

实验四 Linux 进程的管道通信

1. 实验目的

了解管道通信的特点，熟悉管道通信的概念和原理，掌握管道通信的实现方法。

2. 相关知识与系统调用

(1) 管道的概念

管道是用于连接一个读进程和一个写进程的 `pipe` 文件。逻辑上是管道文件；物理上是由文件系统的高速缓冲区构成。管道按 FIFO 方式单向传送消息，只允许在建立者及其子进程间使用。

写进程：以字符流的形式将大量数据送入管道，不断从 `pipe` 一端写入数据流，在规定的最大长度（如 4096 字节）范围内，每次写入的信息长度是可变的。

读进程：在需要时从 `pipe` 的另一端读出数据，读出单位长度也是可变的。

(2) pipe 的建立

`int fd[2] pipe(fd)` 在系统打开文件表中建立该 `pipe` 的两个表目

`fd[1]`: `pipe` 入口，控制写操作

`fd[0]`: `pipe` 的出口，控制读操作

(3) pipe 的读写

写操作：`write(fd[1],buf,size)`

把 `buf` 中的长度为 `size` 字符的消息送入管道入口 `fd[1]`

`buf`: 存放消息的空间，`size`: 要写入的字符长度

读操作：`read(fd[0],buf,size)`，从 `pipe` 出口 `fd[0]` 读出 `size` 字符的消息置入 `buf`

互斥：一个进程正在对 `pipe` 进行读/写时，另一进程必须等待。

同步：

发送进程：写 `pipe` 时，若 `pipe` 文件长度已经到 4096 字节，仍有部分信息没有写入，则进程进入睡眠状态；当读进程收走了全部信息时，被唤醒，将余下信息送入 `pipe` 中。

接收进程：读 `pipe` 时，若 `pipe` 为空，则进入等待状态。一旦有发送进程执行写操作，被唤醒。

(4) 其他相关系统调用

`sleep(n)`: `n` 为整数，为秒(s)级。进程调用后将自动阻塞 `n` 秒，`n` 秒后自动唤醒，进入就绪状态。

用来锁定文件的某些段或者整个文件，本函数使用的头文件为：“`unistd.h`”。

`int lockf(int files, int function, long size)` 其中：`files` 是文件描述符，`function` 是锁定和解锁；1 表示锁定，0 表示解锁。`size` 是锁定和解锁的字节数，若用 0，表示从文件的当前位置到文件尾。

3. 实验程序

【程序 3_11】用 C 语言编写一个程序，建立一个 `pipe`。父进程生成一个子进程，子进程向 `pipe` 中写入一字符串，父进程从 `pipe` 中读出该字符串。如图 3.1 所示。

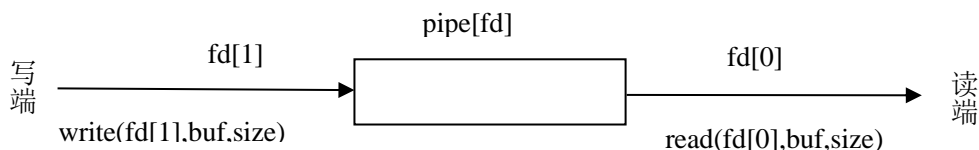


图 3.1 管道通信

```

#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int x ,fd[2];
    char buf [30] ,s[30];
    pipe (fd);    //创建管道
    while((x=fork ( )) == -1);    //创建子程序失败时，循环
    if(x== 0)
    {
        sprintf (buf,"This is an example \n");
        write (fd[1],buf,30);    //将 buf 中字符写入管道
    }
    else    //父进程返回
    {
        wait (0);
        read(fd[0],s,30);    //父进程读管道中字符
        printf("%s",s);
    }
}

```

【程序运行结果截图】

```

administrator@ubuntu:~$ ./li3_11
This is an example
administrator@ubuntu:~$

