同济大学计算机网络

实验报告

姓名：涂远鹏-1652262

题目：socket 编程非 TCP 阻塞方式

一.(01 子目录)写一对 TCP socket 测试程序，分为 client 和 server，分别运行在不同的虚拟机上

1.测试程序 tcp_server1-1,端口号通过 main 函数带参数方式传入:

进入等待连接状态:

```
[root@RHEL74-SVR 01]# ./tcp_server1-1 4000
连接成功!
[root@RHEL74-SVR 01]#
```

2.测试程序 tcp_client1-1 运行时带入服务器 IP 地址和端口号:

```
[root@RHEL74-SVR 01]# ./tcp_client1-1 192.168.80.231 4000
连接成功!
[root@RHEL74-SVR 01]#
```

3.Server 端用于 listen 的 socket，不设置为非阻塞方式，accept 成功后将 accept 的 socket 设置为非阻塞:

Listen 的 socket 在 accept 成功后设置为非阻塞：

```
///listen, 成功返回0, 出错返回-1
if(listen(server_sockfd,QUEUE) == -1)
{
    perror("listen");
    exit(1);
}

///客户端套接字
char buffer[BUFFER_SIZE];
struct sockaddr_in client_addr;
socklen_t length = sizeof(client_addr);

///成功返回非负描述字, 出错返回-1
int conn = accept(server_sockfd, (struct sockaddr*)&client_addr
int flags2 = fcntl(conn, F_GETFL, 0);                          //
```

4.client 端建立的 socket,先不设置为非阻塞方式,accept 成功后，将 accept 的 socket 设置为非阻塞:

代码如下,连接前不设置连接后设置为非阻塞:

```
struct sockaddr_in servaddr;
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(port);  ///服务器端口
servaddr.sin_addr.s_addr = inet_addr(argv[1]);  ///服务器ip


///连接服务器, 成功返回0, 错误返回-1
if (connect(sock_cli, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
{
    perror("connect");
    exit(1);
}
else
    cout<<"连接成功!"<<endl;

int flags1 = fcntl(sock_cli, F_GETFL, 0);                          //获取文件的flags1值。
fcntl(sock_cli, F_SETFL, flags1 | O_NONBLOCK);   //设置成非阻塞模式;
```

5.连接成功后,双方均进入 read(recv)状态,read(recv)函数后直接关闭 socket，程序退出:

```
[root@RHEL74-SVR 01]# ./tcp_client1-1 192.168.80.231 4000
连接成功!
[root@RHEL74-SVR 01]#
```

```
[root@RHEL74-SVR 01]# ./tcp_server1-1 4000
连接成功!
[root@RHEL74-SVR 01]#
```

连接成功后直接退出

为了查看双方是否已进入 read/recv 状态，输出返回值为-1:

```
-1[root@RHEL74-SVR 01]# ./tcp_client1-1 192.168.80.231 4000
连接成功!
-1[root@RHEL74-SVR 01]#
```

```
[root@RHEL74-SVR 01]# ./tcp_server1-1 4000
-1连接成功!
```

6.read(recv)函数的表现会如何?程序会阻塞在 read(recv)还是立即结束?read(recv)函数返回什么?

会立即结束，read(recv)函数输出返回值为-1:

```
-1[root@RHEL74-SVR 01]# ./tcp_client1-1 192.168.80.231 4000
连接成功!
-1[root@RHEL74-SVR 01]#
```

```
[root@RHEL74-SVR 01]# ./tcp_server1-1 4000
-1连接成功!
```

7.测试程序 tcp_server1-2/tcp_client1-2 在 1-1 基础上，用 select 使 read(recv)停下来而不立即返回:

测试结果如下,双方不立即返回:

```
-1[root@RHEL74-SVR 01]# ./tcp_client1-2 192.168.80.231 4000
连接成功!
```

```
[root@RHEL74-SVR 01]# ./tcp_server1-2 4000
连接成功!
```

代码如下,用 switch 结构控制 select 的输出:

Tcp_client1-2.cpp:

```
        switch (select(sock_cli + 1, &fdR, NULL,NULL, NULL))
        {
            case -1:
                perror("select");
                break;/* 这说明select函数出错 */
            case 0:
                sleep(1);
                printf("超时\n");
                break; /* 说明在设定的时间内，socket的状态没有发生变化 */
            default:
            memset(recvbuf,0,sizeof(recvbuf));
            flag=recv(sock_cli, recvbuf, sizeof(recvbuf),MSG_DONTWAIT);
        }
```

