

#0.LINUX 知识补充 - 进程间的通信

1 无名管道

test1-1 test1-2

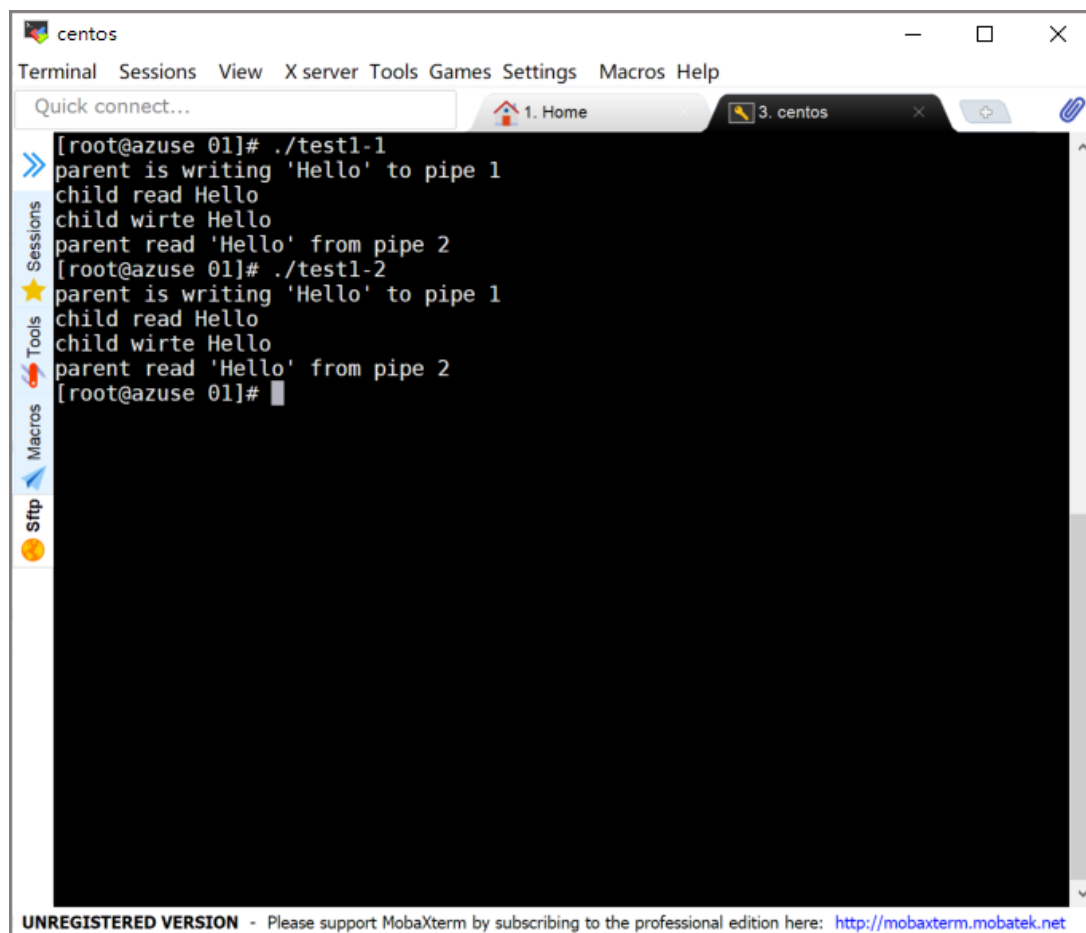
使用`pipe()`建立管道，参数为`int[2]`，其中`p[0]`用来读，`p[1]`用来写。test1-1和test1-2（代码合并为一份）

```
//test1-2/1.c
#define _POSIX_SOURCE
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <error.h>
#include <stdlib.h>

int main() {
    char *send = "Hello";
    char buf[80];
    int p1[2], p2[2];

    if (pipe(p1) != 0)
        perror("first pipe() failed");
    else if (pipe(p2) != 0)
        perror("second pipe() failed");
    else if (fork() == 0) {
        //close(p1[1]);
        //close(p2[0]);
        if (read(p1[0], buf, sizeof(buf)) == -1)
            perror("read() error in parent");
        else {
            printf("child read %s\n", buf);
            if (write(p2[1], buf, strlen(buf)+1) == -1)
                perror("write() error in child");
            else
                printf("child wirte %s\n", send);
        }
        exit(0);
    }
    else {
        //close(p1[0]);
        //close(p2[1]);
        printf("parent is writing '%s' to pipe 1\n", send);
        if (write(p1[1], send, strlen(send)+1) == -1)
            perror("write() error in parent");
        else if (read(p2[0], buf, sizeof(buf)) == -1)
            perror("read() error in parent");
        else printf("parent read '%s' from pipe 2\n", buf);
    }
    return 0;
}
```

运行结果：



```
centos
Terminal Sessions View X server Tools Games Settings Macros Help
Quick connect... 1. Home 3. centos
[ root@azuse 01 ]# ./test1-1
parent is writing 'Hello' to pipe 1
child read Hello
child wirte Hello
parent read 'Hello' from pipe 2
[ root@azuse 01 ]# ./test1-2
parent is writing 'Hello' to pipe 1
child read Hello
child wirte Hello
parent read 'Hello' from pipe 2
[ root@azuse 01 ]#
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <http://mobaxterm.mobatek.net>

test1-3 一个管道是否可以双向通信？

测试：将test1-1和1-2修改为使用同一个管道。测试结果：不可用，主进程写完后必须等到子进程读完、写完后
再读取子进程传来的数据，不然就会读到自己刚刚写进去的数据，无法正常使用。要传递数据必须建立双向来
回两条管道。

代码：

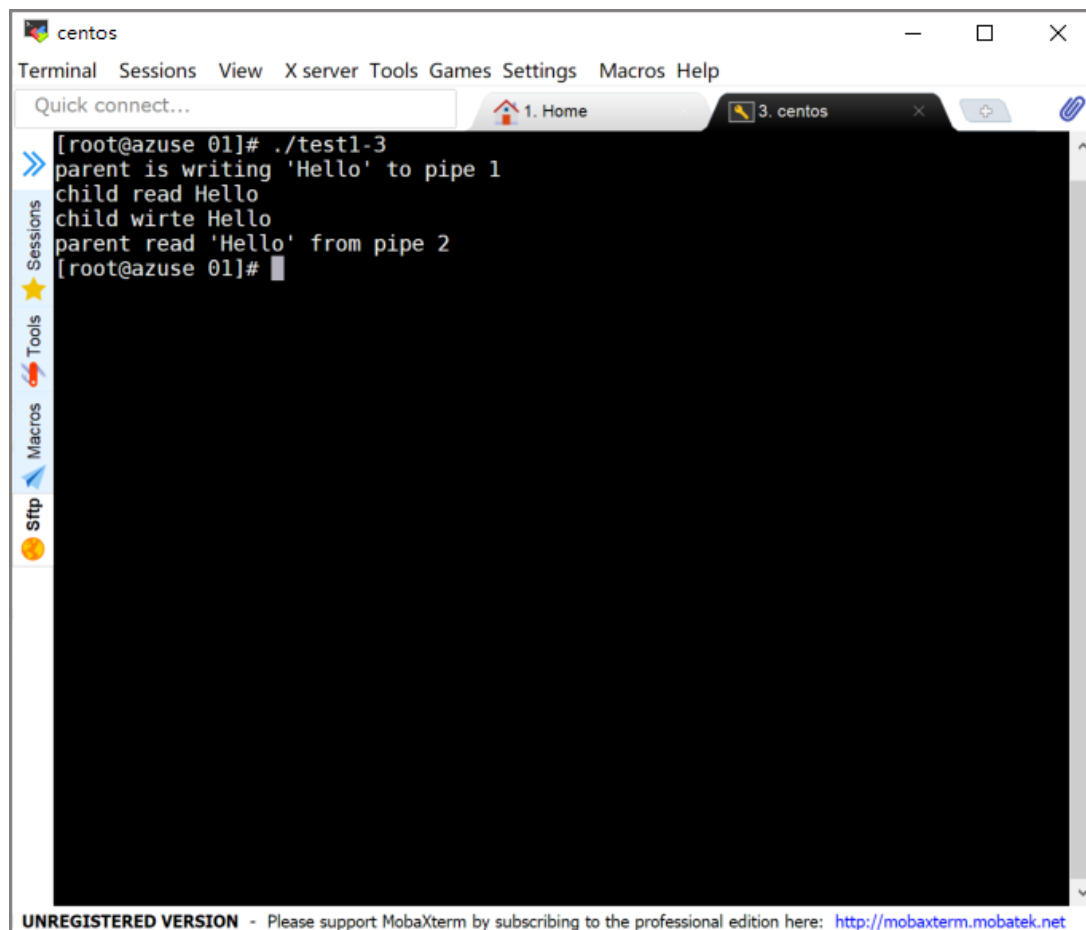
```
//test1-3.c
#define _POSIX_SOURCE
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <error.h>
#include <stdlib.h>
#include <time.h>

int main() {
    char *send ="Hello";
    char buf[80];
    int p1[2];

    if (pipe(p1) != 0)
        perror("first pipe() failed");
```

```
else if (fork() == 0) {
    //close(p1[1]);
    //close(p2[0]);
    if (read(p1[0], buf, sizeof(buf)) == -1)
        perror("read() error in parent");
    else {
        printf("child read %s\n", buf);
        if (write(p1[1], buf, strlen(buf)+1) == -1)
            perror("write() error in child");
        else
            printf("child wirte %s\n", send);
    }
    exit(0);
}
else {
    //close(p1[0]);
    //close(p2[1]);
    printf("parent is writing '%s' to pipe 1\n", send);
    if (write(p1[1], send, strlen(send)+1) == -1)
        perror("write() error in parent");
    sleep(1); //这里加了sleep来保证子进程已经写入了数据
    if (read(p1[0], buf, sizeof(buf)) == -1)
        perror("read() error in parent");
    else printf("parent read '%s' from pipe 2\n", buf);
}
return 0;
}
```

运行结果：



```
centos
Terminal Sessions View X server Tools Games Settings Macros Help
Quick connect... 1. Home 3. centos
[root@azuse 01]# ./test1-3
parent is writing 'Hello' to pipe 1
child read Hello
child wirte Hello
parent read 'Hello' from pipe 2
[root@azuse 01]#
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <http://mobaxterm.mobatek.net>