```

【程序 3_12】编写一程序，建立一个管道。同时，父进程生成子进程 p1，p2，这两个子进程分别向管道中写入各自的字符串，父进程读出它们。如图 3.2 所示。

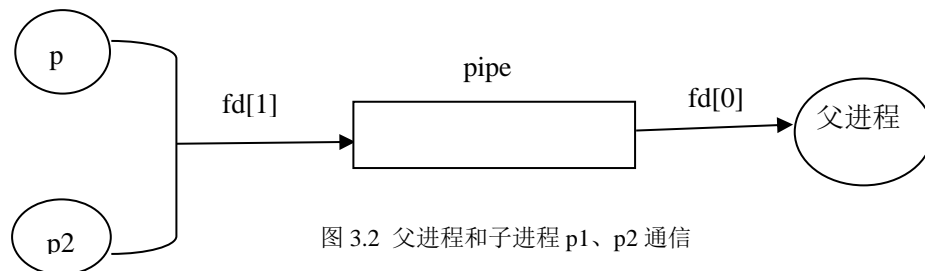


图 3.2 父进程和子进程 p1、p2 通信

```

#include<stdio.h>
#include <stdlib.h>
int main( )
{
    int i , r , p1,p2,fd[2];
    char buf[50] , s[50];
    pipe(fd);    //父进程建立管道
    while((p1=fork()) == -1);    //建子进程 p1，失败时循环
    if(p1==0)    //由子进程 p1 返回，执行子进程 p1
    {

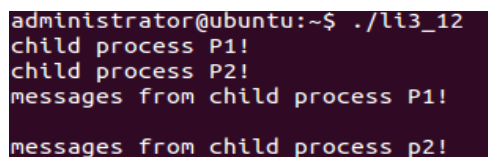
```

```

    lockf(fd[1],1,0);    //加锁锁定写入端
    sprintf(buf, "messages from child process p1! \n");
    printf("child process p1!\n");
    write(fd[1],buf,50);    //将 buf 中的 50 个字符写入管道
    sleep(5);    //睡眠 5 秒，让父进程读
    lockf(fd[1] , 0,0);    //释放管道写入端
    exit(0);    //关闭 p1
}
else    //从父进程返回，执行父进程
{
    while((p2=fork())== -1);    //建子进程 p2，失败时循环
    if(p2== 0)    //从子进程 p2 返回，执行 p2
    {
        lockf(fd[1],1,0);    //锁定写入端
        sprintf(buf, "messages from child process p2!\n");
        printf("child process p2! \n");
        write(fd[1],buf,50);    //将 buf 中字符写入管道
        sleep(5);    //睡眠等待
        lockf(fd[1] , 0,0);    //释放管道写入端
        exit(0);    //关闭 p2
    }
    wait(0);
    if(r=read(fd[0],s,50)== -1)
        printf("can't read pipe\n");
    else printf("%s\n",s);
    wait(0);
    if(r=read(fd[0],s,50)== -1)
        printf("can't read pipe\n");
    else printf("%s\n",s);
    exit(0);
}
}

```

【程序运行结果截图】



```

administrator@ubuntu:~$ ./li3_12
child process P1!
child process P2!
messages from child process P1!
messages from child process p2!

```

4. 实验思考与扩充

- (1) 将【程序 3_11】中的父进程作为 pipe 的写端，而子进程作为 pipe 的读端，是否可以？
- (2) 【程序 3_12】中若子进程有 3 个，情况会怎样？lockf 这个系统调用的作用是什么？

实验五 Linux 进程间通信—软中断信号处理

1. 实验目的

基本掌握 Linux 进程软中断通信的基本原理和实现方法。

2. 相关知识与系统调用

软中断信号的处理，主要是实现同一用户的各进程之间的通信。

(1) kill(pid, sig): 发送信号

(2) signal(sig, func): 指定进程对信号 sig 的处理行为是调用函数 func。

3. 实验程序

【程序 3_13】使用系统调用fork()创建两个子进程，让父进程分别与两个子进程进行软中断通信。

```
#include<unistd.h>
#include<stdio.h>
#include<signal.h>
#include<stdlib.h>
int wait_mark;
void waiting( )    //自定义 waiting 函数与 wait(0)作用不同。其作用是通过循环使子进程停止。
{
    while(wait_mark!=0);
}
void stop( )    //调用此函数时，使 wait_mark=0，结束 waiting 函数中的循环，使子进程继续执行。
{
    wait_mark=0;
}
int main( )
{
    int p1, p2;
    while((p1=fork())!=-1);
    if(p1>0)
    {
        while((p2=fork())!=-1);
        if(p2>0)
        {    //在父进程中
            printf("parent\n");
            kill(p1,16);    //发送信号 16
            kill(p2,17);    //发送信号 17
            wait(0);    //等待 p1、p2 中的某个子进程结束
            wait(0);    //等待 p1、p2 中的某个子进程结束
            printf("parent process id killed!\n");
            exit(0);
        }
    }
}
```

```

    }
    else
    {
        printf("p2\n");
        wait_mark=1;
        signal(17,stop);    //将信号 17 和函数 stop 关联起来
        waiting();    //循环，直到父进程发送信号 17 时，调用 stop 结束循环继续执行

        printf("child process 2 is killed by parent!\n");
        exit(0);
    }
}
else
{ //子进程 p1
    printf("p1\n");
    wait_mark=1;
    signal(16,stop);    //将信号 16 和 stop 关联起来。
    waiting();    //循环，直到父进程发送信号 17 时，调用 stop 后结束循环继续执行
    printf("child process 1 is killed by parent!\n");
    exit(0);
}
}

