

**本科实验报告**

课 程 名 称： 操作系统原理实验 课 程 编 号： 08060157 学 生 姓 名： 阮炜霖 学 号： 2020101603 学 院： 信息科学技术学院

系： 计算机科学系

专 业： 网络工程

指 导 教 师： 郝振明

教 师 单 位： 计算机科学系 开 课 时 间：2021 ~ 2022 学年度第 二 学期

## 暨南大学教务处

2022年 6 月 15 日

操作系统原理实验 **课程实验项目目录**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 实验项目 编号 | 实验项目名称 | 实验性质 | 成绩 | 指导教师 |
| 1 | 0806015701 | 实验环境的使用 | 验证 |  | 郝振明 |
| 2 | 0806015702 | 进程的控制 | 设计 |  | 郝振明 |
| 3 | 0806015703 | 进程间的通信 | 设计 |  | 郝振明 |
| 4 | 0806015704 | 生产者消费者问题 | 综合 |  | 郝振明 |
| 5 | 0806015705 | 简单文件系统实现 | 综合 |  | 郝振明 |
| 6 | 0806015706 | 非必须实验。若选择做此项目，则项目名称自定。 |  |  | 郝振明 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

\*此表由学生按顺序填写。

# 暨南大学本科实验报告专用纸

课程名称 操作系统原理实验 成绩评定

实验项目名称 生产者消费者问题\_ 指导教师 郝振明

实验项目编号 0806015704 实验项目类型 设计型

学生姓名 阮炜霖 学号 2020101603

学院信息科学技术学院 系 计算机科学系 专业 网络工程

实验时间 2022 年 5 月 12 日 上 午 3、4节 实验地点 N116

**一、 实验目的**

实验目的：

1. 使用三种信号量操作函数对信号量进行创建、发送命令或者操作。
2. 使用多线程和System V信号量解决生产者消费者问题。

实验要求：任选 C 高级程序语言编写源程序，在 Linux 操作系统下调试通过，测试正确。

**二、 实验内容**

1.实验任务

（1）System V信号量的操作函数主要有3个，其函数原型分别为：semget、semctl、semop

，编写一段程序，使用这三个函数对信号量进行创建、发送命令或者操作。实验任务结果测试图在文末给出。（**结果测试图仅供参考，不要求完全一致，鼓励同学们自行设计**）

2）使用多线程和System V信号量解决生产者消费者问题：有一个长度为N的缓冲池（共享内存）被生产者和消费者共同使用。只要缓冲池未满，生产者就可以将消息送入缓冲池；只要缓冲池不空，消费者便可从缓冲池中取走一个消息。生产者向缓冲池放入消息的同时，消费者不能操作缓冲池，反之亦然。

2.实验依据

在实验任务（1）中，Linux系统提供了一组 System V信号量接口来对信号进行操作，相关信号量操作函数由sys/ipc.h文件引用，信号量的声明则在头文件sys/semh中定义。System V信号量的操作函数主要有3个，分别为：

1）semget函数

函数原型：int semget(key\_t key,int nsems,int sem\_flgs);

函数功能：创建或打开一个信号量集合。

返回值 ：成功则返回信号量集合的标识符，错误返回-1，并且设置errno值

参数

key: 一个键值，和消息队列生成键值的方式一样

nsems:

1. 当创建一个信号量集合的时候：该参数表示信号量集合中信号量的个数

2. 当获取一个既有标识符的信号量集合的时候，该值必须小于或者等于集合的大小，否则返回EINVAL错误

semflg: 一个位掩码，相关参数如下，可以用or或起来：IPC\_CREATE/IPC\_EXCL

2）segctl函数

函数原型：int semctl(int semid,int semnum,int cmd,union semun\_arg);

