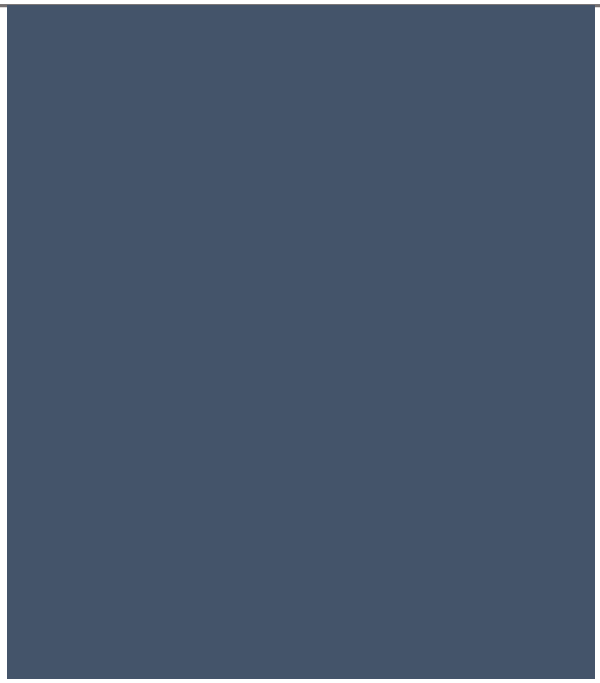


linux-socket-tcp

async

张伯阳 1551265

1-1

设置非阻塞

client

```
        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;

    char recvbuf[BUFFER_SIZE];
    while (1)
    {
        memset(recvbuf, 0, sizeof(recvbuf));
```

server

```
    ///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, &length);

    int flags2 = fcntl(conn, F_GETFL, 0);                          //获取文件的flags2值。
    fcntl(conn, F_SETFL, flags2 | O_NONBLOCK);    //设置成非阻塞模式;

    if(conn<0)
    {
        perror("connect");
        exit(1);
    }
    else
    cout<<"连接成功!"<<endl;
```

连接成功后直接退出

```
[root@RHEL-zby test]# ./tcp_server1-1 4000
连接成功!
[root@RHEL-zby test]#
```

```
[root@RHEL-zby test]# ./tcp_client1-1 192.168.2.231 4000
连接成功!
[root@RHEL-zby test]#
```

为了得到 recv 函数的返回值 修改代码将其输出 得到其返回值为-1

1-2

用 switch 控制 select 的输出 写成模版形式 方便之后超时处理

server

```
int flag;
fd_set  fdR;
FD_ZERO(&fdR);
FD_SET(conn, &fdR);
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);
}
close(conn);
close(server_sockfd);
```

client

```
char recvbuf[BUFFER_SIZE];
int flag;
fd_set  fdR;
    FD_ZERO(&fdR);
    FD_SET(sock_cli, &fdR);
    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);
    }
close(sock_cli);
```

运行结果

```
[root@RHEL-zby test]# ./tcp_server1-3 4000
连接成功！

[root@RHEL-zby test]# ./tcp_client1-2 192.168.2.231 4000
连接成功！
```

1-3

server 端加入 connect 之前的非阻塞控制 加一个 select 进行非阻塞连接

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

```
socklen_t length = sizeof(client_addr);
fd_set  fdR;
FD_ZERO(&fdR);
FD_SET(server_sockfd, &fdR);
int conn;
///成功返回非负描述字，出错返回-1
switch (select(server_sockfd + 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:
        conn = accept(server_sockfd, (struct sockaddr*)&client_addr, &length);
}
cout<<"连接成功!"<<endl;

int flags2 = fcntl(conn, F_GETFL, 0);                    //获取文件的flags2值。
```

client 端将设置非阻塞的位置调整

```
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

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;


///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);
}
```

运行截图

```
[root@RHEL-zby test]# ./tcp_server1-3 4000
连接成功！

[root@RHEL-zby test]# ./tcp_client1-2 192.168.2.231 4000
连接成功！
```