能否在独立进程间用无名管道通信？

不行，因为管道是没有名字的，控制管道的仅仅只是一个局部变量，不能在进程间独立进程间传递。

有名管道

有名管道使用mkfifo建立，使用open函数打开，read/write读写，可以用于父子程序之间的双向通信

test2-1 父进程向子进程发送数据

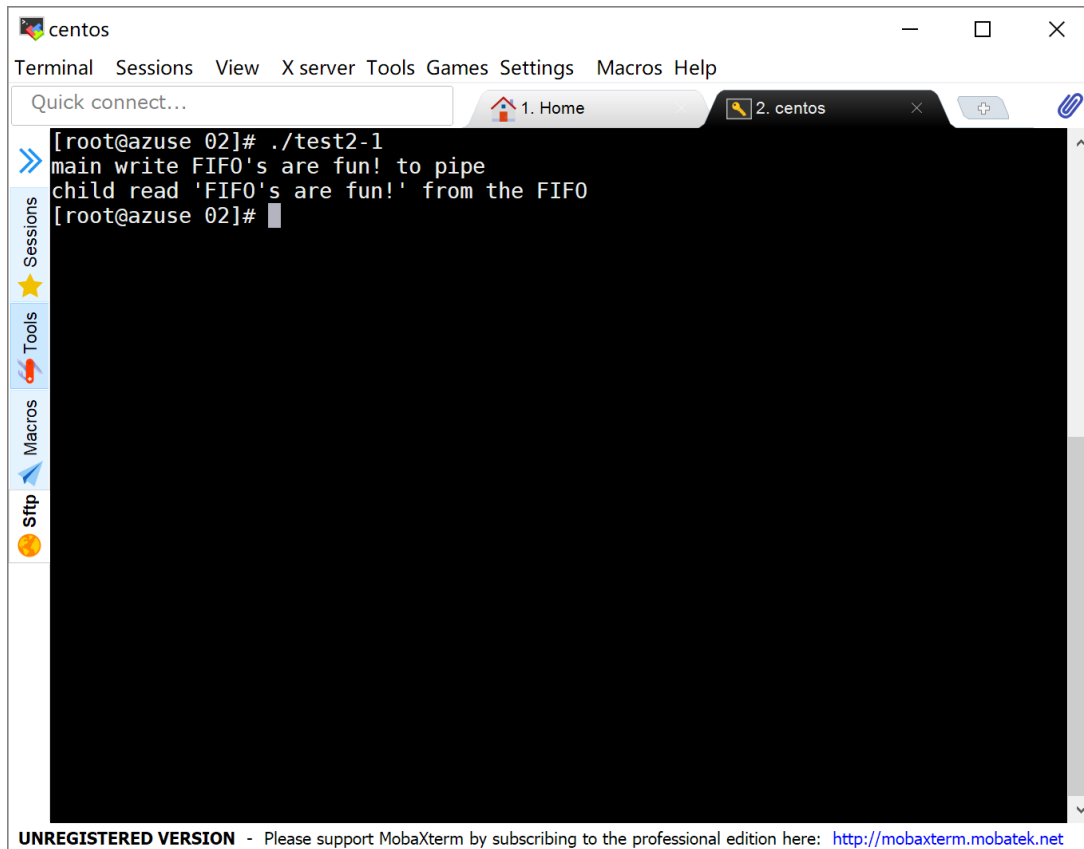
```
//test2-1.c
#define _POSIX_SOURCE
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/prctl.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
    char fn[] = "temp.fifo";
```

```
char out[20] = "FIFO's are fun!", in[20];
int rfd, wfd;

if (mkfifo(fn, S_IRWXU) != 0)
{
    perror("mkfifo() error");
    exit(-1);
}
int pid = fork();
if (pid == 0)
{
    if ((rfd = open(fn, O_RDONLY)) < 0)
        perror("open() error for read end");
    if (read(rfd, in, sizeof(in)) == -1)
        perror("read() error");
    else
        printf("child read '%s' from the FIFO\n", in);
    close(rfd);
    return 0;
}
else
{
    if ((wfd = open(fn, O_WRONLY)) < 0)
        perror("open() error for write end");
    if (write(wfd, out, strlen(out) + 1) == -1)
        perror("write() error");
    else
        printf("main write %s to pipe\n", out);
    close(wfd);
}
unlink(fn);
int stat;
wait(NULL);
}
```

效果:



The screenshot shows a MobaXterm terminal window with the title bar 'centos'. The menu bar includes 'Terminal', 'Sessions', 'View', 'X server Tools', 'Games', 'Settings', 'Macros', and 'Help'. The 'Quick connect...' field is empty. The terminal session shows the following commands and output:

```
[root@azuse 02]# ./test2-1
main write FIFO's are fun! to pipe
child read 'FIFO's are fun!' from the FIFO
[root@azuse 02]#
```

The terminal window also displays a sidebar with icons for 'Sessions', 'Tools', 'Macros', and 'Sftp'. At the bottom, a message reads: 'UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <http://mobaxterm.mobatek.net>'.

test2-2 子进程向父进程发数据

```
//test2-2.c
#define _POSIX_SOURCE
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/prctl.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>

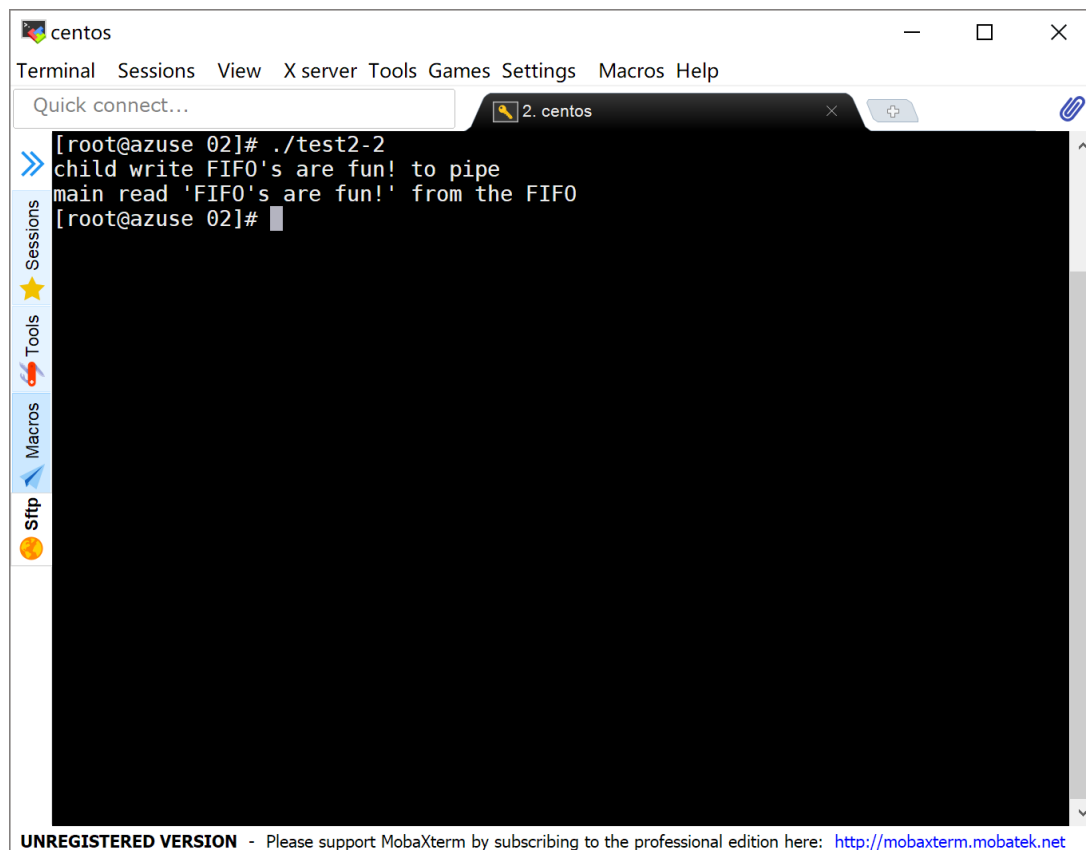
int main()
{
    char fn[] = "temp.fifo";
    char out[20] = "FIFO's are fun!", in[20];
    int rfd, wfd;

    if (mkfifo(fn, S_IRWXU) != 0)
    {
        perror("mkfifo() error");
        exit(-1);
    }
    int pid = fork();
    if (pid == 0)
```

```
{
    if ((wfd = open(fn, O_WRONLY)) < 0)
        perror("open() error for write end");
    if (write(wfd, out, strlen(out) + 1) == -1)
        perror("write() error");
    else
        printf("child write %s to pipe\n", out);
    close(wfd);

    return 0;
}
else
{
    if ((rfd = open(fn, O_RDONLY)) < 0)
        perror("open() error for read end");
    if (read(rfd, in, sizeof(in)) == -1)
        perror("read() error");
    else
        printf("main read '%s' from the FIFO\n", in);
    close(rfd);
}
unlink(fn);
int stat;
wait(NULL);
}
```

效果:



The screenshot shows a MobaXterm terminal window with the following content:

```
centos
Terminal Sessions View X server Tools Games Settings Macros Help
Quick connect...
2. centos
[root@azuse 02]# ./test2-2
child write FIFO's are fun! to pipe
main read 'FIFO's are fun!' from the FIFO
[root@azuse 02]#
```

At the bottom of the terminal window, there is a message: **UNREGISTERED VERSION** - Please support MobaXterm by subscribing to the professional edition here: <http://mobaxterm.mobatek.net>

test2-3 双向传递数据

和无名管道一样，有名管道也无法很好的进行双向通信，如果在write之后立刻read的话，会读出刚刚自己write进去的数据。因为有名管道本质就是一个FIFO的文件，是没有方向的概念的。要双向传递数据，必须建立来回两条管道。

```
//test2-3.c
#define _POSIX_SOURCE
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/prctl.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
    char fn[] = "temp.fifo";
    char out[20] = "FIFO's are fun!", in[20];
    int rfd, wfd;

    if (mkfifo(fn, S_IRWXU) != 0)
    {
        perror("mkfifo() error");
        exit(-1);
    }
    int pid = fork();
    if (pid == 0)
    {
        if ((wfd = open(fn, O_RDWR)) < 0)
            perror("open() error for write end");
        if (write(wfd, out, strlen(out) + 1) == -1)
            perror("write() error");
        else
            printf("child write %s to pipe\n", out);
        sleep(1);

        if (read(wfd, in, sizeof(in)) == -1)
            perror("read() error");
        else
            printf("child read '%s' from the FIFO\n", in);

        close(wfd);

        return 0;
    }
    else
    {
        if ((rfd = open(fn, O_RDWR)) < 0)
```



```

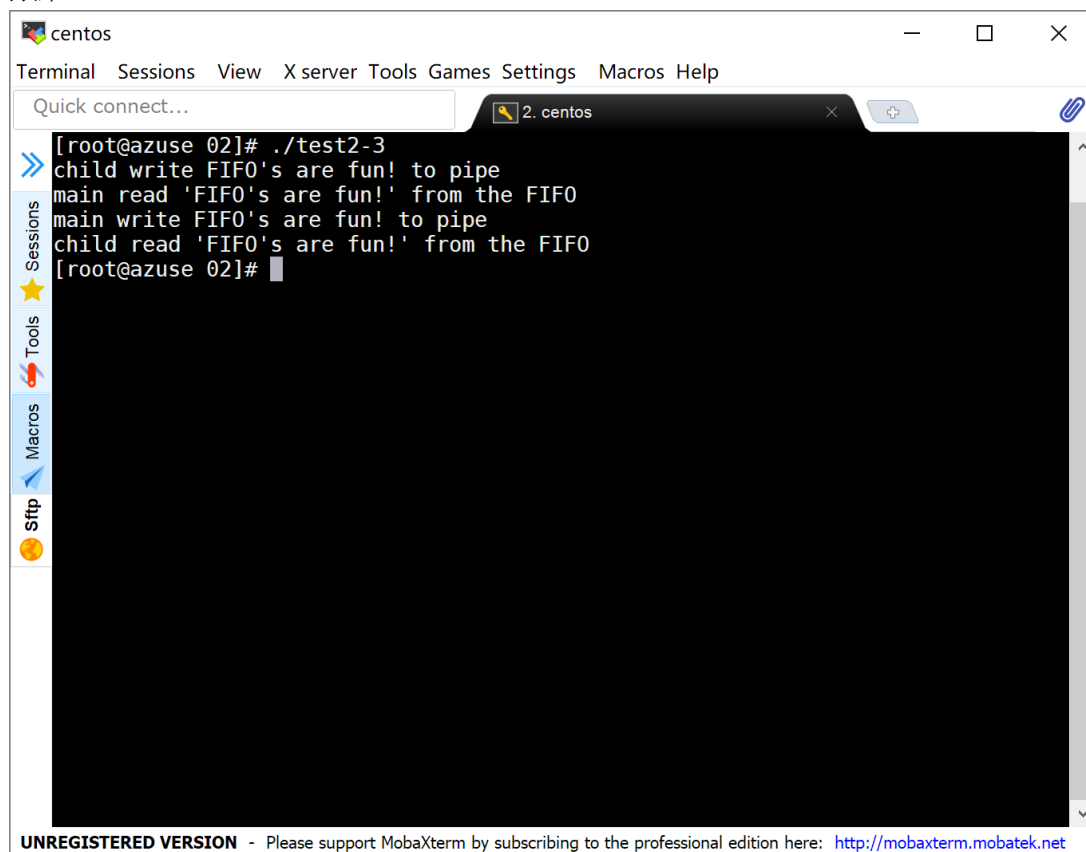
        perror("open() error for read end");
    if (read(rfd, in, sizeof(in)) == -1)
        perror("read() error");
    else
        printf("main read '%s' from the FIFO\n", in);

    if (write(rfd, out, strlen(out) + 1) == -1)
        perror("write() error");
    else
        printf("main write %s to pipe\n", out);

    close(rfd);
}
unlink(fn);
int stat;
wait(NULL);
}