函数功能：在信号量集合或者当中的单个信号量执行相关控制操作

返回值：如果错误会返回-1并且设置errno值，否则返回一个系统调用标识符。

参数

semid:信号量集合的标识符

semnum:对某一个信号量集合中的单个信号量操作时，指明了相关信号量的序号，置0表示忽略该参数。

cmd:指定相关控制操作

semun\_arg: 传入类似如下结构体作为信号量集合

union semun{

short val; /\*SETVAL用的值\*/

struct semid\_ds\* buf; /\*IPC\_STAT、IPC\_SET用的semid\_ds结构\*/

unsigned short\* array; /\*SETALL、GETALL用的数组值\*/

struct seminfo \*buf; /\*为控制IPC\_INFO提供的缓存\*/

}arg;

3)semop函数

函数原型： int semop(int semid,struct sembuf \*sops,size\_t nsops);

函数功能：系统调用在semid标识符的信号量上进行一个或多个原子操作

返回值：返回0表示成功，1表示错误

参数

semid:信号量的标识符

sops:指向进行操作的信号量结构体数组的首地址

nsops:指出sops指向的数组包含的元素个数

在 Linux系统中，使用 System V信号量通常分为以下4个步骤。

(1)调用 semget()函数创建信号量，或获得在系统中已存在的信号量。不同进程通过使用同

一个信号量键值来获得同一个信号量。

(2)使用 sectl()函数的 SETVAL操作初始化信号量。

(3)调用 semop()函数进行信号量的PV操作，这是实现进程间同步和互斥的核心工作。

(4)如果不需要信号量，则从系统中删除它，此时使用 semctl()函数的IPC\_RMID操作。

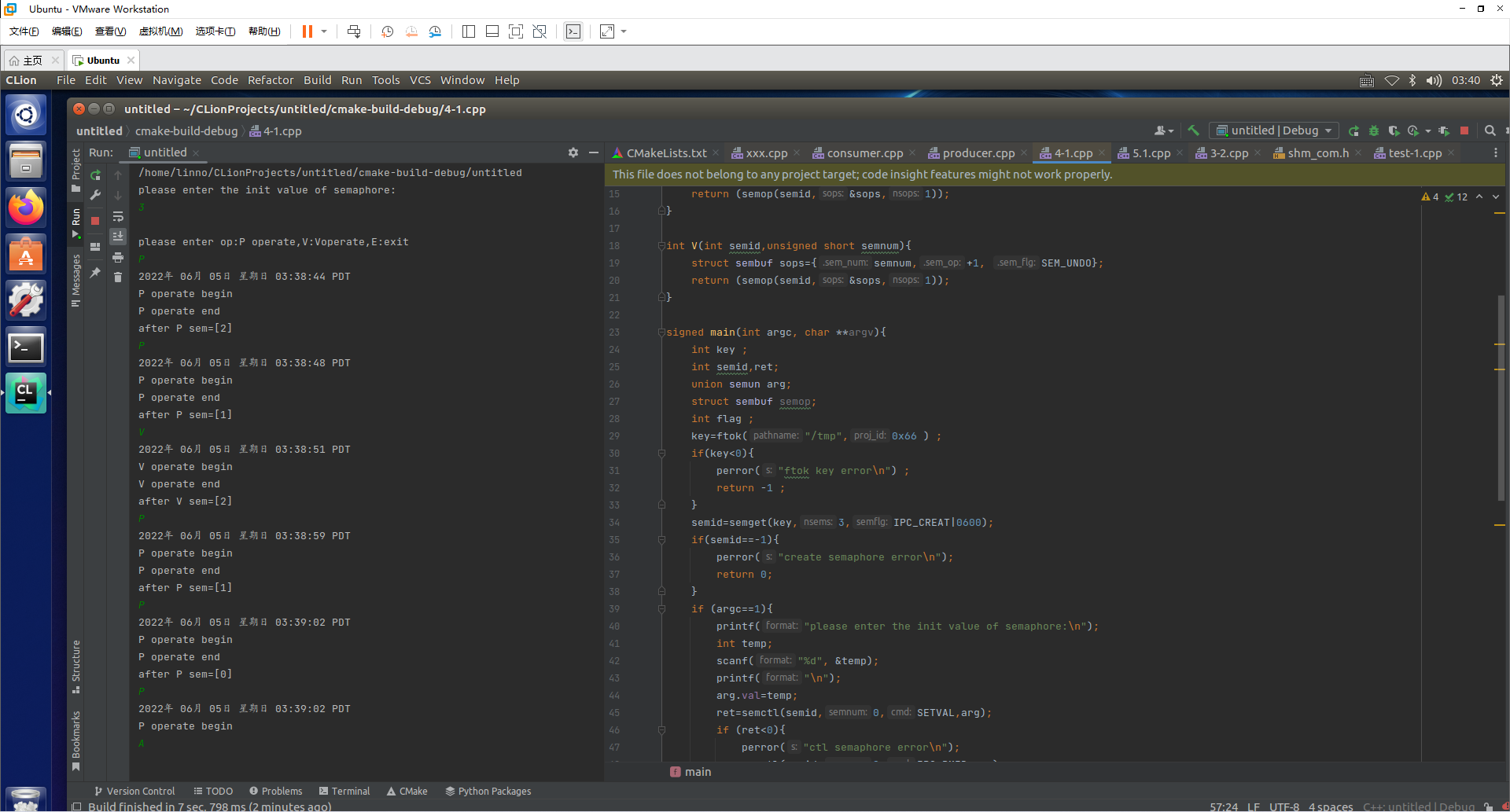
通过使用结构体维护信息集，其成员包括所有者ID，所有者组ID，创建者及其组ID以及访问模式等等。