2-1

两边连接成功后 server 端开始输出收到的字节

```
[root@RHEL-zby test]# ./tcp_server2-1 4000
连接成功！
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789

[root@RHEL-zby test]# ./tcp_client2-1 192.168.2.231 4000
连接成功！
```

将 client 用 ctrl+c 断开 server 端检测不到 输出停止但没有退出

```
[root@RHEL-zby test]# ./tcp_server2-1 4000
连接成功！
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
```

```
[root@RHEL-zby test]# ./tcp_client2-1 192.168.2.231 4000
连接成功！
^C
[root@RHEL-zby test]#
```

将 server 端用 ctrl+c 断开 client 端也断开

```
[root@RHEL-zby test]# ./tcp_server2-1 4000
连接成功！
0123456789
^C
[root@RHEL-zby test]#
```

```
[root@RHEL-zby test]# ./tcp_client2-1 192.168.2.231 4000
连接成功！
[root@RHEL-zby test]#
```

server 端 kill -9 杀死 client 断

```
0123456789
0123456789
已杀死
[root@RHEL-zby test]#
```

```
[root@RHEL-zby test]# ./tcp_client2-1 192.168.2.231 4000
连接成功!
[root@RHEL-zby test]#
```

client 端 kill -9 杀死 server 端继续输出 检测不到

```
0123456789
0123456789
0123456789
0123456789
```

```
[root@RHEL-zby test]# ./tcp_client2-1 192.168.2.231 4000
连接成功!
已杀死
[root@RHEL-zby test]#
```