```

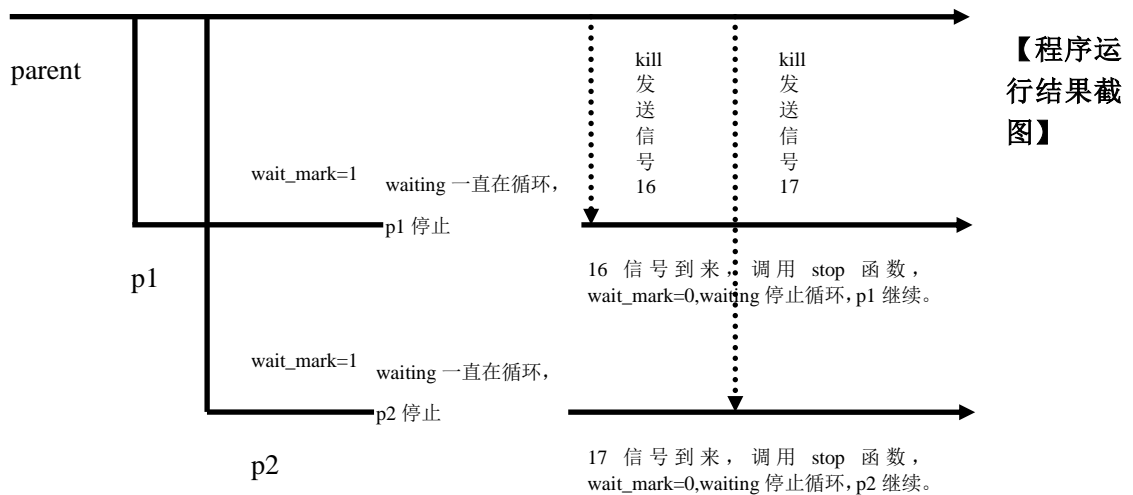


图 3.3 程序执行过程图

```

[shi01@Yangfan201 ~]$ ./li3_13
p1
p2
parent
child process 1 is killed by parent!
child process 2 is killed by parent!
parent process id killed!

```

```
[shi01@Yangfan201 ~]$ ./li3_13
p2
p1
parent
child process 1 is killed by parent!
child process 2 is killed by parent!
parent process id killed!

[shi01@Yangfan201 ~]$ ./li3_13
p1
p2
parent
child process 2 is killed by parent!
child process 1 is killed by parent!
parent process id killed!
```

4. 实验思考与扩充

- (1) 程序运行结果为何不唯一？除了上述 3 种情况外，是否还有其他情况？
- (2) 程序中 `signal()` 的作用是什么？
- (3) 如何修改【程序 3_13】，当用户按 `del` 键时，父进程能够捕捉到它对应的中断信号，然后再向两个子进程发信号？
- (4) 当【程序 3_13】的运行结果是下面这种情况时，说明了什么？应该怎样解决？

```
[shi01@Yangfan201 ~]$ ./li3_13
p1
parent
child process 1 is killed by parent!
p2
```

实验六 Linux 进程间通信—消息队列

1. 实验目的

了解消息队列的通信机制及原理，掌握消息通信相关系统调用的使用方法。

2. 相关知识与系统调用

消息的创建、发送和接收。多个进程通过访问一个公共的消息队列来交换信息。

消息队列：即消息的一个链表。

任何进程都可以向消息队列中发送消息(消息类型及正文)，其他进程都可以从消息队列中根据类型获取相应的消息。

(1) 头文件：“sys/msg.h”

(2) 打开或创建消息队列：int msgget(key_t key, int msgflg);

key：消息队列的键

IPC_PRIVATE：创建一个私有的消息队列

其他：可被多个进程使用的消息队列

msgflg：设置操作类型及访问权限 IPC_CREAT / IPC_EXCL

(3) 获得或设置消息队列属性：int msgctl(int msgid, int cmd, struct msqid_ds *data);