在实验任务（2）中，用两个进程分别代表生产者和消费者，以及一个使用一个库函数作为新定义的缓冲池。生产者和消费者共同使用同一个长度为N的缓冲池（共享内存），在缓冲池没满的情况下，生产者可以将消息放入；在缓冲池不为空的情况下，消费者可以将消息取出。可通过任务一的方法，通过信号量对生产者和消费者存放的变量进行操作。开两个中断分别运行生产者和消费者并测试是否两进程可以完成通信。

## 三、 实验步骤

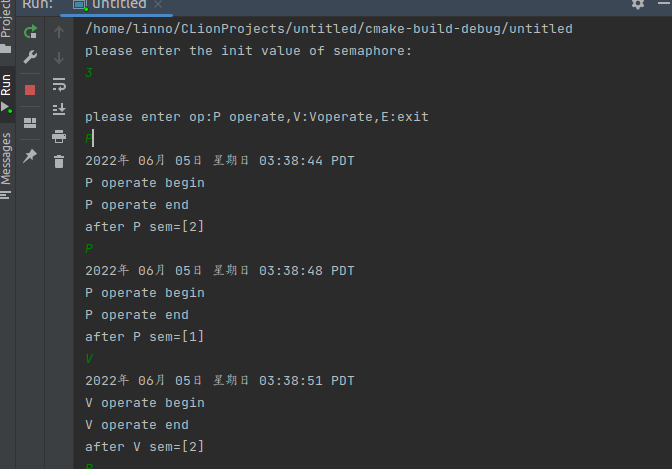
实验一：

运行程序后，先输入初始的信号量。



当在程序中输入P时，生产者会生产一个资源并返回剩余空位。

当在程序中输入V时，生产者会生产一个资源并返回剩余空位。



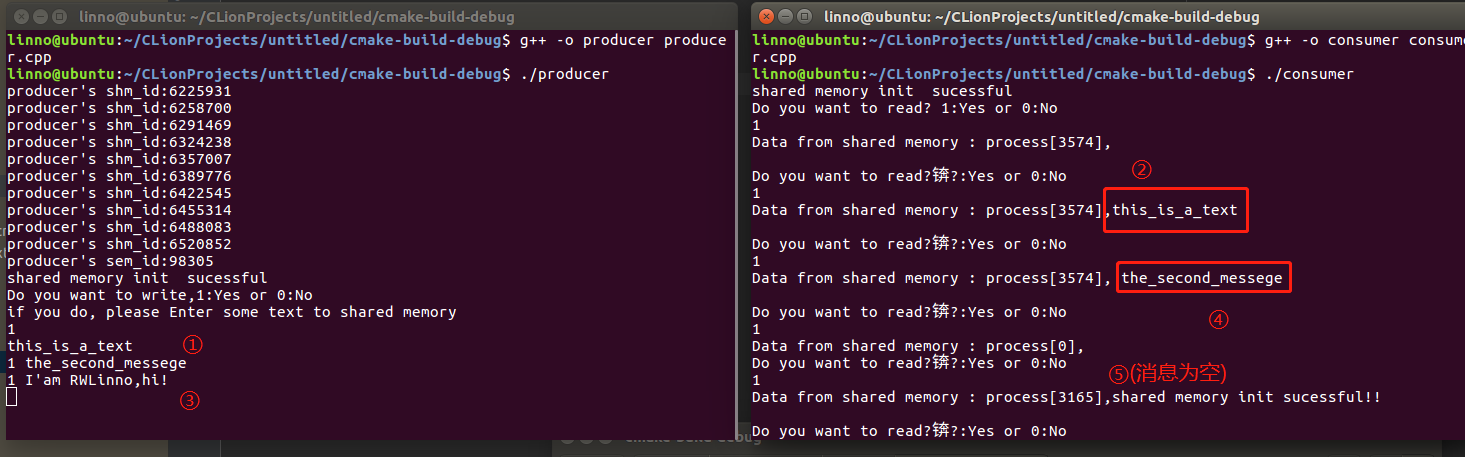
如果在资源值剩余0时调用P或者可用资源剩余值为0时调用V，那么将返回错误信息。



实验二：

由于CLion不支持同时编译运行两份代码，因此在这里使用终端编译两个程序。

1. 调用生产者进程，创建一个共享内存区并写入信息。
2. 调用消费者进程，此时可以将生产者放入的信息下一步读出。
3. 下图为用两个终端运行生产者消费者进程，在生成资料的下一步读出(步骤①②③④为两次写入写出共享内存区)，如果生产的资料被用完，则输出空。



## 五、附录

程序1 4-1.cpp

1. #include <bits/stdc++.h>
2. #include <sys/types.h>
3. #include <sys/ipc.h>
4. #include <sys/sem.h>
6. **union** semun{ //union of sem
7. **int** val;
8. **struct** semid\_ds \*buf; /\* buffer for IPC\_STAT, IPC\_SET \*/
9. unsigned **short** \*array; /\* array for GETALL, SETALL \*/
10. **struct** seminfo \*\_\_buf; /\* buffer for IPC\_INFO \*/
11. };
13. **int** P(**int** semid,unsigned **short** semnum){
14. **struct** sembuf sops={semnum,-1, SEM\_UNDO};
15. **return** (semop(semid,&sops,1));
16. }
18. **int** V(**int** semid,unsigned **short** semnum){
19. **struct** sembuf sops={semnum,+1, SEM\_UNDO};
20. **return** (semop(semid,&sops,1));
21. }
23. **signed** main(**int** argc, **char** \*\*argv){
24. **int** key ;
25. **int** semid,ret;
26. **union** semun arg;
27. **struct** sembuf semop;