Tcp_server1-2.cpp:

```
switch (select(conn + 1, &fdR, NULL,NULL, NULL))
{
        case -1:
            perror("select");
            break;/* 这说明select函数出错 */
        case 0:
            sleep(1);
            printf("超时\n");
            break; /* 说明在设定的时间内，socket的状态没有发生变化 */
        default:
        memset(buffer,0,sizeof(buffer));
        flag=recv(conn, buffer, sizeof(buffer),MSG_DONTWAIT);
}
```

8.测试程序 tcp_server1-3 要求 socket 建立后先设置为非阻塞再进行 bind,listen 和 accept,accept 的新的 socket:

Tcp_server1-3 设置如下,先设置非阻塞然后进行 bind 和 listen:

```
int flags1 = fcntl(server_sockfd, F_GETFL, 0);                          //获取文件的flags
fcntl(server_sockfd, F_SETFL, flags1 | O_NONBLOCK);   //设置成非阻塞模式;

///bind, 成功返回0, 出错返回-1
if(bind(server_sockfd,(struct sockaddr *)&server_sockaddr,sizeof(server_sockaddr))==-1)
{
    perror("bind");
    exit(1);
}

///listen, 成功返回0, 出错返回-1
if(listen(server_sockfd,QUEUE) == -1)
{
    perror("listen");
    exit(1);
}
```

9.测试程序 tcp_client1-3 要求 socket 建立成功后,先设置为非阻塞再 connect:

Tcp_client1-3 设置如下:

```
int flags1 = fcntl(sock_cli, F_GETFL, 0);                          //获取文件的flags1值。
fcntl(sock_cli, F_SETFL, flags1 | O_NONBLOCK);   //设置成非阻塞模式;

fd_set  fdR;
FD_ZERO(&fdR);
FD_SET(sock_cli, &fdR);
switch (select(sock_cli + 1, &fdR, NULL,NULL, NULL)) //非阻塞connect
{
    case -1:
        perror("select");
        break;/* 这说明select函数出错 */
    case 0:
        sleep(1);
        printf("超时\n");
        break; /* 说明在设定的时间内，socket的状态没有发生变化 */
    default:
        cout<<connect(sock_cli, (struct sockaddr *)&servaddr, sizeof(servaddr));
}
```

10.要求 tcp_client1-3 能连接 tcp_server1-3 成功后并在连接成功后,用 select 使 read(recv)停下来而不立即返回:

测试结果如下:

```
[root@RHEL74-SVR 01]# ./tcp_client1-3 192.168.80.231 4000
-1连接成功!
```

```
[root@RHEL74-SVR 01]# ./tcp_server1-3 4000
连接成功!
```

二.(02 子目录)写一对 TCP socket 的测试程序，分为 client 和 server，分别运行在不同的虚拟机上

1.测试程序 tcp_server2-1/tcp_client2-1,client 发数据(每次 10 字节,每隔 1 秒),server 用大小 100 的缓冲区收数据，死循环进行:

测试显示如下:

```
[root@RHEL74-SVR 02]# ./tcp_server2-1 4000
连接成功!
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
```

```
[root@RHEL74-SVR 02]# ./tcp_client2-1 192.168.80.231 4000
连接成功!
```

2.此时在 client(server)端 CTRL+C，server(client)端能否检测到连接中断?

Client 端 CTRL+C 观察 server 端结果,server 端没检测到中断:

```
[root@RHEL74-SVR 02]# ./tcp_client2-1 192.168.80.231 4000
连接成功!
^C
```

```
[root@RHEL74-SVR 02]# ./tcp_server2-1 4000
连接成功!
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
```

Server 端 CTRL+C 观察 client 端结果,server 端中断以后 client 端可以检测到中断, 自动退出:

```
[root@RHEL74-SVR 02]# ./tcp_client2-1 192.168.80.231 4000
连接成功!
[root@RHEL74-SVR 02]#
```

```
[root@RHEL74-SVR 02]# ./tcp_server2-1 4000
连接成功!
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
^C
[root@RHEL74-SVR 02]#
```

3.如果新开一个会话窗口,kill -9 杀 client(server)端程序, server(client)端能否检测到连接中断?