```

效果:



```

centos
Terminal Sessions View X server Tools Games Settings Macros Help
Quick connect... 2. centos
[ root@azuse 02 ]# ./test2-3
child write FIFO's are fun! to pipe
main read 'FIFO's are fun!' from the FIFO
main write FIFO's are fun! to pipe
child read 'FIFO's are fun!' from the FIFO
[ root@azuse 02 ]#

```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <http://mobaxterm.mobatek.net>

test2-4-1/2 一对测试程序有名管道

test2-4-1只读打开fifo，test2-4-2只写打开fifo。运行test2-4-1后阻塞写入，write会等待test2-4-2中read函数读出数据后再返回

```

//test2-4-1.c
#define _POSIX_SOURCE
#include <sys/stat.h>

```

```

#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/prctl.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
    char fn[] = "temp.fifo";
    char out[20] = "FIFO's are fun!", in[20];test4-1-1/2从test4-1-向test4-1-2
单向传递数据
    int rfd, wfd;

    if (mkfifo(fn, 0777) != 0)
    {
        perror("mkfifo() error");
        exit(-1);
    }

    if ((wfd = open(fn, O_WRONLY)) < 0)
        perror("open() error for write end");

    if (write(wfd, out, strlen(out) + 1) == -1)
        perror("write() error");
    else
        printf("test2-4-1 write %s to pipe\n", out);

    close(wfd);

    unlink(fn);
    int stat;
    wait(NULL);
}

```

```

//test2-4-2.c
#define _POSIX_SOURCE
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/prctl.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>

```

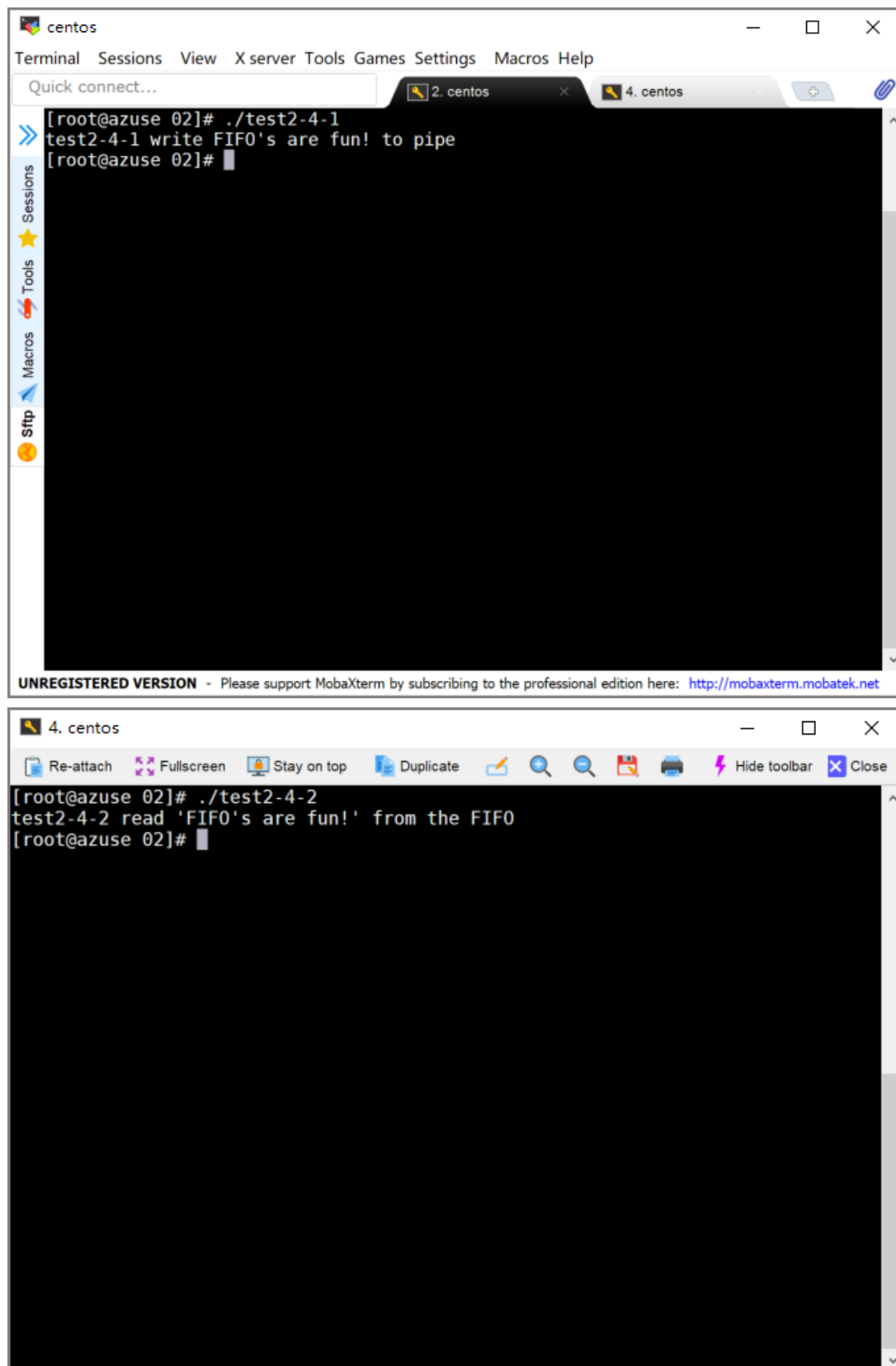
```
int main()
{
    char fn[] = "temp.fifo";
    char out[20] = "FIFO's are fun!", in[20];
    int rfd, wfd;

    // if (mkfifo(fn, S_IRWXU) != 0)
    // {
    //     perror("mkfifo() error");
    //     exit(-1);
    // }

    if ((rfd = open(fn, O_RDONLY)) < 0)
        perror("open() error for read end");
    if (read(rfd, in, sizeof(in)) == -1)
        perror("read() error");
    else
        printf("test2-4-2 read '%s' from the FIFO\n", in);

    close(rfd);
    // unlink(fn);
    int stat;
    wait(NULL);
}
```

效果:



```
centos
Terminal Sessions View X server Tools Games Settings Macros Help
Quick connect... 2. centos 4. centos
[root@azuse 02]# ./test2-4-1
test2-4-1 write FIFO's are fun! to pipe
[root@azuse 02]#

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net

4. centos
Re-attach Fullscreen Stay on top Duplicate Hide toolbar Close
[root@azuse 02]# ./test2-4-2
test2-4-2 read 'FIFO's are fun!' from the FIFO
[root@azuse 02]#
```

test2-5-1/2 两个程序间通过一个有名管道双向传递数据能否做到?

不能，因为管道无论有名无名都是半双工的。两个程序都可以读写但是因为管道就是一个FIFO的文件，程序1写入完之后立刻读取就会读出自己刚刚写入的数据，无法正常使用。

test2-5-1在write之后sleep(1)来防止读出自己写入的数据。

```
//test2-5-1.c
#define _POSIX_SOURCE
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/prctl.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
    char fn[] = "temp.fifo";
    char out[20] = "FIFO's are fun!", in[20];
    int rfd, wfd;

    if (mkfifo(fn, 0777) != 0)
    {
        perror("mkfifo() error");
        exit(-1);
    }

    if ((wfd = open(fn, O_WRONLY)) < 0)
        perror("open() error for write end");

    if (write(wfd, out, strlen(out) + 1) == -1)
        perror("write() error");
    else
        printf("test2-5-1 write %s to pipe\n", out);

    sleep(1);

    if ((rfd = open(fn, O_RDONLY)) < 0)
        perror("open() error for read end");
    if (read(rfd, in, sizeof(in)) == -1)
        perror("read() error");
    else
        printf("test2-5-1 read '%s' from the FIFO\n", in);

    close(wfd);

    unlink(fn);
    int stat;
    wait(NULL);
}
```

```
//test2-5-2.c
#define _POSIX_SOURCE
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/prctl.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
    char fn[] = "temp.fifo";
    char out[20] = "FIFO's are fun!", in[20];
    int rfd, wfd;

    // if (mkfifo(fn, S_IRWXU) != 0)
    // {
    //     perror("mkfifo() error");
    //     exit(-1);
    // }

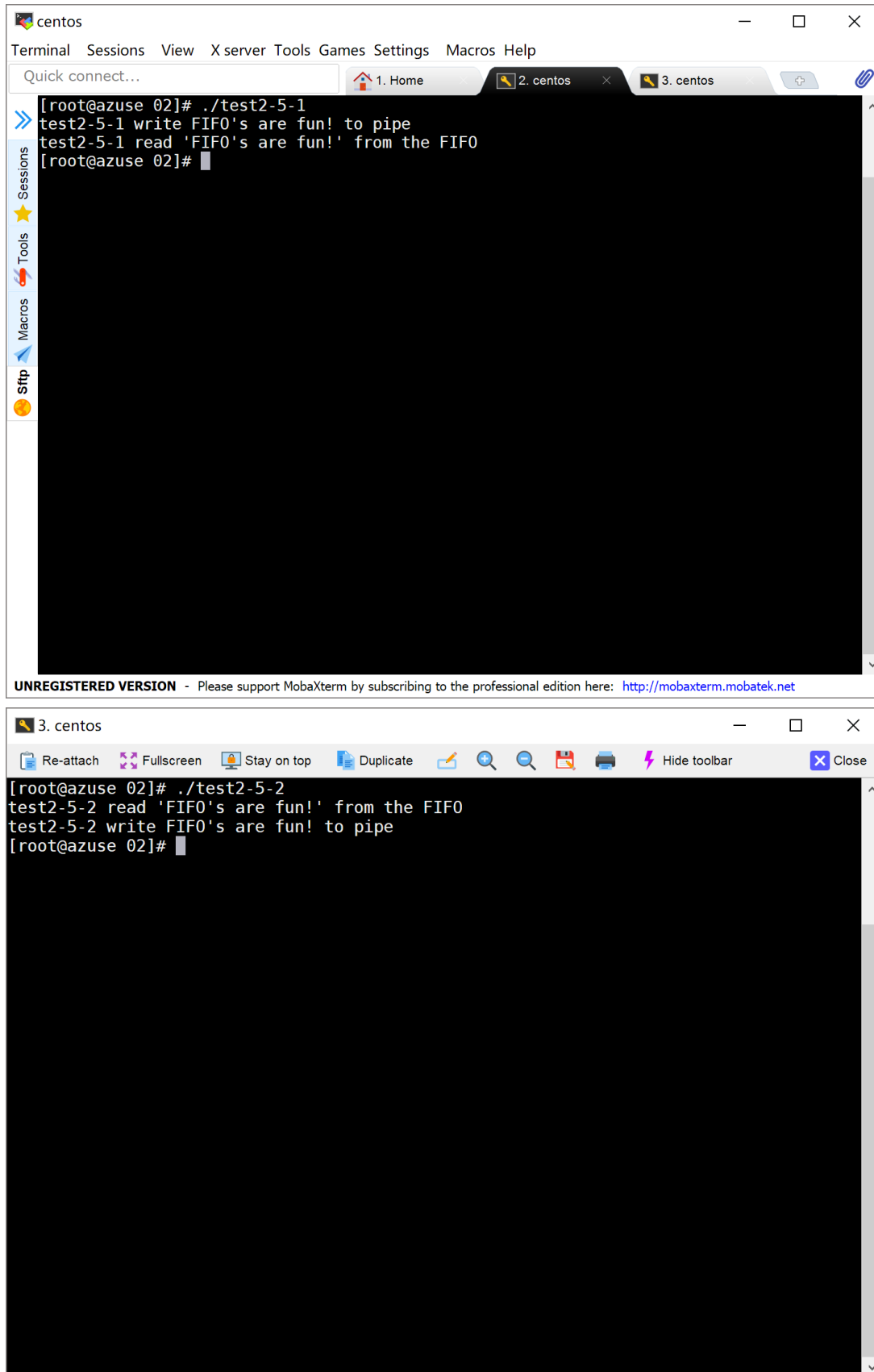
    if ((rfd = open(fn, O_RDONLY)) < 0)
        perror("open() error for read end");
    if (read(rfd, in, sizeof(in)) == -1)
        perror("read() error");
    else
        printf("test2-5-2 read '%s' from the FIFO\n", in);

    if ((wfd = open(fn, O_WRONLY)) < 0)
        perror("open() error for write end");

    if (write(wfd, out, strlen(out) + 1) == -1)
        perror("write() error");
    else
        printf("test2-5-2 write %s to pipe\n", out);

    close(rfd);
    // unlink(fn);
    int stat;
    wait(NULL);
}
```

