to sleep forces the consumer to sit in its critical
        section,
47   which makes the producer wait. */
48
49      shared_stuff = (struct shared_use_st *)shared_memory;
50      shared_stuff->written_by_you = 0; //将written_by_you置为0使shm3pro程序能
        够运行，使shm3con在shm3pro运行之后运行
51      while(running) {
52          if (shared_stuff->written_by_you) {
53              printf("You wrote: %s", shared_stuff->some_text);
54              sleep( rand() % 4 ); /* make the other process wait for us ! */ //等待一会
                儿再将written_by_you置为0
55              shared_stuff->written_by_you = 0; //每次输出完就将written_by_you置为0
                使shm3pro程序能够运行
56              if (strncmp(shared_stuff->some_text, "end", 3) == 0) {
57                  running = 0;
58              }
59          }
60      }
```

```
61
62  /* Lastly, the shared memory is detached and then deleted. */
63
64      if (shmdt(shared_memory) == -1) { //断开与共享地址的链接
65          fprintf(stderr, "shmdt failed\n");
66          exit(EXIT_FAILURE);
67      }
68
69      if (shmctl(shmid, IPC_RMID, 0) == -1) { //删除消息队列
70          fprintf(stderr, "shmctl(IPC_RMID) failed\n");
71          exit(EXIT_FAILURE);
72      }
73
74      exit(EXIT_SUCCESS);
75  }
76
```

3. 理解`shared_stuff->written_by_you`在并发进程中起的作用，效果如何？效率如何？

   shared_stuff->written_by_you在程序中起到了互斥的作用，使shm3con和shm3pro能够交替执行，shm3con首先运行将shared_stuff->written_by_you置为0使shm3pro运行将字符串写入共享内存，而后shm3pro写完后将shared_stuff->written_by_you置为1使shm3con运行输出共享内存的内容然后再将shared_stuff->written_by_you置为0使shm3pro运行，如此往复

编程题

试编写程序，实现父进程和子进程通过共享内存实现信息的交换。要求：子进程先将子进程号写入共享内存，父进程将内容读出并显示。随后，父进程将父进程号写入同一块共享内存，要求子进程读出并显示。使用完毕后，由父进程注销共享内存。

```
1  #include<unistd.h>
2  #include<sys/ipc.h>
3  #include<sys/shm.h>
4  #include<errno.h>
5  #include<sys/wait.h>
6  #include<stdio.h>
7  #include<string.h>
8  #include<stdlib.h>
9
```

```c
#define KEY 1234
#define SIZE 1024

int main()
{
    int shmid;
    pid_t *pid;
    shmid=shmget(KEY,SIZE,IPC_CREAT|0600);
    if(shmid==-1)
    {
        printf("create share memory failed:%s",strerror(errno));
        return 0;
    }
    if(fork( )==0) //child
    {
        pid=(pid_t*)shmat(shmid,NULL,0);
        if(pid==(void*)-1)
        {
            printf("connect to the share memory failed:%s",strerror(errno));
            return 0;
        }
        *pid = getpid();
        sleep(2);//let the father output
        printf("My father's pid is %d\n", *pid);
        shmdt(pid);
        exit(0);

    }else //father
    {
        sleep(1); //let the child run first
        pid=(pid_t*)shmat(shmid,NULL,0);
        if(pid==(void*)-1)
        {
            printf("connect the share memory failed:%s",strerror(errno));
            return 0;
        }
        printf("My child's pid is %d\n", *pid);
        *pid = getpid();
        wait(0);
        shmdt(pid);
        shmctl(shmid,IPC_RMID,NULL);
    }
```

```
52 }
```

```
[bexh0lder@XuBinHan_六 4月 30_01:16_~/Documents/OsLabs/Lab_7]$./MyCode
My child's pid is 4155
My father's pid is 4154
[bexh0lder@XuBinHan_六 4月 30_01:16_~/Documents/OsLabs/Lab_7]$
```