同济大学计算机网络

实验报告

姓名：涂远鹏-1652262

题目：进程间通信

1.无名管道(01 目录)

(1)测试程序 test1-1

答:

Fork 子进程:

```
pid_t pid;
pid=fork();
if(pid==-1||pid>0)
    return 0;
int pfd[2];
pid_t cpid;
char buf;
if (pipe(pfd) == -1)
{
    perror("pipe");
    exit(EXIT_FAILURE);
}
cpid = fork();
if (cpid == -1)
{
    perror("fork");
    exit(EXIT_FAILURE);
}
```

测试结果如下，父进程向子进程发数据成功:

```
root       2417      1  0 15:36 pts/0    00:00:00 ./test1-1
root       2418   2417  0 15:36 pts/0    00:00:00 ./test1-1
[root@RHEL74-SVR 01]# make
cc     test1-1.c   -o test1-1
[root@RHEL74-SVR 01]# ./test1-1
[root@RHEL74-SVR 01]# wo ai shen jian!
```

(2)测试程序 test1-2

答:

测试结果如下，子进程向父进程发数据成功:

```
[root@RHEL74-SVR 01]# make
cc     test1-2.c   -o test1-2
[root@RHEL74-SVR 01]# ./test1-2
[root@RHEL74-SVR 01]# wo ai shen jian test1-2!
ps -ef
```

```
root       2418   2417  0 15:36 pts/0    00:00:00 ./test1-1
root       2419      2  0 15:38 ?        00:00:00 [kworker/0:1]
root       2427      2  0 15:43 ?        00:00:00 [kworker/0:2]
root       2437      1  0 15:45 pts/0    00:00:00 ./test1-2
root       2438   2437  0 15:45 pts/0    00:00:00 ./test1-2
root       2439   2318  0 15:46 pts/0    00:00:00 ps -ef
[root@RHEL74-SVR 01]#
```

(3)测试程序 test1-3

答:

测试结果如下，双向传输数据成功:

```
[root@RHEL74-SVR 01]# make
cc     test1-3.c   -o test1-3
[root@RHEL74-SVR 01]# ./test1-3
[root@RHEL74-SVR 01]# wo ai shen jian test1-3!(child)
wo ai shen jian test1-3!(parent)
```

```
root        2437      1  0 15:45 pts/0    00:00:00 ./test1-2
root        2438   2437  0 15:45 pts/0    00:00:00 ./test1-2
root        2440      2  0 15:46 ?        00:00:00 [kworker/0:3]
root        2449      1  0 15:49 pts/0    00:00:00 ./test1-3
root        2450   2449  0 15:49 pts/0    00:00:00 ./test1-3
root        2451   2318  0 15:49 pts/0    00:00:00 ps -ef
[root@RHEL74-SVR 01]#
```

(4)无名管道方式传递的数据类型，长度是否有限制?

答:

管道传递的是更加本质的纯字节流。这个字节流可以表达任意内容，两方程序协调一致就好，管道不关心其具体内容。理由很简单：因为只能是字节流，不可能是任何别的东西。因为管道只是数据的通道，唯有纯字节流不做任何处理，就能在所有设备之中畅通无阻。只要不是字节流，传输流程中就必然要到处增加转换、翻译的环节，这个效率浪费是不必要的。

当管道一端不断地读取数据，另一端却不输出数据。根据 unix 的实现机制当管道读满是输出端自动阻塞。所以这个管道是有大小的 PIPE_SIZE 是管道的最大值一般为 64K

(5)能否在独立进程间用无名管道通信?

答:

无名管道通信的进程，它们的关系一定是父子进程的关系或者是兄弟进程之间，所以不能使用无名管道在非父子独立进程间通信.管道.只能用于父子进程或者兄弟进程之间(具有亲缘关系的进程);

2.有名管道(02 目录)

(1)测试程序 test2-1:

答:

测试结果如下，父进程向子进程发数据成功:

```
[root@RHEL74-SVR 02]# ./test2-1
[root@RHEL74-SVR 02]# wo ai shen jian test2-1!
ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root        2482       1  0 16:01 ?        00:00:00 /usr/sbin/anacron -s
root        2485       2  0 16:03 ?        00:00:00 [kworker/0:3]
root        2488       2  0 16:08 ?        00:00:00 [kworker/0:1]
root        2533       1  0 16:11 pts/0    00:00:00 ./test2-1
root        2534    2533  0 16:11 pts/0    00:00:00 ./test2-1
root        2535    2318  0 16:11 pts/0    00:00:00 ps -ef
[root@RHEL74-SVR 02]#
```