效果:



The image displays two screenshots of a MobaXterm terminal window. The top screenshot shows a terminal session where a user runs `./test2-5-1`. The output shows the program writing 'FIFO's are fun!' to a pipe and then reading 'FIFO's are fun!' from the FIFO. The bottom screenshot shows a terminal session where a user runs `./test2-5-2`. The output shows the program reading 'FIFO's are fun!' from the FIFO and then writing 'FIFO's are fun!' to the pipe. Both screenshots show the terminal window with a menu bar (Terminal, Sessions, View, X server, Tools, Games, Settings, Macros, Help) and a toolbar (Quick connect..., 1. Home, 2. centos, 3. centos). The bottom screenshot also shows a toolbar with icons for Re-attach, Fullscreen, Stay on top, Duplicate, and Hide toolbar.

```
[root@azuse 02]# ./test2-5-1
test2-5-1 write FIFO's are fun! to pipe
test2-5-1 read 'FIFO's are fun!' from the FIFO
[root@azuse 02]#
```

```
[root@azuse 02]# ./test2-5-2
test2-5-2 read 'FIFO's are fun!' from the FIFO
test2-5-2 write FIFO's are fun! to pipe
[root@azuse 02]#
```

有名管道传递数据的类型？长度是否有限制？和无名管道比有否区别？

有名管道传递的数据没有类型，内存里是什么就传递什么。长度有限制，根据系统和32位/64位管道buffer最大长度有所不同。和无名管道相比又有名管道

3 信号方式

test3-1-1/2 截获信号并用自定义函数处理

在test3-1-2中重定义信号SIGRTMIN，并打印信息后继续运行；重定义信号SIGRTMIN+1，打印信息后退出。在test3-1-1用shell方式获得test3-1-2的进程号，之后用kill向test3-1-2发送SIGRTMIN和SIGRTMIN+1。

```
//test3-1-1.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/time.h>
#include <string.h>
#include <signal.h>

int main()
{
    FILE *fp = popen("ps -e | grep 'test3-1-2' | awk '{print $1}\\'", "r");
    //打开管道，执行shell 命令
    char buffer[10] = {0};
    while (NULL != fgets(buffer, 10, fp))
    {
        printf("test3-1-2 pid: %s\n", buffer);
    }
    pclose(fp);

    int pid = atoi(buffer);
    printf("send signal SIGMIN\n");
    kill(pid, SIGRTMIN);
    sleep(1);
    printf("send signal SIGMIN+1\n");
    kill(pid, SIGRTMIN+1);

    return 0;
}
```

```
//test3-1-2.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/time.h>
#include <string.h>
#include <signal.h>

void static sig_pause(int signal)
{
    printf("signal captured : %d\n", signal);
    printf("keep running... \n");
}
```



```
}

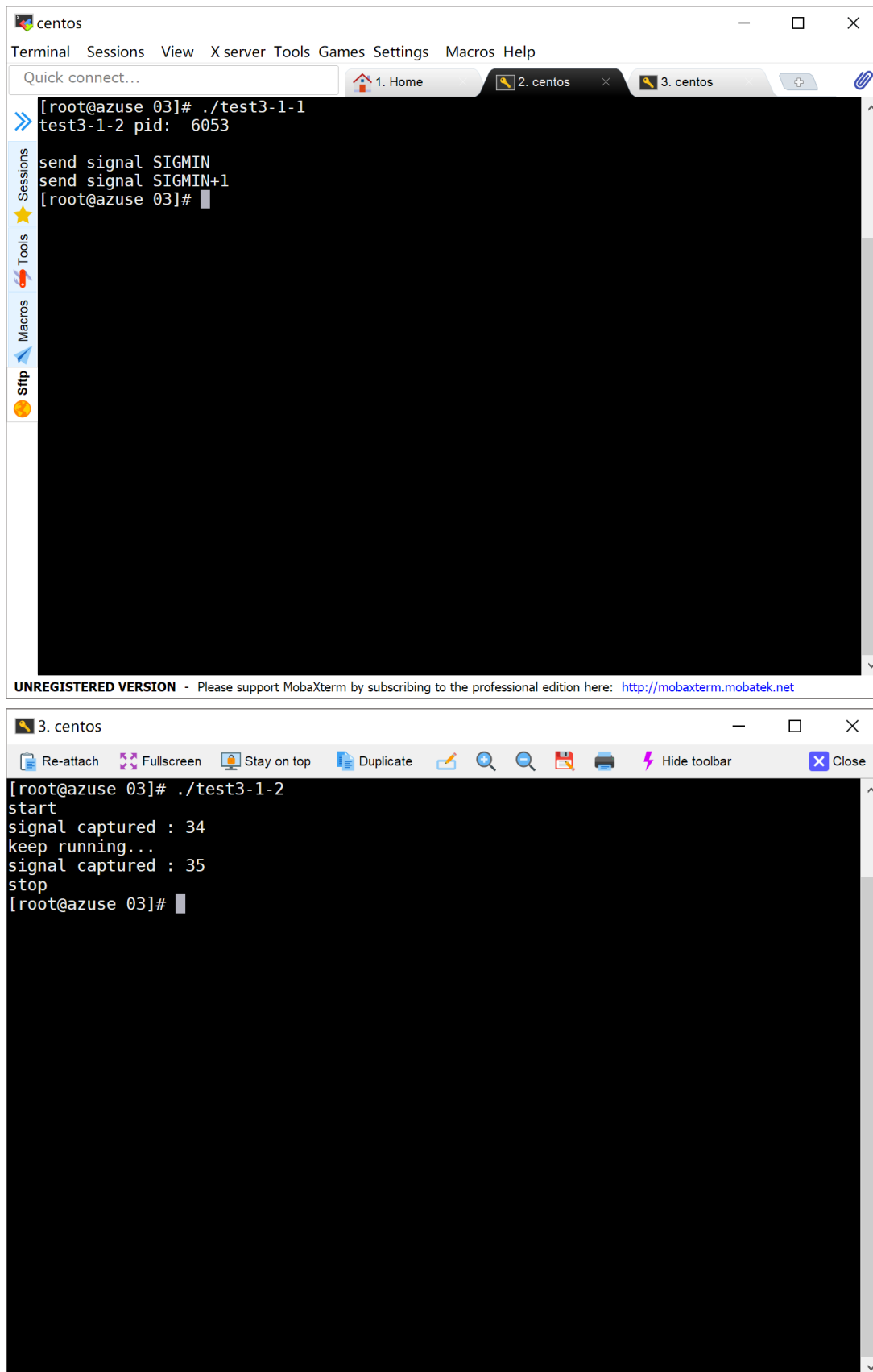
void static sig_stop(int signal)
{
    printf("signal captured : %d\n", signal);
    printf("stop\n");
    exit(0);
}

int main(){
    signal(SIGRTMIN, sig_pause);
    signal(SIGRTMIN+1, sig_stop);

    printf("start\n");
    while(1)
        sleep(1);

    return 0;
}
```

效果：



The image shows two screenshots of a MobaXterm terminal window. The top screenshot shows a terminal session on a CentOS system. The user runs `./test3-1-1`, which outputs `test3-1-2 pid: 6053`. The user then enters `send signal SIGMIN` and `send signal SIGMIN+1`. The bottom screenshot shows the same terminal session. The user runs `./test3-1-2`, which outputs `start`, `signal captured : 34`, `keep running...`, `signal captured : 35`, and `stop`. The terminal window has a menu bar with options like Terminal, Sessions, View, X server, Tools, Games, Settings, Macros, and Help. The bottom of the window displays a message: "UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <http://mobaxterm.mobatek.net>".

```
[root@azuse 03]# ./test3-1-1
test3-1-2 pid: 6053

send signal SIGMIN
send signal SIGMIN+1
[root@azuse 03]#
```

```
[root@azuse 03]# ./test3-1-2
start
signal captured : 34
keep running...
signal captured : 35
stop
[root@azuse 03]#
```

使用终端手动发送信号

先查询test3-1-2的pid，的在终端中输入`kill -34 pid`，之后输入`kill -35 pid`

效果：

The first screenshot shows a terminal window titled 'centos' with a menu bar (Terminal, Sessions, View, X server, Tools, Games, Settings, Macros, Help) and a 'Quick connect...' field. The terminal output shows the command `ps -ef|grep test3-1-2` being executed, displaying two lines of process information. Subsequent commands `kill -34 6105` and `kill -35 6105` are shown. The second screenshot shows the same terminal window with the command `./test3-1-2` executed, resulting in the output: `start`, `signal captured : 34`, `keep running...`, `signal captured : 35`, and `stop`.

```
[root@azuse 03]# ps -ef|grep test3-1-2
root      6105   5455    0 10:08 pts/1    00:00:00 ./test3-1-2
root      6109   5258    0 10:08 pts/0    00:00:00 grep --color=auto test3-1-2
[root@azuse 03]# kill -34 6105
[root@azuse 03]# kill -35 6105
[root@azuse 03]#
```

```
[root@azuse 03]# ./test3-1-2
start
signal captured : 34
keep running...
signal captured : 35
stop
[root@azuse 03]#
```

信号能否带数据?