(4) 发送消息：int msgsnd(int msgid, const void *msgp, size_t msgsize, int flags);

参数：

msgid：消息队列标识符 id

msgp：指针，用户自定义缓冲区，可定义成结构体类型，包含两项 long mtype，代表消息类型，char mtext[MTEXTSIZE]；消息正文

msgsize：要发送消息正文的长度

mflags：标志，若设置 IPC_NOWAIT 则不等待消息发出就返回

返回值：成功：返回 0

错误：返回-1 (置 errno)

(5) 接收消息：int msgrcv(int msgid, void *msgp, size_t mtexsize, long msgtype, int flags);

参数：与 msgsnd 类似

msgtype：>0：只接收指定类型消息的第一个

=0：不管什么消息类型都读取队列中第一个数据

<0：接收等于或小于其绝对值的最低类型的第一个，如有 5、6、17 三类，若为-6，则获取类型 5 的。

返回值：成功：返回消息正文字节数

错误：返回-1 (置 errno)

3. 实验程序

【程序 3_14】一个进程内部的消息通信，用来检验 msgtype 的作用。

```
#include <sys/types.h>
```

```
#include <sys/msg.h>
```

```
#include <unistd.h>
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
int main( )
```

```
{
```

```

int msgid; int status;
char str1[ ]={"test message:hello!"};
char str2[ ]={"test message:goodbye!"};
char str3[ ]={"The third message!"};
struct msgbuf {
    long msgtype;
    char msgtext[1024];
}sndmsg, rcvmsg;
if((msgid=msgget(IPC_PRIVATE,0666)) ==-1)
{
    printf("msgget error!\n");
    exit(254);
}
sndmsg.msgtype=111;
sprintf(sndmsg.msgtext,str1);
if( msgsnd(msgid,&sndmsg,sizeof(str1)+1,0)==-1)
{
    printf("msgsnd error!\n");
    exit(254);
}
sndmsg.msgtype=222;
sprintf(sndmsg.msgtext,str2);
if( msgsnd(msgid,&sndmsg,sizeof(str2)+1,0)==-1)
{
    printf("msgsnd error!\n");
    exit(254);
}
sndmsg.msgtype=333;
sprintf(sndmsg.msgtext,str3);
if( msgsnd(msgid,&sndmsg,sizeof(str3)+1,0)==-1)
{
    printf("msgsnd error!\n");
    exit(254);
}
if(status= msgrcv(msgid,&rcvmsg,80,222,IPC_NOWAIT ) ==-1)
{
    printf("msg rcv error!\n");
    exit(254);
}
printf("The receieved message: %s.\n",rcvmsg.msgtext);
msgctl(msgid,IPC_RMID,0);
exit(0);
}

```

【程序运行结果截图】


```
administrator@ubuntu:~$ ./li3_14
The received message: test message:goodbye!.
```

程序思考：将上述程序中的 `msgrcv(msgid,&rcvmsg,80,222,IPC_NOWAIT)` 这句里的 222 分别换成 333 和 0，来分别再做做，观察运行结果有什么变化？

【程序 3_15】不同进程间的消息通信。通信的进程间通过 `MSGKEY` 值来确定消息队列。

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/msg.h>
#include <sys/ipc.h>
#include <stdlib.h>
#define MSGKEY 75
struct msgform{
    long mtype;
    char msgtext[1030];
}msg;
int msgqid,i;
int main( )
{
    while((i=fork())== -1);
    if(!i) SERVER( );
    while((i=fork())== -1);
    if(!i) CLIENT( );
    wait(0);
    wait(0);
}
void SERVER( )
{
    msgqid= msgget(MSGKEY,0777|IPC_CREAT);
    do
    {
        msgrcv(msgqid,&msg,1030,0,0);
        printf("(server)received message %d \n", msg.mtype );
    }while(msg.mtype!=1);
    msgctl(msgqid,IPC_RMID,0);
    exit(0);
}
void CLIENT( )
{
    int i;
    msgqid=msgget(MSGKEY,0777);
    for(i=10;i>=1;i--)
    {
        msg.mtype=i;
```

```

        printf("(client)sent\n");
        msgsnd(msgqid,&msg,1030,0);
    }
    exit(0);
}

```

【程序运行结果截图】