(2)测试程序 test2-2:

答:

测试结果如下，子进程向父进程发数据成功:

```
[root@RHEL74-SVR 02]# ./test2-2
[root@RHEL74-SVR 02]# wo ai shen jian test2-2!
ps -ef
root        2582       2  0 16:18 ?        00:00:00 [kworker/0:3]
root        2611       1  0 16:20 pts/0    00:00:00 ./test2-1
root        2612    2611  0 16:20 pts/0    00:00:00 ./test2-1
root        2614       1  0 16:20 pts/0    00:00:00 ./test2-2
root        2615    2614  0 16:20 pts/0    00:00:00 ./test2-2
root        2616    2318  0 16:21 pts/0    00:00:00 ps -ef
[root@RHEL74-SVR 02]#
```

(3)测试程序 test2-3:
答:
测试结果如下，子进程与父进程双向传输数据成功:

```
[root@RHEL74-SVR 02]# ./test2-3
[root@RHEL74-SVR 02]# parent:wo ai shen jian test2-3!(to child)
child:wo ai shen jian test2-3!(to parent)
```

```
root       2624      2  0 16:23 ?        00:00:01 [kworker/0:0]
root       2638      1  0 16:28 pts/0    00:00:00 ./test2-3
root       2639   2638  0 16:28 pts/0    00:00:00 ./test2-3
root       2640   2318  0 16:28 pts/0    00:00:00 ps -ef
[root@RHEL74-SVR 02]#
```

(4)测试程序 test2-4-1/test2-4-2，test2-4-1 向 test2-4-2 发送数据:
答:
Test2-4-1 向 test2-4-2 发送数据成功:

```
[root@RHEL74-SVR 02]# ./test2-4-1
[root@RHEL74-SVR 02]# Process 2755 opening FIFO O_WRONLY
write返回内容:wo ai shen jian test2-4-1!
```

```
[root@RHEL74-SVR 02]# ./test2-4-2
[root@RHEL74-SVR 02]# Process 2757 opening FIFO O_RDONLY
read:wo ai shen jian test2-4-1!
```

(5)测试程序 test2-5-1/test2-5-2，test2-5-1 与 test2-5-2 双向发送数据:
答:

命名管道可以在任意进程间通信,通信是双向的,任意一端都可读可写,但是在同一时间只能有一端读,一端写。一个管道不能同时双向读写，只能建立两个管道实现双向读写或者串行在一个管道轮流读写,所以可以通过轮流读写实现双向发送数据:

```
[root@RHEL74-SVR 02]# make
cc     test2-5-1.c   -o test2-5-1
cc     test2-5-2.c   -o test2-5-2
[root@RHEL74-SVR 02]# ./test2-5-1
[root@RHEL74-SVR 02]# ./test2-5-2
[root@RHEL74-SVR 02]# write返回内容:wo ai shen jian test2-5-1!
read返回内容:wo ai shen jian test2-5-1!
write返回内容:wo ai shen jian test2-5-1!
read返回内容:wo ai shen jian test2-5-1!
```

(6)有名管道方式传递的数据类型，长度是否有限制？和无名管道相比是否有区别?
答:

管道所传开的是无格式字节流，这就要求管道的读出方和写入方必须事先约定好数据的格式 管道的缓冲区是有限的（管道制存在于内存中，在管道创建时，为缓冲区分配一个页面大小）当要写入的数据量大于 PIPE_BUF 时,Unix 将不再保证写入的原子性。在写满所有 FIFO 空闲缓冲区后，写操作返回。当要写入的数据量不大于 PIPE_BUF 时,unix 将保证写入的原子性。

3. 信号方式(03 目录)

(1)测试程序 test3-1-1/test3-1-2:

答:

SIGUSR1 使程序输出并继续运行而 SIGUSR2 会使程序输出并退出

```
[root@RHEL74-SVR 03]# ./test3-1-2
[root@RHEL74-SVR 03]# ./test3-1-1
[root@RHEL74-SVR 03]#
OUCH1! - I got signal 10
kill -2 2257
OUCH2! - I got signal 12

[root@RHEL74-SVR 03]#
```