不能

哪几个信号不能被截获和重定义

SIGSTOP(19)和SIGKILL(9)

写测试程序，用一种方式证明进程收到了kill -9

某一个进程收到SIGKILL之后系统会立刻终止这个进程的运行并且回收进程，所以用这个进程中的代码来判断它收到了kill -9是不行的，只能通过父子进程的方法，在子进程退出时，父进程截获子进程退出的信号。

test3-2-1/2 两个程序轮流读写文件并互相用信号通知对方：

使用ftruncate清空文件，使用lseek重置fd的指针，fclose保存。在写打开，写入，保存，发信号通知对方。

```
//test3-2-1.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/time.h>
#include <string.h>
#include <signal.h>
FILE *fd;
int pid = 1;

void static sigmin(int signal)
{
    printf("signal recived\n");

    fd = fopen("temp", "r");

    char buffer[256] = {0};
    fread(buffer, 256, 1, fd);
    printf("read from file: %s\n", buffer);

    ftruncate(fd, 0);
    lseek(fd, 0, SEEK_SET);
    fclose(fd);

    fd = fopen("temp", "w");

    strcpy(buffer, "this is 3-2-1, over");
    fwrite(buffer, 256, 1, fd);
    printf("write to file: %s\n", buffer);

    fclose(fd);
    printf("send signal SIGMIN to pid %d\n\n\n", pid);
    kill(pid, SIGRTMIN);
}

int main()
{
    while (1)
    {
        FILE *fp = popen("ps -e | grep \'test3-2-2\' | awk \'{print $1}\'", "r"); //打开管道，执行shell 命令
```

```

        char buffer[10] = {0};
        while (NULL != fgets(buffer, 10, fp))
        {
            printf("test3-2-2 pid: %s\n", buffer);
        }
        pclose(fp);
        pid = atoi(buffer);
        if (pid != 0)
            break;
    }

    signal(SIGRTMIN, sigmin);

    fd = fopen("temp", "wr");
    char buffer[256];
    strcpy(buffer, "this is 3-2-1, over");
    fwrite(buffer, 256, 1, fd);
    printf("send signal SIGMIN to pid %d\n", pid);

    fclose(fd);

    kill(pid, SIGRTMIN);

    while (1)
        sleep(1);

    return 0;
}

```

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/time.h>
#include <string.h>
#include <signal.h>
FILE *fd;
int pid = 1;

void static sigmin(int signal)
{
    printf("signal recived\n");

    fd = fopen("temp", "r");

    char buffer[256] = {0};
    fread(buffer, 256, 1, fd);
    printf("read from file: %s\n", buffer);

    ftruncate(fd, 0);
}

```

```

        lseek(fd,0,SEEK_SET);
        fclose(fd);

        fd = fopen("temp", "w");

        strcpy(buffer, "this is 3-2-2, over");
        fwrite(buffer, 256, 1, fd);
        printf("write to file: %s\n", buffer);

        fclose(fd);
        printf("send signal SIGMIN to pid %d\n\n\n", pid);
        kill(pid, SIGRTMIN);
    }

    int main()
    {
        while (1)
        {
            FILE *fp = popen("ps -e | grep 'test3-2-1' | awk '{print $1}'", "r"); //打开管道, 执行shell 命令
            char buffer[10] = {0};
            while (NULL != fgets(buffer, 10, fp))
            {
                printf("test3-2-1 pid: %s\n", buffer);
            }
            pclose(fp);
            pid = atoi(buffer);
            if(pid != 0)
                break;
        }

        signal(SIGRTMIN, sigmin);

        printf("start\n");
        while (1)
            sleep(1);

        return 0;
    }

```

消息队列方式

test4-1-1/2从test4-1-向test4-1-2单向传递数据

使用msgget来创建消息队列，msgsnd来向消息队列写数据，test4-1-2中msgget获得消息队列，msgrcv读取数据，msgctl删除消息队列。使用iqcs -q查看消息队列情况。

```

//test4-1-1.c
#include <unistd.h>
#include <stdlib.h>

```

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/msg.h>

struct msgbuf {
    long mtype;
    char mtext[255];
};

int main() {
    int msg_id = msgget(411, IPC_CREAT | 0666);
    if (msg_id != -1) {
        struct msgbuf mybuf;
        mybuf.mtype = 1;
        strcpy(mybuf.mtext, "I'm send process.\n");

        if (msgsnd(msg_id, &mybuf, sizeof(mybuf.mtext), 0))
            printf("success\n");
        else
            perror("msgsnd:");
    } else {
        perror("msgget:");
    }

    return 0;
}
```

```
//test4-1-2.c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

struct msgbuf {
    long mtype;
    char mtext[255];
};

int main() {
    int msg_id = msgget(411, IPC_CREAT | 0666);
    if (msg_id != -1) {
        struct msgbuf mybuf;
        if (msgrcv(msg_id, &mybuf, sizeof(mybuf.mtext), 0, IPC_NOWAIT) != -1) {
            printf("read success: %s\n", mybuf.mtext);
            if (msgctl(msg_id, IPC_RMID, 0) != -1)
                printf("delete msg success\n");
        } else {
            perror("msgsnd:");
        }
    }
}
```

```

    }

    } else {
        perror("msgget:");
    }

    return 0;
}

```

效果：

```

centos
Terminal Sessions View X server Tools Games Settings Macros Help
Quick connect... 1. Home 2. centos 3. centos
[root@azuse 04]# ./test4-1-1
msgsnd:: Success
[root@azuse 04]# ipcs -q
----- 消息队列 -----
键      msqid    拥有者    权限      已用字节数    消息
0x0000019b 32768      root      666        255           1
[root@azuse 04]# ./test4-1-2
read success: I'm send process.
delete msg success
[root@azuse 04]#
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net

```

test4-2-1/2 一对程序间双向传递数据

可以在传递的msg中的mtype设置一个传递的数据类型int，然后再msgrcv中只读出指定类型的消息。然后test2-4-1写类型1，读类型2；test2-4-2写类型2，读类型1，就可以实现双向读写了。

```

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/msg.h>

struct msgbuf
{
    long mtype;
    char mtext[255];
};

```



```

int main()
{
    struct msgbuf mybuf;

    int snd_id = msgget(411, IPC_CREAT | 0666);
    if (snd_id != -1)
    {
        mybuf.mtype = 1;
        strcpy(mybuf.mtext, "I'm test2-4-1.\n");
        int sndret = msgsnd(snd_id, &mybuf, sizeof(mybuf.mtext), 0);
        if (sndret == 0)
            printf("test2-4-1 msgsnd success, msgsnd return %d\n", sndret);
        else
            perror("msgsnd:");
    }
    else
    {
        perror("msgget:");
    }

    int rcv_id = msgget(412, IPC_CREAT | 0666);
    if (msgrcv(rcv_id, &mybuf, sizeof(mybuf.mtext), 2, 0) != -1)
    {
        printf("test2-4-1 read success: %s\n", mybuf.mtext);
        if (msgctl(rcv_id, IPC_RMID, 0) != -1)
            printf("test2-4-1 delete msg success\n");
    }
    else
    {
        perror("test2-4-1 read error:");
    }
    if (msgctl(rcv_id, IPC_RMID, 0) != -1)
        printf("delete msg success\n");
    return 0;
}

```

```

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

struct msgbuf
{
    long mtype;
    char mtext[255];
};

```

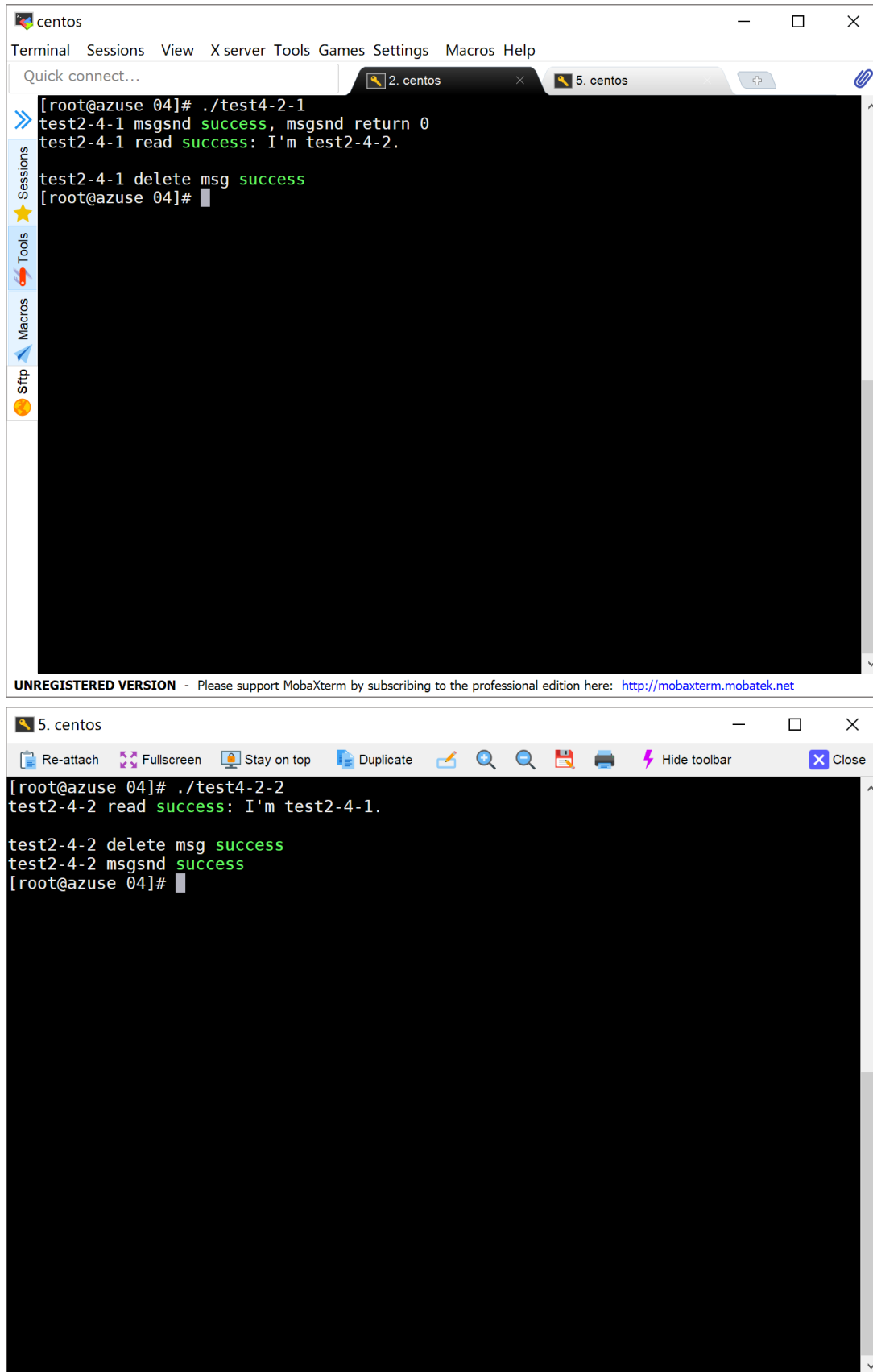
```
int main()
{
    struct msgbuf mybuf;

    int rcv_id = msgget(411, IPC_CREAT | 0666);
    if (msgrcv(rcv_id, &mybuf, sizeof(mybuf.mtext), 1, 0) != -1)
    {
        printf("test2-4-2 read success: %s\n", mybuf.mtext);
        if (msgctl(rcv_id, IPC_RMID, 0) != -1)
            printf("test2-4-2 delete msg success\n");
    }
    else
    {
        perror("test2-4-2 msgcv error:");
    }
    if (msgctl(rcv_id, IPC_RMID, 0) != -1)
        printf("delete msg success\n");

    struct msgbuf sndbuf;
    int snd_id = msgget(412, IPC_CREAT | 0666);
    if (snd_id != -1)
    {
        sndbuf.mtype = 2;
        strcpy(sndbuf.mtext, "I'm test2-4-2.\n");
        int sndret = msgsnd(snd_id, &sndbuf, sizeof(sndbuf.mtext), 0);
        if (sndret == 0)
            printf("test2-4-2 msgsnd success\n");
        else
        {
            perror("test2-4-2 msgsnd error:");
        }
    }
    else
    {
        perror("test2-4-2 msgget error:");
    }

    return 0;
}
```