新建窗口杀死 client 端，server 端情况如下，检测不到连接已中断:

```
root       2503   1446  0 19:47 ?        00:00:00 sshd: root@pts/1
root       2507   2503  0 19:47 pts/1    00:00:00 -bash
root       2525      1  0 19:47 ?        00:00:00 /usr/sbin/abrt-dbus -t133
root       2552   2340  0 19:48 pts/0    00:00:00 ./tcp_client2-1 192.168.80.231
root       2553   2507  0 19:48 pts/1    00:00:00 ps -ef
[root@RHEL74-SVR homework_second]# kill -9 2552
[root@RHEL74-SVR homework_second]#
```

```
[root@RHEL74-SVR 02]# ./tcp_client2-1 192.168.80.231 4000
连接成功!
已杀死
[root@RHEL74-SVR 02]#
```

```
[root@RHEL74-SVR 02]# ./tcp_server2-1 4000
连接成功!
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
```

新建窗口杀死 server 端，client 端情况如下，可以检测到连接已中断，自动退出：

```
[root@RHEL74-SVR 02]# ./tcp_client2-1 192.168.80.231 4000
连接成功!
[root@RHEL74-SVR 02]#
```

```
[root@RHEL74-SVR 02]# ./tcp_server2-1 4000
连接成功!
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
已终止
[root@RHEL74-SVR 02]#
```

```
root       2282   1665  0 18:43 tty1     00:00:00 -bash
root       2324   1427  0 18:45 ?        00:00:01 sshd: root@pts/0,pts/1
root       2328   2324  0 18:45 pts/0    00:00:00 -bash
root       2423      2  0 19:28 ?        00:00:04 [kworker/0:3]
root       2446      2  0 19:38 ?        00:00:00 [kworker/0:1]
root       2456      2  0 19:43 ?        00:00:00 [kworker/0:0]
root       2459   2324  0 19:47 pts/1    00:00:00 -bash
root       2508   2328  0 19:51 pts/0    00:00:00 ./tcp_server2-1 4000
root       2509   2459  0 19:51 pts/1    00:00:00 ps -ef
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]# kill 2508
[root@RHEL74-SVR ~]#
```

4.测试程序 tcp_server2-2/tcp_client2-2,server 发数据(每次 10 字节，间隔 1 秒)，client 用大小 100 的缓冲区收数据，死循环进行:

首先展示传输数据测试的结果:

```
[root@RHEL74-SVR 02]# ./tcp_client2-2 192.168.80.231 4000
连接成功!
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
```

```
[root@RHEL74-SVR 02]# ./tcp_server2-2 4000
连接成功!
```

Client 用 CTRL+C 中断 server 端可以发现连接中断:

```
[root@RHEL74-SVR 02]# ./tcp_client2-2 192.168.80.231 4000
连接成功!
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
ilcpsklryv
^C
[root@RHEL74-SVR 02]#
```

```
[root@RHEL74-SVR 02]# ./tcp_server2-2 4000
连接成功!
[root@RHEL74-SVR 02]#
```

Server 用 CTRL+C 中断 client 端不能发现连接中断:

```
[root@RHEL74-SVR 02]# ./tcp_client2-2 192.168.80.231 4000
连接成功!
ilcpsklryv
ilcpsklryv
ilcpsklryv
```

```
[root@RHEL74-SVR 02]# ./tcp_server2-2 4000
连接成功!
^C
[root@RHEL74-SVR 02]#
```

三.(03 子目录)写一对 TCP socket 的测试程序，分为 client 和 server，分别运行在不同的虚拟机上

1.测试程序 tcp_server3-1/tcp_client3-1,server 发数据(每次 10 字节,间隔 1 秒)并用大小为 100 的缓冲区收数据, client 发数据(每次 15 字节,间隔 3 秒)并同时用大小为 100 的缓冲区收数据，死循环进行:

由于 sleep 函数信号与 select 函数信号平等, 运行时会出现冲突情况, 所以利用定时器 alarm 函数发送中断给 select 函数实现双方并行，使用 alarm 函数进行定时控制，测试结果如下，任意一端退出另一端在发信号时都能检测到并退出:

Client 端测试结果:

```
[root@RHEL74-SVR 03]# ./tcp_client3-1 192.168.80.231 4000
与服务端连接成功!
recv内容:ilcpsklryv
recv内容:ilcpsklryv
send内容:ilcpsklryvmcpjn
recv内容:ilcpsklryv
recv内容:ilcpsklryv
recv内容:ilcpsklryv
send内容:ilcpsklryvmcpjn
recv内容:ilcpsklryv
recv内容:ilcpsklryv
recv内容:ilcpsklryv
send内容:ilcpsklryvmcpjn
recv内容:ilcpsklryv
^C
```

Server 端测试结果:

```
[root@RHEL74-SVR 03]# ./tcp_server3-1 4000
与客户端连接成功!
send内容:ilcpsklryv
send内容:ilcpsklryv
recv内容:ilcpsklryvmcpjn
send内容:ilcpsklryv
send内容:ilcpsklryv
send内容:ilcpsklryv
recv内容:ilcpsklryvmcpjn
send内容:ilcpsklryv
send内容:ilcpsklryv
send内容:ilcpsklryv
recv内容:ilcpsklryvmcpjn
send内容:ilcpsklryv
send内容:ilcpsklryv
[root@RHEL74-SVR 03]#
```

2.测试程序 tcp_server3-2/tcp_client3-2,server 发数据(每 10 字节，间隔一秒)并用大小为 88 的缓冲区收数据，能否在非阻塞模式下保证每次必须收到 88 字节才返回?

非阻塞情况下无法实现阻塞到一定字节数然后再返回数据，因为一旦设置为非阻塞，server 端的 recv 就算最后一个参数改成 MSG_WAITALL 也会失去作用，运行结果如下,与之前的运行结果一样...:

```
[root@RHEL74-SVR 03]# ./tcp_client3-2 192.168.80.231 4000
与服务端连接成功!
recv内容:ilcpsklryv
recv内容:ilcpsklryv
send内容:ilcpsklryvmcpjn
recv内容:ilcpsklryv
recv内容:ilcpsklryv
recv内容:ilcpsklryv
send内容:ilcpsklryvmcpjn
recv内容:ilcpsklryv
recv内容:ilcpsklryv
recv内容:ilcpsklryv
send内容:ilcpsklryvmcpjn
recv内容:ilcpsklryv
^C
[root@RHEL74-SVR 03]#
```

```
[root@RHEL74-SVR 03]# ./tcp_server3-2 4000
与客户端连接成功!
send内容:ilcpsklryv
send内容:ilcpsklryv
recv如下:ilcpsklryvmcpjn
send内容:ilcpsklryv
send内容:ilcpsklryv
send内容:ilcpsklryv
recv如下:ilcpsklryvmcpjn
send内容:ilcpsklryv
send内容:ilcpsklryv
send内容:ilcpsklryv
recv如下:ilcpsklryvmcpjn
send内容:ilcpsklryv
send内容:ilcpsklryv
[root@RHEL74-SVR 03]#
```

四.(04 子目录)写一对 TCP socket 的测试程序，分为 client 和 server，分别运行在不同的虚拟机上

1.测试程序 tcp_server4-1 接受 client 的连接成功后,用 getchar()进入暂停运行:

测试结果如下，服务端连接成功后等待运行:

```
[root@RHEL74-SVR 04]# ./tcp_server4-1 4000
与客户端连接成功!
```

2.测试程序 tcp_client4-1,用 write 向服务器不断写入，直到 write 失败为止,用 netstat 观察读写队列:

192.168.80.230 × 192.168.80.231

```
发送字节数:444680
发送字节数:444700
发送字节数:444720
发送字节数:444740
发送字节数:444760
发送字节数:444780
发送字节数:444800
发送字节数:444820
发送字节数:444840
发送字节数:444860
发送字节数:444880
发送字节数:444900
发送字节数:444920
发送字节数:444940
发送字节数:444960
发送字节数:444980
发送字节数:445000
发送字节数:445020
发送字节数:445040
发送字节数:445060
发送字节数:445080
发送字节数:445100
发送字节数:445120
发送字节数:445140
发送字节数:445160
发送字节数:445180
```