接受 kill -2 2257 前后，test[sub-10]被杀死:

```
postfix    2076   2063  0 20:18 ?        00:00:00 qmgr -l -t unix -u
root       2126   1070  0 20:18 ?        00:00:00 ./test [sub-08 01:05:12]
root       2257   1070  0 20:18 ?        00:00:00 ./test [sub-10 01:05:10]
root       2260   1438  0 20:19 ?        00:00:09 sshd: root@pts/0
root       2264   2260  0 20:19 pts/0    00:00:00 -bash
root       2304   1438  0 20:19 ?        00:00:00 sshd: root@pts/1
root       2308   2304  0 20:19 pts/1    00:00:00 -bash
```

```
root       2126   1070  0 20:18 ?        00:00:00 ./test [sub-08 01:07:33]
root       2260   1438  0 20:19 ?        00:00:09 sshd: root@pts/0
root       2264   2260  0 20:19 pts/0    00:00:00 -bash
root       2304   1438  0 20:19 ?        00:00:00 sshd: root@pts/1
root       2308   2304  0 20:19 pts/1    00:00:00 -bash
```

(2) 信号能否带数据?

答:

信号可以带数据

(3)哪几个信号不能被截获并重定义?

答:

信号 SIGKILL 和 SIGSTOP 不能被截获并重定义，因为如果应用程序可以忽略这 2 个信号，系统管理无法杀死、暂停进程，无法对系统进行管理。

额外附加题:

1.首先创建一个 test1-1 进程，并查看到它的 pid 为 3204:

```
[root@RHEL74-SVR 01]# ./test1-1
1652262
1652262
1652262
ps
```

```
root       3153      2  0 20:31 ?        00:00:00 [kworker/0:1]
root       3174      1  0 20:33 pts/0    00:00:00 ./test-extra-1
root       3200      1  0 20:35 pts/0    00:00:00 ./test-extra-1
root       3204   2939  0 20:35 pts/1    00:00:00 ./test1-1
root       3205   3001  0 20:35 pts/2    00:00:00 ps -ef
[root@RHEL74-SVR homework_second]#
```

2. 随后在 test-extra-1.c 中加入 kill(3204,SIGKILL)，杀死 test1-1 进程：

192.168.80.230 × 192.168.80.230 (1) 192.168.80

```
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
int main()
{
        pid_t pid_kill;
        int pid_create;
        char buf_fp[10] = { 0 };
        FILE *fp = popen("ps -e | grep \'tes
        pid_kill = fork();
        if (pid_kill == -1 || pid_kill > 0)
                return 0;
        }
        fgets(buf_fp, 10, fp);
        pid_create = atoi(buf_fp);
        kill(pid_create, SIGUSR1);
        sleep(5);

        kill(2995,SIGKILL);
        sleep(5);

        kill(pid_create, SIGUSR2);
        sleep(5);
        kill(pid_create, SIGINT);
        while (1) {
                sleep(1);
        }
}
~
```

3. 在 test-extra-2.c 中加入 find_pid()函数查找进程名为 test1-1 的进程，如果查不到输出 not found,否则输出 test1-1 进程的个数：

```
void find_pid()
{
    FILE* fp=NULL;
    int count=0;
    int BUFSZ=100;
    char buf[BUFSZ];
    char command[150];


    //system() 也可以检测进程是否存在，但非常耗费资源
    sprintf(command, "ps -C %s|wc -l", "test1-1" );


    if((fp = popen(command,"r")) == NULL)
    {
        cout<<"(fp = popen(command,\"r\")) == NULL"<<endl;
    }

    if( (fgets(buf,BUFSZ,fp))!= NULL )
    {
        count = atoi(buf);

        if((count - 1) == 0)
        {

            printf("%s not found\n","test1-1");

            //可以用system()开启test
        }
```

4. 运行 test-extra-2 与 test-extra-1 进程，测试结果如下：

收到 SIGUSER1 信号前存在一个 test1-1 进程，在收到 SIGUSER2 之后显示 test1-1 not found：

```
[root@RHEL74-SVR 03]# ./test-extra-2
[root@RHEL74-SVR 03]# ./test-extra-1
[root@RHEL74-SVR 03]#
OUCH1! - I got signal 10
process : test total is 1

OUCH2! - I got signal 12
test1-1 not found
```

说明 test1-1 被杀死，在另外一个窗口也可查看到结果，说明 test-extra-2 收到了 kill -9 3204 信号：

```
[root@RHEL74-SVR 01]# ./test1-1
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
1652262
已杀死
[root@RHEL74-SVR 01]#
```