```

administrator@ubuntu:~$ ./li3_15
(client)sent
(client)sent
(client)sent
(client)sent
(client)sent
(client)sent
(client)sent
(client)sent
(client)sent
(client)sent
(server)received message 10
(server)received message 9
(server)received message 8
(server)received message 7
(server)received message 6
(server)received message 5
(server)received message 4
(server)received message 3
(server)received message 2
(server)received message 1

```

```

administrator@ubuntu:~$ ./li3_15
(client)sent
(client)sent
(client)sent
(client)sent
(client)sent
(server)received message 10
(server)received message 9
(client)sent
(client)sent
(server)received message 8
(client)sent
(client)sent
(server)received message 7
(client)sent
(server)received message 6
(server)received message 5
(client)sent
(server)received message 4
(server)received message 3
(server)received message 2
(server)received message 1

```

【程序 3_16】在【程序 3_15】的基础上，发送和接收有内容的消息。

```

#include <stdio.h>
#include <sys/types.h>
#include <sys/msg.h>
#include <sys/ipc.h>
#include <stdlib.h>
#define MSGKEY 75
struct msgform{
    long mtype;
    char msgtext[1030];    //存放消息的内容
}msg;
int msgqid, i;
void CLIENT( )
{
    int i; char string_i[5];    //存放信息
    msgqid=msgget(MSGKEY,0777);    //打开一个消息队列；0777 是文件的存取权限
    for(i=10;i>=1;i--)
    {
        msg.mtype=i;
        printf("(client)sent\n");
        sprintf(msg.msgtext,"the content of message is:  ");
        sprintf(string_i, "message  %d",i);
        strcat(msg.msgtext,string_i);
        strcat(msg.msgtext,"\n");
        msgsnd(msgqid,&msg,1030,0);
    }
}

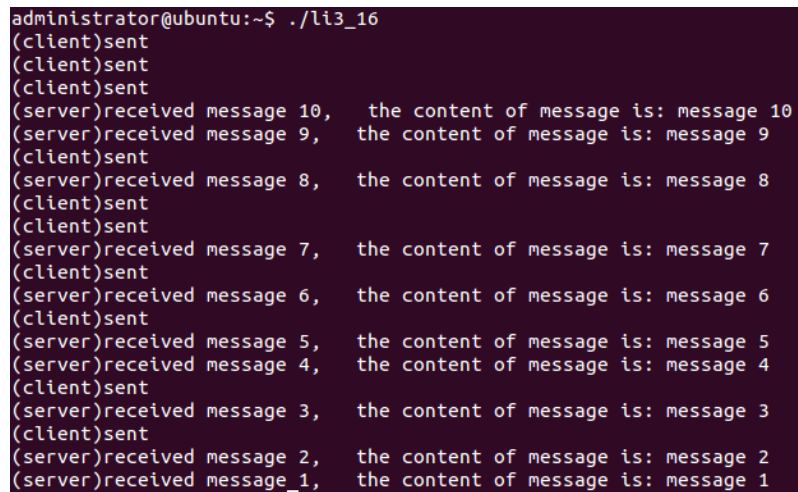
```

```

        exit(0);
    }
void SERVER( )
{
    msgqid=msgget(MSGKEY,0777|IPC_CREAT);    //创建一个消息队列等
    do
    {
        msgrcv(msgqid,&msg,1030,0,0);
        printf("(server)received message %d \n", msg.mtype );
        printf("%s", msg.msgtext );
    }while(msg.mtype!=1);
    msgctl(msgqid,IPC_RMID,0);
    exit(0);
}
int main( )
{
    while((i=fork())==1);
    if(!i) SERVER();
    while((i=fork())==1);
    if(!i) CLIENT();
    wait(0);
    wait(0);
}

```

【程序运行结果截图】



```

administrator@ubuntu:~$ ./li3_16
(client)sent
(client)sent
(client)sent
(server)received message 10, the content of message is: message 10
(server)received message 9, the content of message is: message 9
(client)sent
(server)received message 8, the content of message is: message 8
(client)sent
(client)sent
(server)received message 7, the content of message is: message 7
(client)sent
(server)received message 6, the content of message is: message 6
(client)sent
(server)received message 5, the content of message is: message 5
(server)received message 4, the content of message is: message 4
(client)sent
(server)received message 3, the content of message is: message 3
(client)sent
(server)received message 2, the content of message is: message 2
(server)received message 1, the content of message is: message 1

```