28. **int** flag ;
29. key=ftok("/tmp",0x66 ) ;
30. **if**(key<0){
31. perror("ftok key error\n") ;
32. **return** -1 ;
33. }
34. semid=semget(key,3,IPC\_CREAT|0600);
35. **if**(semid==-1){
36. perror("create semaphore error\n");
37. **return** 0;
38. }
39. **if** (argc==1){
40. printf("please enter the init value of semaphore:\n");
41. **int** temp;
42. scanf("%d", &temp);
43. printf("\n");
44. arg.val=temp;
45. ret=semctl(semid,0,SETVAL,arg);
46. **if** (ret<0){
47. perror("ctl semaphore error\n");
48. semctl(semid,0,IPC\_RMID,arg);
49. **return** -1 ;
50. }
51. }
52. ret =semctl(semid,0,GETVAL,arg);
53. //printf("after semctl, sem[0].val =[%d]\n",ret);
54. **char** c='0';
55. printf("please enter op:P operate,V:Voperate,E:exit\n");
56. **while**(c!='E'){
57. scanf("%c",&c);
58. **if**(c=='P'){
59. system("date");
60. printf("P operate begin\n") ;
61. flag=P(semid,0);
62. ret=semctl(semid,0,GETVAL,arg);
63. **if**(flag){
64. perror("P operate error\n");
65. **return** -1;
66. }
67. printf("P operate end\n") ;
68. printf("after P sem=[%d]\n",ret);
69. }
70. **if**(c=='V'){
71. system("date");
72. printf("V operate begin\n");
73. **if**(V(semid, 0) < 0){
74. perror("V operate error\n");
75. **return** -1;
76. }
77. printf("V operate end\n") ;
78. ret=semctl(semid,0,GETVAL,arg);
79. printf("after V sem=[%d]\n",ret);
80. **if**(argc>1) semctl(semid,0,IPC\_RMID,arg);
81. }
82. }
83. **return** 0 ;
84. }