(4)测试程序 test3-2-1/test3-2-2:

答:

测试结果如下:

```
[root@RHEL74-SVR 03]# ./test3-2-2
[root@RHEL74-SVR 03]# ./test3-2-1
[root@RHEL74-SVR 03]# read:wo ai shen jian wo
write:hello world!
read内容为:hello world!
write:hello world!
read:hello world!
write:hello world!
read内容为:hello world!
write:hello world!
read:hello world!
write:hello world!
read内容为:hello world!
write:hello world!
read:hello world!
write:hello world!
read内容为:hello world!
write:hello world!
read:hello world!
write:hello world!
read内容为:hello world!
write:hello world!
```

4. 消息队列方式(04 目录)

(1)测试 test4-1-1/test4-1-2,建立消息队列方式，从 test4-1-1 向 test4-1-2 单项传递数据:

答:

测试结果如下:

Test4-1-1 运行之前没有数据传输:

```
[root@RHEL74-SVR 04]# ./test4-1-2
[root@RHEL74-SVR 04]# ps
   PID TTY          TIME CMD
  2264 pts/0    00:00:00 bash
  5685 pts/0    00:00:00 test4-1-2
  5686 pts/0    00:00:00 ps
[root@RHEL74-SVR 04]# ./test4-1-2
```

Test4-1-1 运行后传输数据：

```
[root@RHEL74-SVR 04]# ./test4-1-1
[root@RHEL74-SVR 04]# send:hello world!
recv:hello world!
send:hello world!
recv:hello world!
send:hello world!
recv:hello world!
send:hello world!
recv:hello world!
send:hello world!
recv:hello world!
send:hello world!
recv:hello world!
send:hello world!
recv:hello world!
```

(2)测试 test4-2-1/test4-2-2,建立消息队列方式，test4-2-1 与 test4-2-2 双向传递数据:
答:
测试结果如下，双向传输数据:

```
[root@RHEL74-SVR 04]# ./test4-2-1
[root@RHEL74-SVR 04]# send:hello world1!
recv:hello world2!
send:hello world1!
recv:hello world2!
send:hello world1!
recv:hello world2!
send:hello world1!
recv:hello world2!
send:hello world1!
recv:hello world2!
send:hello world1!
recv:hello world2!
send:hello world1!
```

```
[root@RHEL74-SVR 04]# ./test4-2-2
[root@RHEL74-SVR 04]# send:hello world2!
recv:hello world1!
send:hello world2!
recv:hello world1!
send:hello world2!
recv:hello world1!
send:hello world2!
recv:hello world1!
send:hello world2!
recv:hello world1!
send:hello world2!
recv:hello world1!
```

杀死一端后传输失败:

```
send:hello world1!
recv:hello world2!
send:hello world1!
recv:hello world2!
send:hello world1!
recv:hello world2!
send:hello world1!
msgsnd failed
```

(3)消息队列方式传递的数据类型，长度是否有限制？和无名/有名管道相比区别在哪?
答:

消息数据的类型一般为 char，自己构建时也可以存放任意形式的任意数据消息的大小存在一个内部的限制。在 Linux 中，它在 Linux/msg.h 中的定义如下：#define MSGMAX

8192 消息的总大小不能超过 8192 个字节，这其中包括了 mtype 成员，它的长度为 4 个字节(long 类型）消息队列和管道对每个数据都有一个最大长度的限制。

与命名管道相比，消息队列的优势/区别在于:

1、消息队列也可以独立于发开和接收进程而存在，从而消除了在同步命名管道的打开和关闭时可能产生的困难。

2、同时通过发开消息还可以避免命名管道的同步和阻塞问题，不需要由进程自己来提供同步方法。

3、接收程序可以通过消息类型有选择地接收数据，而不是像命名管道中那样，只能默认地接收

5.共享内存方式(05 目录)

(1)测试程序 test5-1/test5-2:

答:

此处由于不能使读写同时进行，需要设置写入锁，当一端写入时另一端无法写，使两端写入的内容不会冲突和覆盖，测试结果如下:

```
[root@RHEL74-SVR 05]# ./test5-1
[root@RHEL74-SVR 05]# 发送内容为:wo ai shen jian test5-1!
./test5-2
[root@RHEL74-SVR 05]# 发送内容为:wo ai shen jian test5-1!
读取内容为:wo ai shen jian test5-1!
发送内容为:wo ai shen jian test5-1!
读取内容为:wo ai shen jian test5-1!
发送内容为:wo ai shen jian test5-1!
读取内容为:wo ai shen jian test5-1!
发送内容为:wo ai shen jian test5-1!
读取内容为:wo ai shen jian test5-1!
发送内容为:wo ai shen jian test5-1!
读取内容为:wo ai shen jian test5-1!
发送内容为:wo ai shen jian test5-1!
读取内容为:wo ai shen jian test5-1!
发送内容为:wo ai shen jian test5-1!
读取内容为:wo ai shen jian test5-1!
发送内容为:wo ai shen jian test5-1!
读取内容为:wo ai shen jian test5-1!
发送内容为:wo ai shen jian test5-1!
读取内容为:wo ai shen jian test5-1!
发送内容为:wo ai shen jian test5-1!
读取内容为:wo ai shen jian test5-1!
```

(2)能否只能在父子进程间共享内存？还是可以独立进程间共享？

答：可以在独立进程间共享。

(3)如果两个进程同时写，共享内存内容是否会乱？如何防止共享内存内容乱？

答:

同时写共享内存会混乱。共享内存并未提供同步机制，也就是说，在第一个进程结束对共享内存的写操作之前，并无自动机制可以阻止第二个进程开始对它进行读取。通常需要用其他的机制来同步对共享内存的访问，例如信号量。

在代码中实现了简单化的信号来控制同步读写。严格实现可以每次进入读写时发送一个进入读写信号给另一个进程，读写退出时再发开一个退出读写信号给另一个进程，另一个进程可以根据这些信号做出读写的控制，防止共享内存内容混乱。

6.Unix 套接字方式(06 目录)

(1)测试程序 test6-1-1/test6-1-2

答:

测试结果如下，双向通信:

```
[root@RHEL74-SVR 06]# ./test6-1-2
[root@RHEL74-SVR 06]# 与客户端连接成功!
服务端发送内容:wo ai shen jian test6-1!
连接成功
客户端发送内容:wo ai shen jian test6-1!
客户端读取内容:wo ai shen jian test6-1!
服务端读取内容:wo ai shen jian test6-1!
服务端发送内容:wo ai shen jian test6-1!
客户端发送内容:wo ai shen jian test6-1!
客户端读取内容:wo ai shen jian test6-1!
服务端读取内容:wo ai shen jian test6-1!
客户端发送内容:wo ai shen jian test6-1!
服务端发送内容:wo ai shen jian test6-1!
服务端读取内容:wo ai shen jian test6-1!
客户端读取内容:wo ai shen jian test6-1!
服务端发送内容:wo ai shen jian test6-1!
客户端发送内容:wo ai shen jian test6-1!
客户端读取内容:wo ai shen jian test6-1!
服务端读取内容:wo ai shen jian test6-1!
客户端发送内容:wo ai shen jian test6-1!
服务端发送内容:wo ai shen jian test6-1!
服务端读取内容:wo ai shen jian test6-1!
客户端读取内容:wo ai shen jian test6-1!
服务端发送内容:wo ai shen jian test6-1!
客户端发送内容:wo ai shen jian test6-1!
```

杀掉一个以后另一个就停了：

```
[root@RHEL74-SVR 06]# 客户端发送内容:wo ai shen jian test6-1!
客户端读取内容:wo ai shen jian test6-1!
```

(2)Unix 类型 Socket 建立/使用与 TCP Socket 相比有什么相同和不同点？

答：

1. 相同点:几乎都是相同的，有两种类型的套接字，字节流套接字和数据报套接字，节流套接字类似于 TCP。

不同点:

2. tcp socket 是在两台主机之间进行通信,其 ip 地址不同进程间通信是在一台主机上两进程之间的通信 client 连接本机的 ip 地址 unix 域套接字与 TCP 套接字相比较，在同一台主机的传输速度前者是后者的两倍

2.Unix 域套接字可以在同一台主机上的各进程之间传递描述符

3.Unix 域套接字与传统套接字的区别是用路径名来表示协议族的描述

4.Unix 域套接字的地址结构不同，在 connect()函数发现监听套接字队列满时，会立刻返回错误与 TCP 不同

5.Unix Socket 分为 SOCK_STREAM(流套接字)和 SOCK_DGRAM(数据包套接字),由于是在本机通过内核通信,不会丢包也不会出现发送包的次序和接收包的次序不一致的问题这与 TCP 则不同