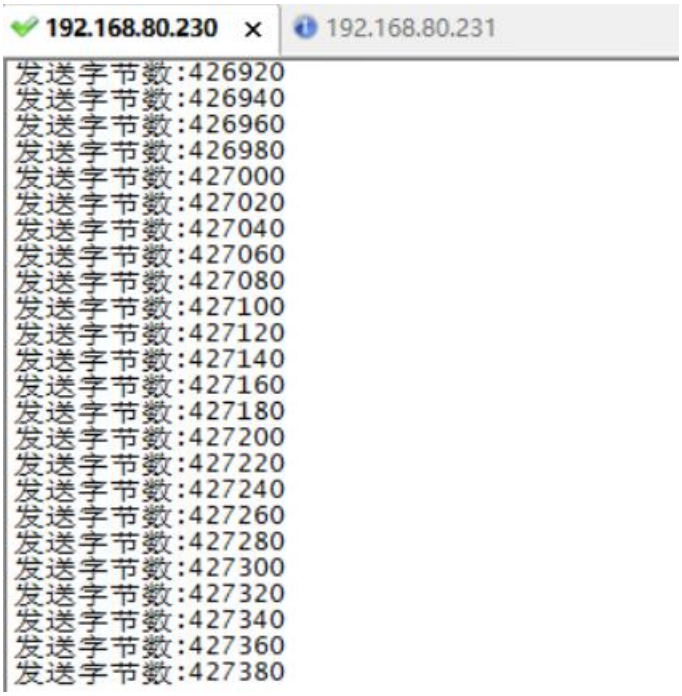
Server 端 netstat -t 结果:

```
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp   215116      0 RHEL74-SVR:terabase     192.168.80.230:35282    ESTABLISHED
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:56270      ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:57147      ESTABLISHED
[root@RHEL74-SVR ~]#
```

Client 端 netstat -t 结果:

```
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:56948      ESTABLISHED
tcp        0 230016 RHEL74-SVR:35282        192.168.80.231:terabase ESTABLISHED
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:56269      ESTABLISHED
[root@RHEL74-SVR ~]#
```

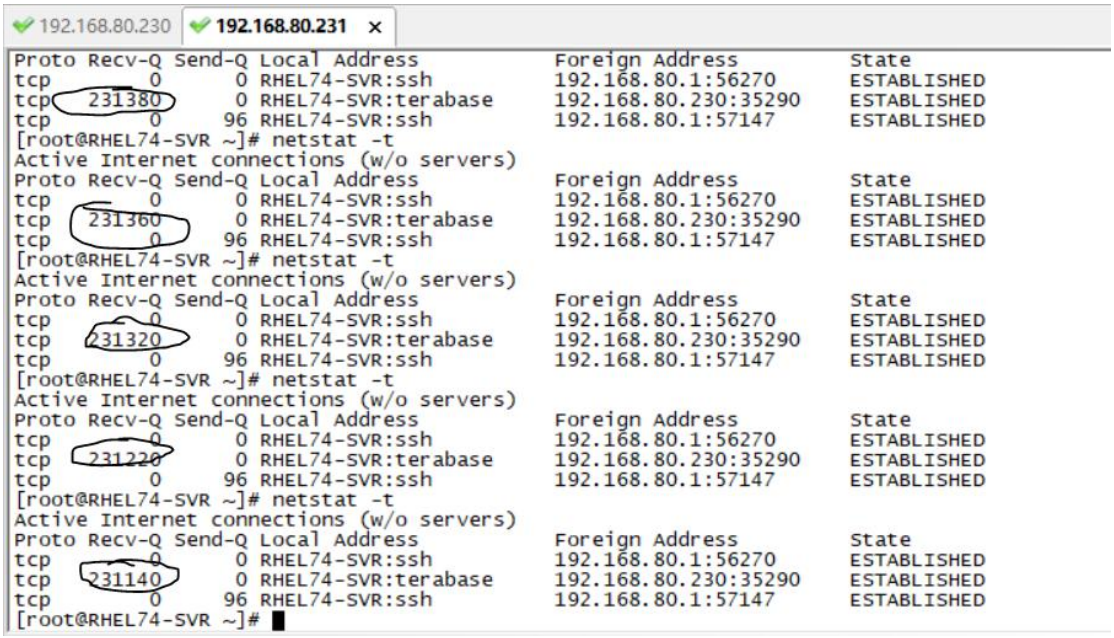
3.测试程序 tcp_server4-2，接受 client 的连接成功后，用每读 20 字节就延时一秒的方法循环读数据:
测试结果如下:



Client 端 netstat -t 结果如下，其 Send -Q 等待 Recv 和 Send 均满了之后就不变了(没有恢复为写状态的原因是因为 server 端读的太慢了，recv -Q 减得也就慢，所以 write 还是不能写):

```
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:56948      ESTABLISHED
tcp        0 195480 RHEL74-SVR:35290        192.168.80.231:terabase ESTABLISHED
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:56269      ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:56948      ESTABLISHED
tcp        0 195480 RHEL74-SVR:35290        192.168.80.231:terabase ESTABLISHED
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:56269      ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
```

而 server 端 netstat -t 结果如下，其 recv -Q 等待 Recv 和 Send 均满了之后由于仍在持续读取 20 字节的数据，所以 Recv 区数据在不断变小:



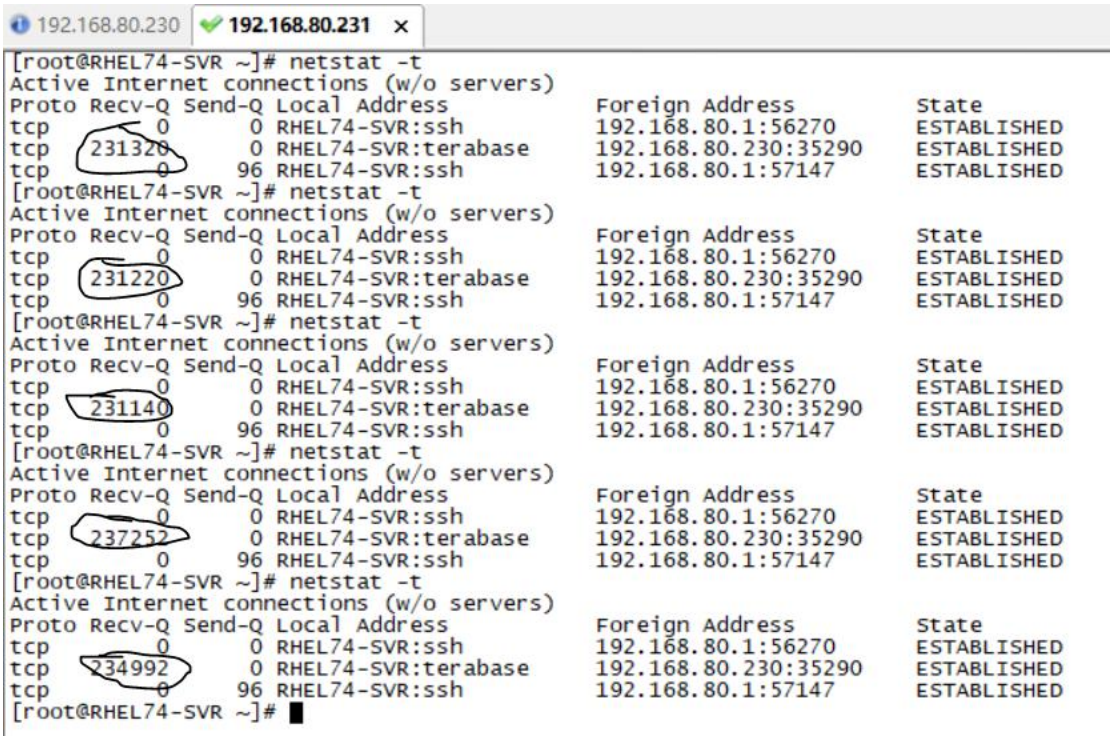
4.测试程序 tcp_client4-2，连接服务端成功后,用 write 向服务端大量写入,直到 write 失败为止,用 netstat 观察读写队列,write 失败后如何恢复为写状态?

答:

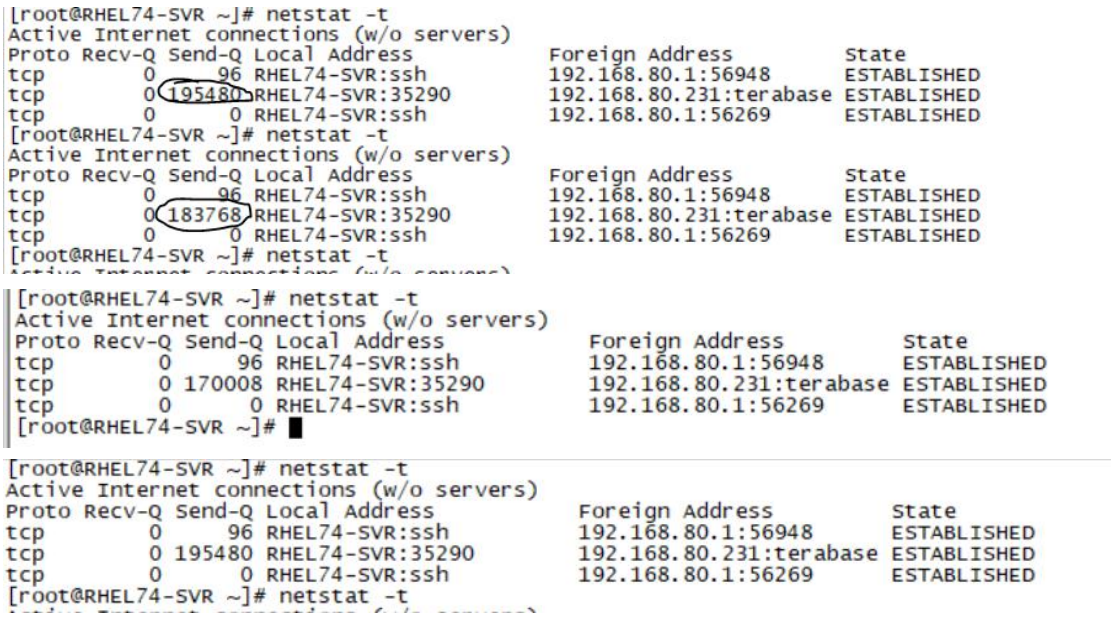
server 端的 Recv-Q 在 Recv -Q 和 Send -Q 均满了之后会出现先减后增的情况，增到一个缓冲区上限左右又开始慢慢减少，在 Recv-Q 减少的过程中 Send-Q 基本不变，Recv-Q 数据量增加时 Send -Q 减少。

解释：Server 的读缓冲区因读取一点点减少，减少到一定量时 client 端 的写缓冲区将一部分数据移至 server 读缓冲区将其填满，此时 client 不会马上执行 write，测试过程发生这样的移动发生几次后 client 的 write 会恢复写入状态，很快又将自己的写缓冲区填满，然后这样的循环重新执行，测试的结果如下:

(1)server 端的 recv -Q 读缓冲区先减后增的情况如下:



(2)client 的写缓冲区 Send -Q 数据先下降后增加情况如下：



五.(05 子目录)写一对 TCP socket 的测试程序，分为 client 和 server，分别运行在不同的虚拟机上

1.测试程序 tcp_server5，接受连接成功后，server 发数据(每次 10 字节，间隔 1 秒)并同时用大小 100 的缓冲区收数据，死循环运行测试程序 tcp_client5-1，连接成功后，client 发数据(每次 15 字节，间隔 3 秒)并同时用大小 100 的缓冲区收数据，死循环运行,server 端先接受一个 client 的连接，进入死循环读写状态,要求此时 server 能接受一个新的 client 端的连接，也进入死循环读写状态:用两个会话窗口分别启动两个 tcp_server5:

Tcp_server5-1 与 tcp_client5-1 测试结果如下:

Server 端:

```
[root@RHEL74-SVR 05]# ./tcp_server5 4000
与客户端连接成功!
客户端send内容:ilcpsklryv
客户端send内容:ilcpsklryv
客户端send内容:ilcpsklryv
与客户端连接成功!
服务端recv内容:ilcpsklryvmcpjn
客户端send内容:ilcpsklryv
客户端send内容:ilcpsklryv
客户端send内容:ilcpsklryv
服务端recv内容:ilcpsklryvmcpjn
服务端recv内容:ilcpsklryvmcpjn
客户端send内容:ilcpsklryv
客户端send内容:ilcpsklryv
客户端send内容:ilcpsklryv
服务端recv内容:ilcpsklryvmcpjn
服务端recv内容:ilcpsklryvmcpjn
客户端send内容:ilcpsklryv
客户端send内容:ilcpsklryv
客户端send内容:ilcpsklryv
服务端recv内容:ilcpsklryvmcpjn
服务端recv内容:ilcpsklryvmcpjn
客户端send内容:ilcpsklryv
客户端send内容:ilcpsklryv
客户端send内容:ilcpsklryv
服务端recv内容:ilcpsklryvmcpjn
服务端recv内容:ilcpsklryvmcpjn
客户端send内容:ilcpsklryv
客户端send内容:ilcpsklryv
```