程序4-2 sem\_com.h

1. //sem\_com:
2. #ifndef \_SEM\_COM\_H
3. #define \_SEM\_COM\_H
4. #include<bits/stdc++.h>
5. #include <unistd.h>
6. #include <sys/types.h>
7. #include <sys/ipc.h>
8. #include <sys/sem.h>
10. **union** semun{
11. **int** val;
12. **struct** semid\_ds \*buf;
13. unsigned **short** \*array;
14. **struct** seminfo \* \_buf;
15. };
17. **int** sem\_init(**int** sem\_id,**int** init\_value){
18. **union** semun sem\_union;
19. sem\_union.val = init\_value;
20. **if** (semctl(sem\_id,0,SETVAL,sem\_union)==-1) **return** -1;
21. **else** **return** 0;
22. }
24. **int** sem\_delete(**int** sem\_id){
25. **if**(semctl(sem\_id,0,IPC\_RMID,NULL) < 0){
26. perror("semctl\_delete");
27. **return**(-1);
28. } **else** **return** 0;
30. }
32. **int** sem\_p(**int** sem\_id)
33. {
34. **struct** sembuf buf;
35. buf.sem\_num = 0;
36. buf.sem\_op = -1;
37. buf.sem\_flg = SEM\_UNDO;
38. **if**(semop(sem\_id,&buf,1)<0){
39. perror("semop1");
40. **return** -1;
41. }
42. **return** 0;
43. }
45. **int** sem\_v(**int** sem\_id){
46. **struct** sembuf buf;
47. buf.sem\_num = 0;
48. buf.sem\_op = 1;
49. buf.sem\_flg = SEM\_UNDO;
50. **if**(semop(sem\_id,&buf,1) < 0 ){
51. perror("semop2");
52. **return** -1;
53. }
54. **return** 0;
55. }
57. #endif

程序4-3 shm\_com.h

1. //  shm\_com:
2. #ifndef  \_SHM\_COM\_H
3. #define \_SHM\_COM\_H
4. #include <sys/types.h>
5. #include <sys/ipc.h>
6. #include <sys/shm.h>
8. #define SHM\_BUFSIZE 1024
9. **typedef** **struct**
10. {
11. pid\_t pid;
12. **char** buffer[SHM\_BUFSIZE];
13. }shm\_buf;
14. #endif