效果：



The image shows two screenshots of a MobaXterm terminal window. The top screenshot shows a terminal session on a CentOS system. The user is running a program called test4-2-1. The output shows that test2-4-1 successfully sent a message (msgsnd) and then read it (read), displaying "I'm test2-4-2." The user then deletes the message (delete msg) successfully. The bottom screenshot shows a terminal session on a CentOS system. The user is running a program called test4-2-2. The output shows that test2-4-2 successfully read a message (read), displaying "I'm test2-4-1." The user then deletes the message (delete msg) successfully and sends a message (msgsnd) successfully.

```
[root@azuse 04]# ./test4-2-1
test2-4-1 msgsnd success, msgsnd return 0
test2-4-1 read success: I'm test2-4-2.
test2-4-1 delete msg success
[root@azuse 04]#
```

```
[root@azuse 04]# ./test4-2-2
test2-4-2 read success: I'm test2-4-1.
test2-4-2 delete msg success
test2-4-2 msgsnd success
[root@azuse 04]#
```

消息队列传递的数据类型，长度是否有限制？和有名/无名管道的区别？

消息队列传递的数据类型为char，长度限制为16380字节，和有名无名管道的区别在于消息队列可以自定义msgtype，实现全双工，并且消息队列保存在系统中，程序退出后如果没有销毁消息队列，消息队列会留在队列里。

共享内存方式

test5-1/2 共享一段内存，test5-1向这段内存写入数据后，test5-2读到内容

使用shmget获得一个共享内存空间并且设置key，通过shmat获取指定key的内存地址，然后就可以用这个地址读写数据了。

```
#include <stdlib.h>
#include <stdio.h>
#include <semaphore.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <sys/prctl.h>
#include <fcntl.h>
#include <stdarg.h>

int main()
{
    char mem[1024];
    int shmem_id;
    char *shmaddr;
    shmem_id = shmget(51, sizeof(mem), IPC_CREAT | 0600);
    shmaddr = (char *)shmat(shmem_id, NULL, 0);
    strcpy(shmaddr, "This is test5-1 speaking, out...");
    printf("msg send to shared memory\n");

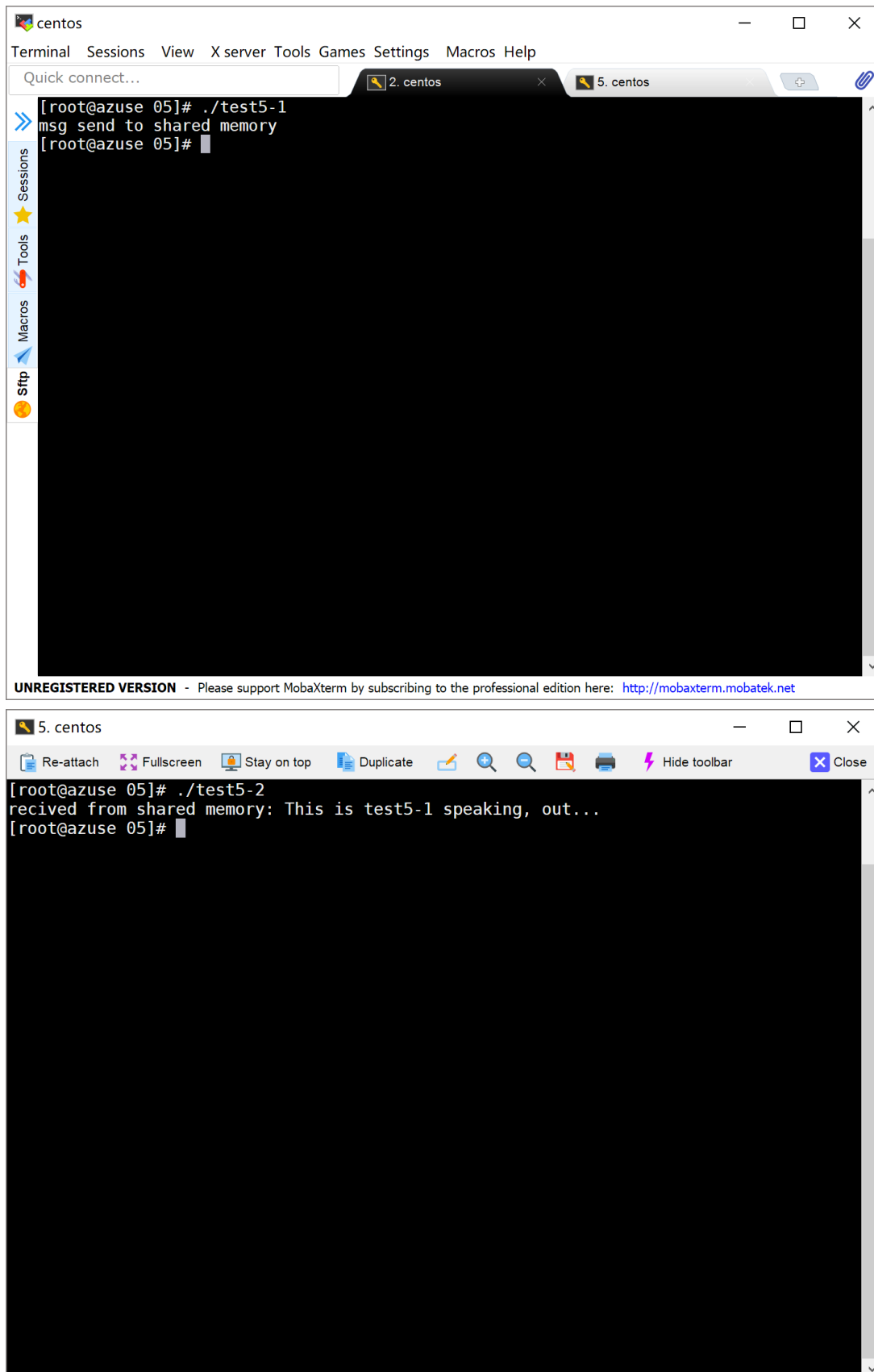
    return 0;
}
```

```
#include <stdlib.h>
#include <stdio.h>
#include <semaphore.h>
#include <sys/time.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/prctl.h>
#include <fcntl.h>
#include <stdarg.h>
#include <sys/shm.h>

int main()
{
    char mem[1024];
    int shmem_id;
    char *shmaddr;
    shmem_id = shmget(51, sizeof(mem), IPC_CREAT | 0600);
    shmaddr = (char *)shmat(shmem_id, NULL, 0);
    printf("recived from shared memory: %s\n", shmaddr);
}
```

```
    return 0;  
}
```

效果：



能否在父子进程间共享内存？还是可以在独立进程间共享？

都可以，shmget设置IPC_PRIVATE可以设置别的独立进程看不到，只有父子进程看得到。

如果两个进程同时写，共享内存是否会乱？如何防止内存内容乱？

会，用一个信号量加上一把写锁，不让两个进程同时写内存。

UNIX套接字方式

test6-1-1/2 建立UNIX类型的socket，双向通信

地址使用sockaddr_un数据结构，其中un_path设置socket文件地址，'\0'可以使用抽象文件名，sun_family设置为AF_UNIX，建立的socket类型也设置为AF_UNIX，accept是第二个第三个实参为NULL，其他都和socket一样。

```
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <sys/select.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <sys/time.h>
#include <signal.h>

struct sockaddr_un
{
    sa_family_t sun_family;
    char sun_path[108];
};

int main()
{
    int server_fd, valread;
    struct sockaddr_un address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[100] = {0};
    char buffer_send[10] = {0};

    // Creating socket file descriptor
    if ((server_fd = socket(AF_UNIX, SOCK_STREAM, 0)) == 0)
    {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    address.sun_family = AF_UNIX;
    strcpy(address.sun_path, "temp");
```

```
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0)
{
    perror("bind failed");
    exit(EXIT_FAILURE);
}
else
{
    printf("bind success\n");
}

if (listen(server_fd, 3) < 0)
{
    perror("listen failed");
    exit(EXIT_FAILURE);
}
else
{
    printf("listening\n");
}

int new_socket = accept(server_fd, NULL, NULL);
if (new_socket == -1)
{
    perror("accept failed");
    exit(-1);
}
printf("start reading...\n");

int read = recv(new_socket, buffer, 100, 0);
if (read == 0)
{
    perror("Connect closed");
}
else if (read < 0)
{
    printf("recv return : %d\n", read);
    perror("");
}
else
{
    printf("%d bytes read, msg is: \n", read);
    printf("%s\n", buffer);
}

unlink(address.sun_path);
printf("socket over\n");

return 0;
}
```

```
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <sys/select.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/time.h>
#include <string.h>
#include <signal.h>

struct sockaddr_un {
    sa_family_t sun_family;
    char sun_path[108];
};

int main()
{
    struct sockaddr_un address;
    int sock = 0, valread;
    struct sockaddr_un serv_addr;

    char *hello = "This is test6-1-2, Come in, Over";
    char buffer[100] = {0};
    char buffer_send[15] = {0};

    if ((sock = socket(AF_UNIX, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        return -1;
    }

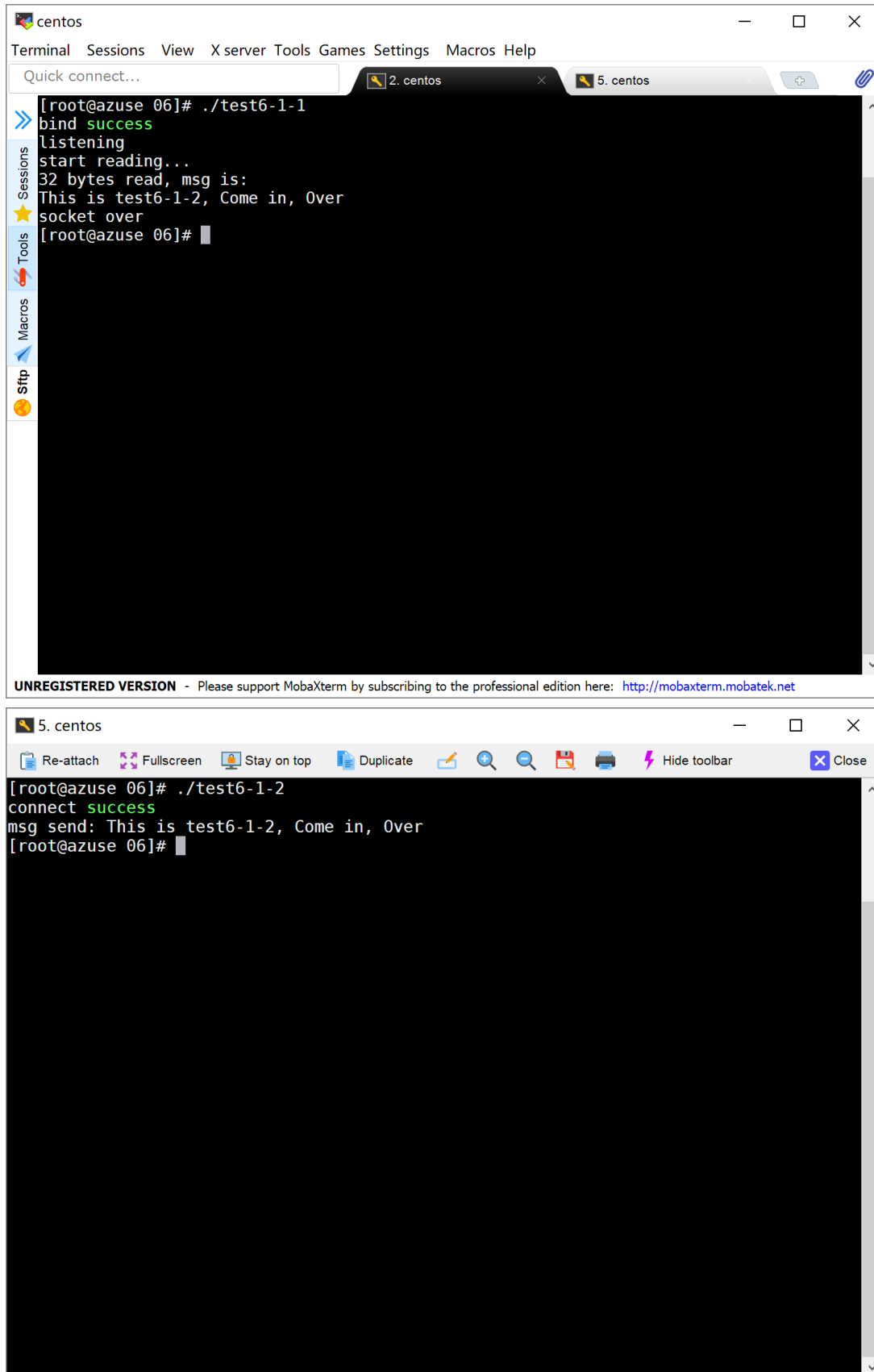
    serv_addr.sun_family = AF_UNIX;
    strcpy(serv_addr.sun_path, "temp");

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        printf("\nConnection Failed \n");
        return -1;
    }
    else
    {
        printf("connect success\n");
    }

    send(sock , hello , strlen(hello) , 0);
    printf("msg send: %s\n", hello);

    return 0;
}
```