第一个 client 端连接:

```
192.168.80.230 ×   192.168.80.231   192.168.80.230 (1)
recv内容:ilcpsklryv
^C
[root@RHEL74-SVR 03]# cd ..
[root@RHEL74-SVR 1652262-000109-client]# cd 05
[root@RHEL74-SVR 05]# ./tcp_client5-1 192.168.80.231 4000
与服务端连接成功!
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
服务端send内容:ilcpsklryvmcpjn
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
服务端send内容:ilcpsklryvmcpjn
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
服务端send内容:ilcpsklryvmcpjn
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
服务端send内容:ilcpsklryvmcpjn
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
服务端send内容:ilcpsklryvmcpjn
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
服务端send内容:ilcpsklryvmcpjn
[root@RHEL74-SVR 05]#
```

第二个 client 端连接:

```
[root@RHEL74-SVR 05]# ./tcp_client5-1 192.168.80.231 4000
与服务端连接成功!
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
服务端send内容:ilcpsklryvmcpjn
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
服务端send内容:ilcpsklryvmcpjn
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
服务端send内容:ilcpsklryvmcpjn
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
服务端send内容:ilcpsklryvmcpjn
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
客户端recv内容:ilcpsklryv
服务端send内容:ilcpsklryvmcpjn
[root@RHEL74-SVR 05]# █
```

2.测试程序 tcp_client5-2,运行时带两个端口，建立两个 socket,分别连接两个 server 端，client 发数据(每次 15 字节，每隔 3 秒)并用大小 100 缓冲区收数据，死循环进行：

Tcp_server5-2 与 tcp_client5-2 测试结果如下：

第一个 Server 端，端口为 4000：

```
[root@RHEL74-SVR 05]# ./tcp_server5 4000
与客户端连接成功!
send内容:ilcpsklryv
send内容:ilcpsklryv
send内容:ilcpsklryv
recv内容:ilcpsklryvmcpjn
send内容:ilcpsklryv
send内容:ilcpsklryv
send内容:ilcpsklryv
recv内容:ilcpsklryvmcpjn
send内容:ilcpsklryv
send内容:ilcpsklryv
send内容:ilcpsklryv
recv内容:ilcpsklryvmcpjn
send内容:ilcpsklryv
send内容:ilcpsklryv
█
```

第二个 server 端连接，端口为 5000：

```
[root@RHEL74-SVR 05]# ./tcp_server5 5000
与客户端连接成功!
send内容:ilcpsklryv
send内容:ilcpsklryv
send内容:ilcpsklryv
recv内容:ilcpsklryvmcpjn
send内容:ilcpsklryv
send内容:ilcpsklryv
send内容:ilcpsklryv
recv内容:ilcpsklryvmcpjn
send内容:ilcpsklryv
send内容:ilcpsklryv
send内容:ilcpsklryv
recv内容:ilcpsklryvmcpjn
send内容:ilcpsklryv
█
```

client 端连接：

```
[root@RHEL74-SVR 05]# ./tcp_client5-2 192.168.80.231 4000 5000
服务端1号连接成功!
服务端2号连接成功!
1号服务端recv内容ilcpsklryv
2号服务端recv内容:ilcpsklryv
1号服务端recv内容ilcpsklryv
2号服务端recv内容:ilcpsklryv
1号服务端recv内容ilcpsklryv
2号服务端recv内容:ilcpsklryv
send内容:ilcpsklryvmcpjn
2号服务端recv内容:ilcpsklryv
1号服务端recv内容ilcpsklryv
2号服务端recv内容:ilcpsklryv
1号服务端recv内容ilcpsklryv
2号服务端recv内容:ilcpsklryv
1号服务端recv内容ilcpsklryv
send内容:ilcpsklryvmcpjn
1号服务端recv内容ilcpsklryv
2号服务端recv内容:ilcpsklryv
1号服务端recv内容ilcpsklryv
2号服务端recv内容:ilcpsklryv
1号服务端recv内容ilcpsklryv
2号服务端recv内容:ilcpsklryv
send内容:ilcpsklryvmcpjn
2号服务端recv内容:ilcpsklryv
1号服务端recv内容ilcpsklryv
^C
```