程序4-4 producer.cpp

1. //producer:
2. #include <unistd.h>
3. #include "sem\_com.h"
4. #include "shm\_com.h"
5. #define IPC\_KEY\_PATH    "/"
6. #define SHM\_KEY\_PROJ    1
7. #define SEM\_KEY\_PROJ    2
8. **int** main(){
9. **int** shm\_id[15],sem\_id;
10. shm\_buf \*shmbuf[15];
11. **for**(**int** i=0;i<10;i++){
12. **if** ((shm\_id[i] = shmget((ftok(IPC\_KEY\_PATH, i)), **sizeof**(shm\_buf), IPC\_CREAT | 0666)) < 0) {
13. perror("shmget");
14. exit(-1);
15. }
16. }
17. **for**(**int** i=0;i<10;i++) printf("producer\'s shm\_id:%d\n",shm\_id[i]);
18. **if**((sem\_id=semget(ftok(IPC\_KEY\_PATH,1),1,IPC\_CREAT|0666))<0){
19. perror("semget");
20. exit(-1);
21. }
22. printf("producer\'s sem\_id:%d\n",sem\_id);
23. sem\_init(sem\_id,1);
24. **for**(**int** i=0;i<10;i++)
25. **if**((shmbuf[i]=(shm\_buf \*)shmat(shm\_id[i],NULL,0))<0){
26. perror("shmat");
27. exit(-1);
28. }
29. printf("shared memory init  sucessful\n");
30. **int** cnt=0;
31. printf("Do you want to write,1:Yes or 0:No\nif you do, please Enter some text to shared memory\n");
32. **while**(1){
33. cnt=(cnt+1)%10;
34. **int** op;
35. scanf("%d",&op);
36. **if**(op==1){
37. sem\_p(sem\_id);
38. **if** (fgets(shmbuf[cnt]->buffer,SHM\_BUFSIZE,stdin)==NULL)
39. {
40. perror("fgets");
41. sem\_v(sem\_id);
42. **break**;
43. }
44. shmbuf[cnt]->pid = getpid();
45. sem\_v(sem\_id);
46. }
47. **if**(op==0) **break**;
48. }
49. sem\_delete(sem\_id);
50. **for**(**int** i=0;i<10;i++){
51. **if** (shmdt(shmbuf[i])<0){
52. perror("shmdt");
53. exit(-1);
54. }
55. }
56. **return** 0;
57. }

程序4-5 Consumer.cpp

1. //Consumer:
2. #include <stdlib.h>
3. #include <string.h>
4. #include <stdio.h>
5. #include <unistd.h>
6. #include <time.h>
7. #include "sem\_com.h"
8. #include "shm\_com.h"
9. #define IPC\_KEY\_PATH    "/"
10. #define SHM\_KEY\_PROJ    1
11. #define SEM\_KEY\_PROJ    2
12. **int** main(){
13. **int** shm\_id[15],sem\_id;
14. shm\_buf \*shmbuf[15];
15. **for**(**int** i=0;i<10;i++) {
16. **if** ((shm\_id[i] = shmget((ftok(IPC\_KEY\_PATH, i)), **sizeof**(shm\_buf), IPC\_CREAT | 0666))<0){
17. perror("shmget");
18. exit(-1);
19. }
20. }
21. **if**((sem\_id=semget(ftok(IPC\_KEY\_PATH,1),1,IPC\_CREAT|0666))<0){
22. perror("semget");
23. exit(-1);
24. }
25. **for**(**int** i=0;i<10;i++){
26. **if** ((shmbuf[i] = (shm\_buf \*) shmat(shm\_id[i], NULL, 0)) < 0) {
27. perror("shmat");
28. exit(-1);
29. }
30. }
31. printf("shared memory init  sucessful\n");
32. **int** cnt=0;
33. printf("Do you want to read? 1:Yes or 0:No\n");
34. **int** op;
35. scanf("%d",&op);
36. **while**(1){
37. cnt=(cnt+1)%10;
38. sem\_p(sem\_id);
39. printf("Data from shared memory : process[%d],%s \n",shmbuf[cnt]->pid,shmbuf[cnt]->buffer);
40. **if**(strncmp(shmbuf[cnt]->buffer,"quit",4)==0) **break**;
41. shmbuf[cnt]->pid = 0 ;
42. memset(shmbuf[cnt]->buffer,0,SHM\_BUFSIZE);
43. sem\_v(sem\_id);
44. printf("Do you want to read?锛?:Yes or 0:No\n");
45. scanf("%d",&op);
46. }
47. **for**(**int** i=0;i<10;i++){
48. **if**(shmdt(shmbuf[i])<0) {
49. perror("shmdt");
50. exit(-1);
51. }
52. }
53. **for**(**int** i=0;i<10;i++){
54. **if**(shmctl(shm\_id[i], IPC\_RMID, NULL)<0){
55. perror("shmctl\_remove");
56. exit(-1);
57. }
58. }
59. **return** 0;
60. }