6.UDS 传输不需要经过网络协议栈,不需要打包拆包等操作,只是数据的拷贝过程

(3)unix 类型 Socket 是否有阻塞和非阻塞方式？是否能通过 select 来读写？写满后是返回不可写还是继续可写而导致数据丢失？

答：

1. 存在阻塞与非阻塞形式：

与 TCP 设置非阻塞方式类似，通过以下代码实现：

```
int flags=fcntl(s,F_GETFL,0);
fcntl(s,F_SETFL,flags|O_NONBLOCK);
~
```

2.也可以通过 select 读写：

```
select(s+1,&fdR,NULL,NULL,NULL);
~
```

3.写满后不可继续写入：

```
while (1)
{
    if(sendto(sock_cli, sendbuf, BUFFER_SIZE, 0, (struct sockaddr *) &servaddr, slen)<0)
        break;
    printf("client_send:%s\n",sendbuf);
    recvfrom(sock_cli, recvbuf, BUFFER_SIZE, 0, (struct sockaddr *) &servaddr, &slen);
    printf("client_recv:%s\n",recvbuf);
    //sleep(1);
}
```

```
client_recv:wo ai shen jian test6-1hellohellohello
client_send:wo ai shen jian test6-1hellohellohello!
server_recv:wo ai shen jian test6-1hellohellohello!
server_send:wo ai shen jian test6-1hellohellohello
client_recv:wo ai shen jian test6-1hellohellohello
client_send:wo ai shen jian test6-1hellohellohello!
server_recv:wo ai shen jian test6-1hellohellohello!
server_send:wo ai shen jian test6-1hellohellohello
client_recv:wo ai shen jian test6-1hellohellohello
client_send:wo ai shen jian test6-1hellohellohello!
[root@RHEL74-SVR 06]#
```

7.锁机制(07 目录)

(1)测试程序 test7-1-1/test7-1-2:

答：测试结果如下，直到 7-1-1 释放文件写锁之后 7-1-2 才退出阻塞状态读取写入内容：

```
[root@RHEL74-SVR 07]# ./test7-1-1
[root@RHEL74-SVR 07]# Write lock set by 2827
write:11
Release lock by 2827
```

```
[root@RHEL74-SVR 07]# ./test7-1-2
[root@RHEL74-SVR 07]# Read lock set by 2825
read:11
process 1st
Release lock by 2825
```

(2)测试程序 test7-2-1/test7-2-2:

答：测试结果如下，直到 7-2-1 释放文件写锁之后 7-2-2 才退出阻塞状态读取写入内容：

```
[root@RHEL74-SVR 07]# Write lock set by 2834
./test7-2-2
[root@RHEL74-SVR 07]# Write lock already set by 2834
write:11
Release lock by 2834
Read lock set by 2836
read:11
process 1st
Release lock by 2836
```

(3)锁定文件有几种方法？不同的方法对阻塞/非阻塞方式的 fd 是否有区别？

答：

Flock 主要三种操作类型：

LOCK_SH，共享锁，多个进程可以使用同一把锁，常被用作读共享锁；

LOCK_EX，排他锁，同时只允许一个进程使用，常被用作写锁；

LOCK_UN，释放锁；

进程使用 flock 尝试锁文件时，如果文件已经被其他进程锁住，进程会被阻塞直到锁被释放掉，或者在调用 flock 的时候，采用 LOCK_NB 参数，在尝试锁住该文件的时候，发现已经被其他服务锁住，会返回错误，errno 错误码为 EWOULDBLOCK。即提供两种工作模式：阻塞与非阻塞类型。

服务会阻塞等待直到锁被释放：flock(lockfd,LOCK_EX)服务会返回错误发现文件已经被锁住时：ret = flock(lockfd,LOCK_EX|LOCK_NB)同时 ret = -1, errno = EWOULDBLOCK

(4)在一个程序对文件加写锁后，另一个程序不加锁直接读写(设置为阻塞)，是阻塞在 read/write，还是直接返回失败？

答：

Read 正常返回空内容并且另一个进程不阻塞。

(5)在一个程序对文件加写锁后，另一个程序不加锁直接读写(设置为非阻塞)，是阻塞在 read/write，还是直接返回失败？

答：

Read 正常返回空内容并且另一个进程不阻塞。