效果：



The image shows two screenshots of a MobaXterm terminal window. The top screenshot shows a process running `./test6-1-1` which successfully binds to a port, starts listening, and receives a message from another process: "This is test6-1-2, Come in, Over". The bottom screenshot shows a process running `./test6-1-2` which successfully connects to the first process and sends the message "This is test6-1-2, Come in, Over".

```
[root@azuse 06]# ./test6-1-1
bind success
listening
start reading...
32 bytes read, msg is:
This is test6-1-2, Come in, Over
socket over
[root@azuse 06]#
```

```
[root@azuse 06]# ./test6-1-2
connect success
msg send: This is test6-1-2, Come in, Over
[root@azuse 06]#
```

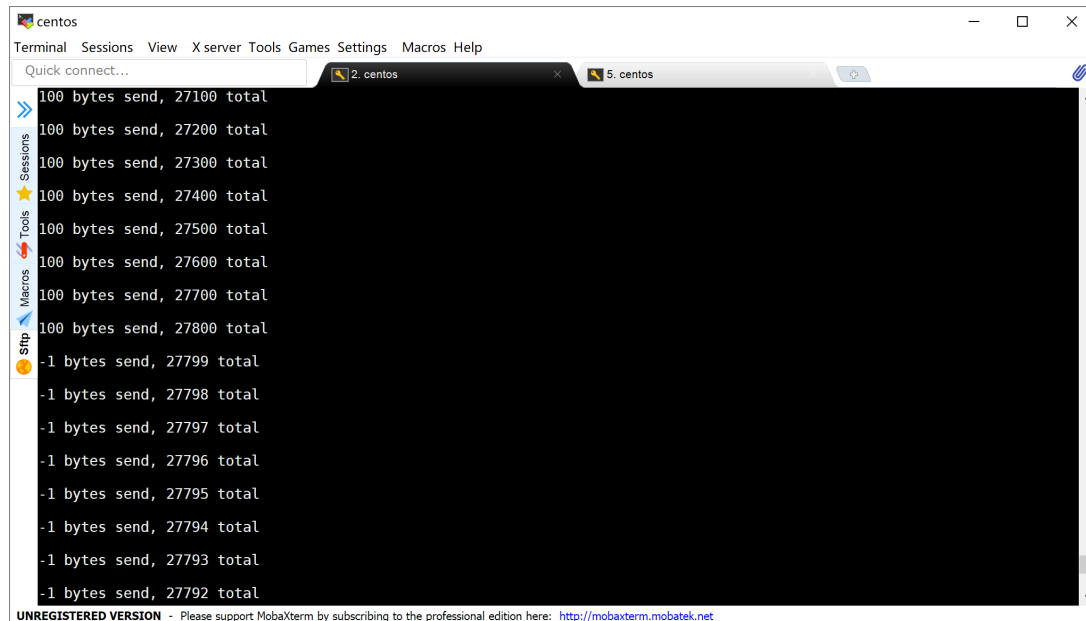
UNIX类型socket的建立/使用和TCP socket相比有什么相同和不同点？

所有socket类型是AF_INT的属性都要改成AF_UNIX。

addr要使用sockaddr_un数据类型，设置un_path文件路径替代TCP socket使用ip地址。

UNIX类型的socket是否有阻塞和非阻塞方式？能否通过select来读写？写满后返回不可写还是导致数据丢失？

有阻塞和非阻塞，可以通过select来读写，写满后返回-1。



The screenshot shows a terminal window with a menu bar (Terminal, Sessions, View, X server, Tools, Games, Settings, Macros, Help) and a toolbar. The terminal output is as follows:

```
>> 100 bytes send, 27100 total
100 bytes send, 27200 total
100 bytes send, 27300 total
100 bytes send, 27400 total
100 bytes send, 27500 total
100 bytes send, 27600 total
100 bytes send, 27700 total
100 bytes send, 27800 total
-1 bytes send, 27799 total
-1 bytes send, 27798 total
-1 bytes send, 27797 total
-1 bytes send, 27796 total
-1 bytes send, 27795 total
-1 bytes send, 27794 total
-1 bytes send, 27793 total
-1 bytes send, 27792 total
```

At the bottom, a message reads: "UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <http://mobaxterm.mobatek.net>"

// 修改后的测试程序

```
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <sys/select.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <sys/time.h>
#include <signal.h>

struct sockaddr_un
{
    sa_family_t sun_family;
    char sun_path[108];
};

int main()
{
    int server_fd, valread;
    struct sockaddr_un address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[100] = {0};
    char buffer_send[10] = {0};
```

```
// Creating socket file descriptor
if ((server_fd = socket(AF_UNIX, SOCK_STREAM, 0)) == 0)
{
    perror("socket failed");
    exit(EXIT_FAILURE);
}

address.sun_family = AF_UNIX;
strcpy(address.sun_path, "temp");

if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0)
{
    perror("bind failed");
    exit(EXIT_FAILURE);
}
else
{
    printf("bind success\n");
}

if (listen(server_fd, 3) < 0)
{
    perror("listen failed");
    exit(EXIT_FAILURE);
}
else
{
    printf("listening\n");
}

int new_socket = accept(server_fd, NULL, NULL);
if (new_socket == -1)
{
    perror("accept failed");
    exit(-1);
}
printf("start reading...\n");

int read = recv(new_socket, buffer, 100, 0);
if (read == 0)
{
    perror("Connect closed");
}
else if (read < 0)
{
    printf("recv return : %d\n", read);
    perror("");
}
else
{
    printf("%d bytes read, msg is: \n", read);
    printf("%s\n", buffer);
}
```

```
printf("testing select ...\n");

int flags = fcntl(new_socket, F_GETFL, 0);
fcntl(new_socket, F_SETFL, flags | O_NONBLOCK);
fd_set rfds, wfds;
struct timeval tv;

FD_ZERO(&rfds);
FD_ZERO(&wfds);
FD_SET(new_socket, &rfds);
// FD_SET(sock, &wfds);
/* set select() time out */
tv.tv_sec = 10;
tv.tv_usec = 0;
int selres = select(new_socket + 1, &rfds, NULL, NULL, &tv);
switch (selres)
{
case -1:
    perror("select error: ");
    break;
case 0:
    printf("select time out\n");
    break;
default:
    printf("select return %d\n", selres);
    int read = recv(new_socket, buffer, 100, 0);
    if (read == 0)
    {
        perror("Connect closed");
    }
    else if (read < 0)
    {
        printf("recv return : %d\n", read);
        perror("");
    }
    else
    {
        printf("%d bytes read, msg is: \n", read);
        printf("%s\n", buffer);
    }
}

// flags = fcntl(new_socket, F_GETFL, 0);
// fcntl(new_socket, F_SETFL, flags & ~O_NONBLOCK);
printf("testing max send bytes ...\n");
int sum = 0;
while(1)
{
    int sndb = send(new_socket, buffer, 100, 0);
    sum += sndb;
    printf("%d bytes send, %d total\n", sndb, sum);
    getchar();
}
```

```
        unlink(address.sun_path);  
        printf("socket over\n");  
  
        return 0;  
    }
```

文件锁机制

test7-1-1/2 7-1-1打开某个特定的文件之后加锁，延时一段时间之后写入一些内容，释放锁，同时启动7-1-2当锁释放后，7-1-2可以退出阻塞

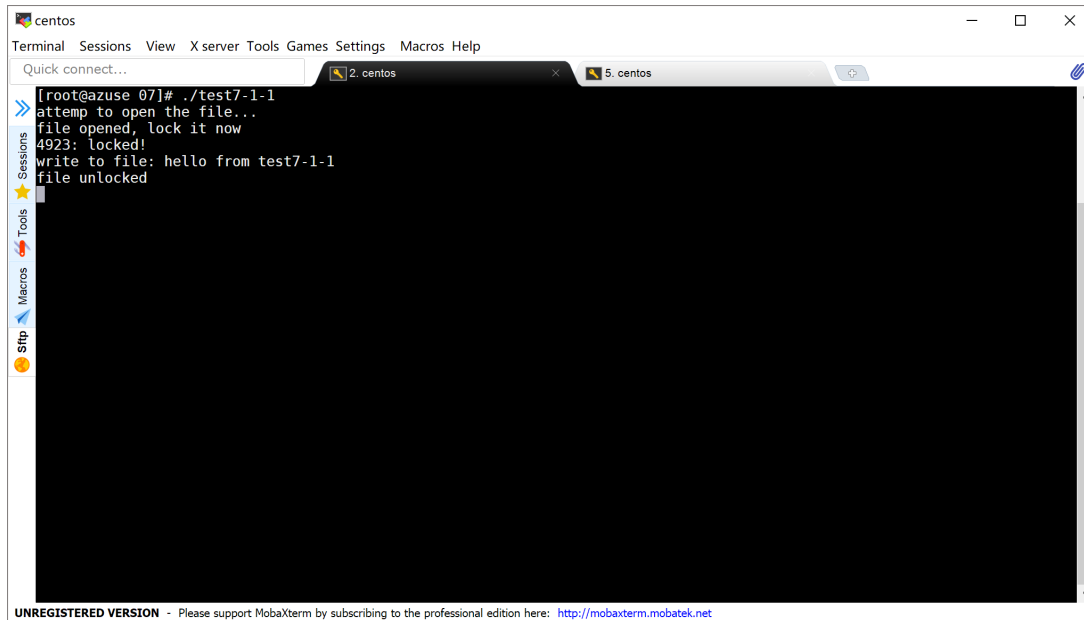
使用open/write/read函数操作文件，使用flock(fd, LOCK_EX)加锁，flock(fd, LOCK_UN)解锁。

```
//test7-1-1.c  
#include <stdlib.h>  
#include <stdio.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <unistd.h>  
#include <sys/file.h>  
#include <wait.h>  
  
int main()  
{  
    int fd;  
    printf("attemp to open the file...\n");  
  
    fd = open("temp", O_RDWR|O_CREAT|O_TRUNC, 0644);  
    if (fd < 0) {  
        perror("open() error: ");  
        exit(1);  
    }  
  
    printf("file opened, lock it now \n");  
    if (flock(fd, LOCK_EX) < 0) {  
        perror("flock() error: ");  
        exit(1);  
    }  
    printf("file locked!\n");  
  
    sleep(10);  
  
    char buff[] = "hello from test7-1-1";  
    write(fd, buff, sizeof(buff) + 1);  
    printf("write to file: %s\n", buff);  
  
    if (flock(fd, LOCK_UN) < 0) {  
        perror("flock() error: ");  
        exit(1);  
    }  
}
```

```
    }  
    printf("file unlocked\n");  
  
    while(1)  
        sleep(1);  
    //unlink("temp");  
    exit(0);  
}
```