```
administrator@ubuntu:~$ ./li3_16
(client)sent
(client)sent
(client)sent
(client)sent
(client)sent
(client)sent
(client)sent
(client)sent
(client)sent
(client)sent
(client)sent
(server)received message 10, the content of message is: message 10
(server)received message 9, the content of message is: message 9
(server)received message 8, the content of message is: message 8
(server)received message 7, the content of message is: message 7
(server)received message 6, the content of message is: message 6
(server)received message 5, the content of message is: message 5
(server)received message 4, the content of message is: message 4
(server)received message 3, the content of message is: message 3
(server)received message 2, the content of message is: message 2
(server)received message 1, the content of message is: message 1
```

4. 实验思考与扩充

- (1) 观察上面两次运行的结果有什么不同，为什么？
- (2) 将【程序 3_16】和软中断程序结合，实现客户端每发送一条消息，服务器端接受一条消息。

实验七 Linux 进程间通信—共享内存

1. 实验目的

熟悉和掌握共享存储区机制及其实现方法。

2. 相关知识与系统调用

共享内存是指同一系统中的几个进程可共享某块物理内存。

(1) 头文件：“sys/shm.h”

(2) 打开或创建共享区：int shmget(key_t key, size_t size, int shmflg);

参数：key：键值

IPC_PRIVATE：创建一个私有的 shm

其他：非 IPC_PRIVATE 整数值。

size：指明 shm 的大小，若 shm 已经存在，则 size 应为 0

shmflg：设置访问权限及 IPC_CREAT / IPC_EXCL

返回值：成功：该 shm 的 id，当前进程是其拥有者及创建者

错误：-1

(3) 将共享内存连接到进程中：void *shmat(int shmid, const void *shmaddr, int flags);

参数：shmid：共享内存标识符 id

shmaddr：进程映射内存段的地址，可指定，但一般设为 NULL，表示由系统安排。

flags：对该内存的段设置是否只读 (SHM_RDONLY)，默认是读写。

返回值：成功：进程中该内存段的地址

错误：-1

3. 实验程序

【程序 3_17】客户端发送数据，服务器端接收数据。完成共享存储区的通信。

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/msg.h>
#include <sys/ipc.h>
#include <stdlib.h>
#define SHMKEY 75
int shmid,i;
int *addr;
void CLIENT() //客户端
{
    int i;
    shmid=shmget(SHMKEY, 1024, 0777); //打开共享区等
    addr=shmat(shmid, 0, 0); //指向第一个单元
    for(i=5;i>=0;i--)
    {
        while(*addr!=-1); //服务器端没有取走，反复等待
        printf("(client)sent, ");
        *addr=i; //实现同步
        printf("client i: %d\n", i);
    }
}
```

```

        exit(0);
    }
void SERVER( )    //服务器端
{
    shmid=shmget(SHMKEY,1024,0777|IPC_CREAT);    //创建共享区，并先取得 id
    addr=shmat(shmid,0,0);    //通过 id 取得起始地址 addr
    do
    {
        *addr=-1;    //将数据取走
        while(*addr== -1);
        printf("(server)received ,");
        printf("server *addr %d\n", *addr);
    }while(*addr);
    shmctl(shmid,IPC_RMID,0);
    exit(0);
}
int main( )
{
    while ((i=fork())== -1);
    if (!i) SERVER();
    while((i=fork())== -1);
    if (!i) CLIENT();
    wait(0);
    wait(0);
}

```

4. 【程序运行结果截图】

```

administrator@ubuntu:~$ ./li3_17
(client)sent, client i: 5
(server)received ,server *addr 5
(client)sent, client i: 4
(server)received ,server *addr 4
(client)sent, client i: 3
(server)received ,server *addr 3
(client)sent, client i: 2
(server)received ,server *addr 2
(client)sent, client i: 1
(server)received ,server *addr 1
(client)sent, client i: 0
(server)received ,server *addr 0

```

```

administrator@ubuntu:~$ ./li3_17
(client)sent, client i: 5
(server)received ,server *addr 5
(server)received ,server *addr 4
(client)sent, client i: 4
(client)sent, client i: 3
(server)received ,server *addr 3
(client)sent, client i: 2
(server)received ,server *addr 2
(client)sent, client i: 1
(server)received ,server *addr 1
(client)sent, client i: 0
(server)received ,server *addr 0

```

5. 实验思考与扩充

- (1) 编辑并运行程序，详细分析程序的运行结果。
- (2) 在此基础上对程序进行修改：使得 **CLIENT** 向共享区连续发送 20 个整数, **SERVER** 从共享区接收 20 个整数，并输出之。
- (3) 运行该程序时，如果出现了右边那种情况，如何解释？