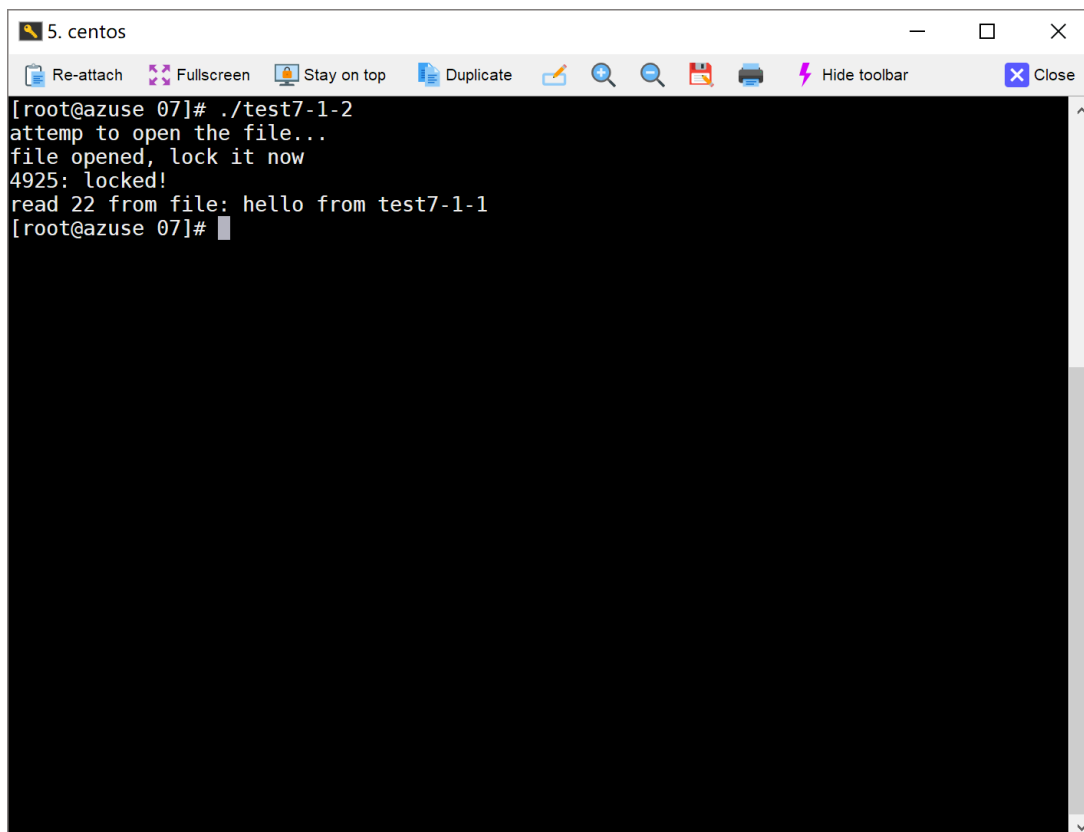
```
//test7-1-2.c  
#include <stdlib.h>  
#include <stdio.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <unistd.h>  
#include <sys/file.h>  
#include <wait.h>  
  
int main()  
{  
    int fd;  
  
    printf("attemp to open the file...\n");  
    fd = open("temp", O_RDWR|O_CREAT|O_TRUNC, 0644);  
    if (fd < 0) {  
        perror("open() error: ");  
        exit(1);  
    }  
  
    printf("file opened, lock it now \n");  
    if (flock(fd, LOCK_EX) < 0) {  
        perror("flock() error: ");  
        exit(1);  
    }  
    printf("file locked!\n");  
  
    char buff[256];  
    int readret = read(fd, buff, sizeof(buff));  
    if(readret >= 0)  
        printf("read %d from file: %s\n", readret, buff);  
    else  
        perror("read error: ");  
  
    unlink("temp");  
    exit(0);  
}
```

效果：



```
[root@azuse 07]# ./test7-1-1
attempt to open the file...
file opened, lock it now
4923: locked!
write to file: hello from test7-1-1
file unlocked
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <http://mobaxterm.mobatek.net>



```
[root@azuse 07]# ./test7-1-2
attempt to open the file...
file opened, lock it now
4925: locked!
read 22 from file: hello from test7-1-1
[root@azuse 07]#
```

test7-2-1/2 打开文件后设置fd为非阻塞模式

在open的flag中加上O_NONBLOCK即可以非阻塞打开。

```
//test7-2-1.c
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/file.h>
#include <wait.h>
```

```
#include <string.h>

int main()
{
    int fd;
    pid_t pid;

    printf("attemp to open the file...\n");
    fd = open("temp", O_RDWR | O_CREAT | O_TRUNC | O_NONBLOCK, 0644);
    if (fd < 0)
    {
        perror("open() error: ");
        exit(1);
    }
    printf("file opened, lock it now \n");
    struct flock lock;
    memset (&lock, 0, sizeof(lock));
    lock.l_type = F_WRLCK;
    if (fcntl (fd, F_SETLK, &lock) < 0)
    {
        perror("lock error: ");
        exit(1);
    }
    printf("file locked!\n");

    sleep(10);

    char buff[] = "hello from test7-2-1";
    write(fd, buff, sizeof(buff) + 1);
    printf("write to file: %s\n", buff);

    lock.l_type = F_UNLCK;
    if (fcntl (fd, F_SETLK, &lock) < 0)
    {
        perror("flock() error: ");
        exit(1);
    }
    printf("file unlocked\n");

    while (1)
        sleep(1);

    //unlink("temp");
    exit(0);
}
```

```
//test7-2-2.c
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
```



```
#include <fcntl.h>
#include <unistd.h>
#include <sys/file.h>
#include <wait.h>
#include <string.h>

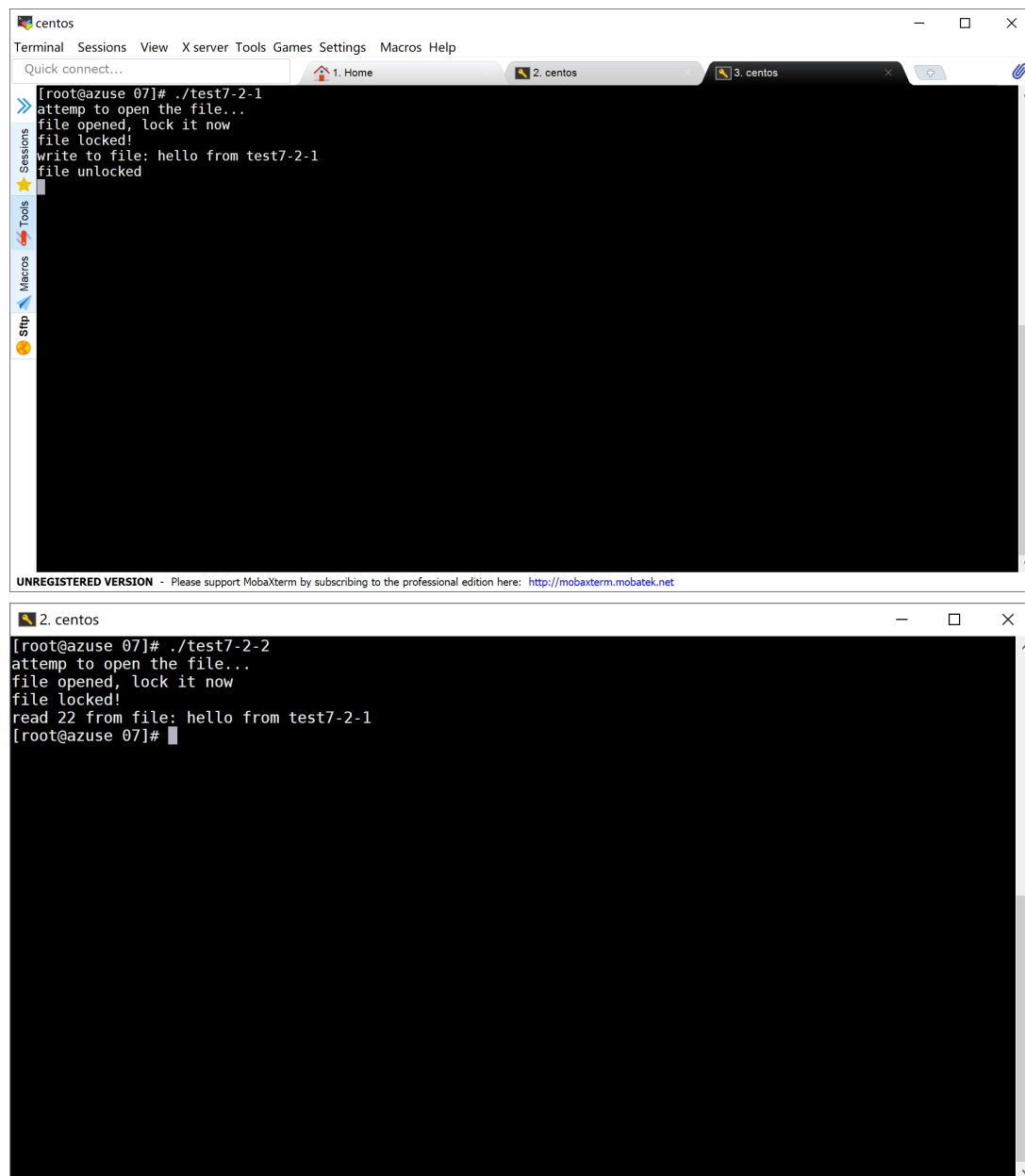
int main()
{
    int fd;
    pid_t pid;
    printf("attemp to open the file...\n");

    fd = open("temp", O_RDWR | O_CREAT | O_TRUNC | O_NONBLOCK, 0644);
    if (fd < 0)
    {
        perror("open() error: ");
        exit(1);
    }
    printf("file opened, lock it now \n");
    struct flock lock;
    memset (&lock, 0, sizeof(lock));
    lock.l_type = F_WRLCK;
    if (fcntl (fd, F_SETLKW, &lock) < 0)
    {
        perror("lock error: ");
        exit(1);
    }
    printf("file locked!\n");

    char buff[256];
    int readret = read(fd, buff, sizeof(buff));
    if (readret >= 0)
        printf("read %d from file: %s\n", readret, buff);
    else
        perror("read error: ");

    unlink("temp");
    exit(0);
}
```

效果:



```
[root@azuse 07]# ./test7-2-1
attemp to open the file...
file opened, lock it now
file locked!
write to file: hello from test7-2-1
file unlocked

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net
```

```
2. centos
[root@azuse 07]# ./test7-2-2
attemp to open the file...
file opened, lock it now
file locked!
read 22 from file: hello from test7-2-1
[root@azuse 07]#
```

锁定文件有几种方法？

有两种方法，flock和fcntl，flock上的是劝告锁，其他程序还是可以写入，fcntl是强制锁，上锁之后其他程序无法写入。

在一个文件加锁之后，另一个文件直接read/write，是阻塞还是失败？

read不阻塞，直接返回0，什么都没有读到

write可以写入，返回buff的size，因为flock加的是劝告锁

在一个文件加锁之后，另一个文件直接read/write（非阻塞方式），是阻塞还是失败？

read直接返回0，什么都没有读到 write可以正常写入。