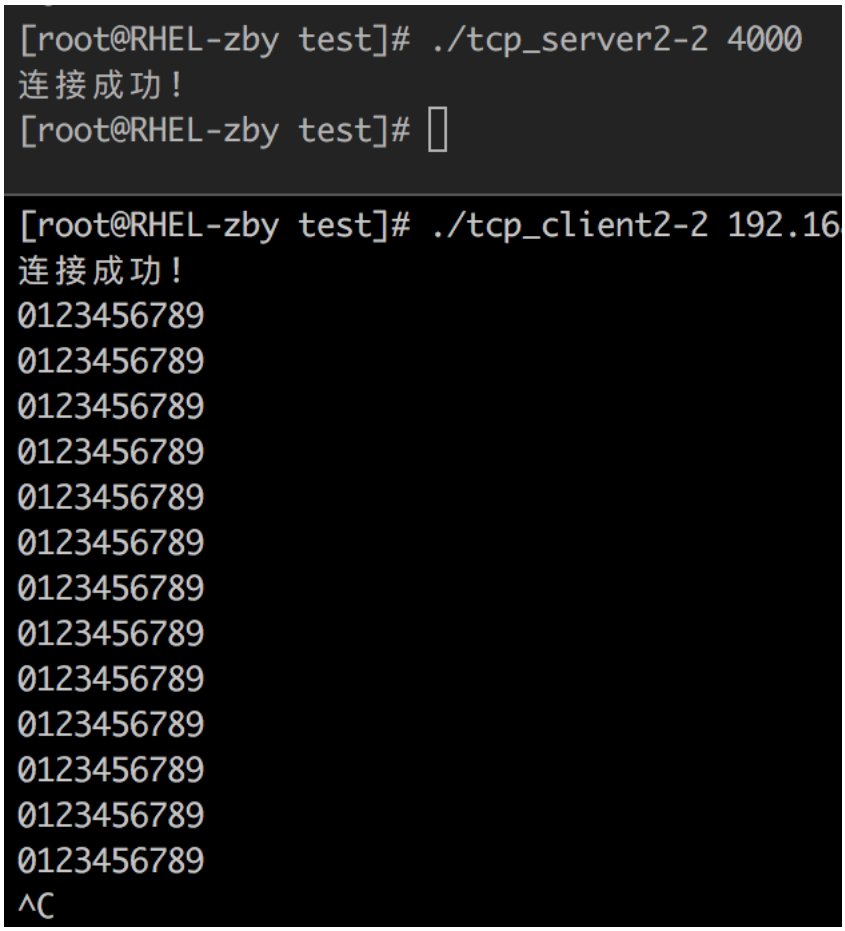
2-2

两边连通 client 端开始输出收到的字节

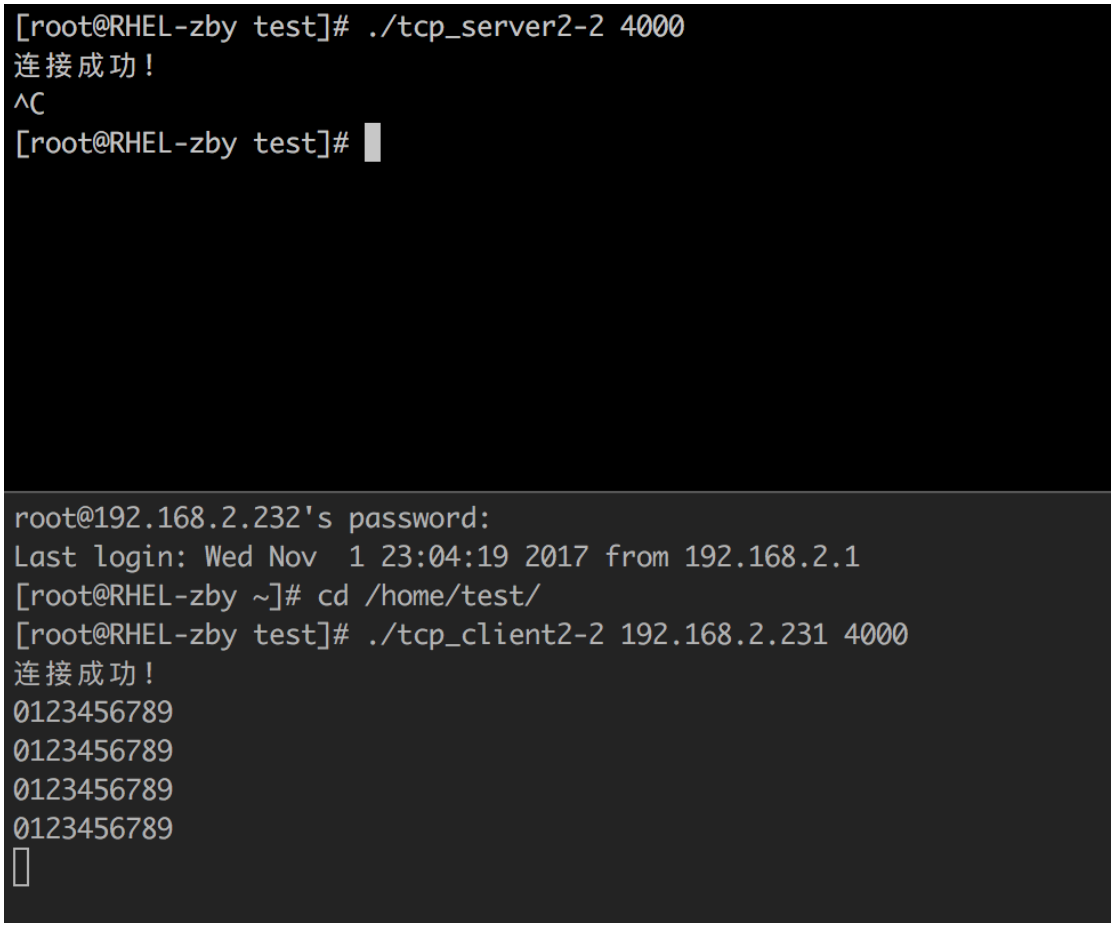
```
[root@RHEL-zby test]# ./tcp_server2-2 4000
连接成功!
```

```
[root@RHEL-zby test]# ./tcp_client2-2 192.168.2.231 4000
连接成功!
0123456789
0123456789
0123456789
0123456789
```

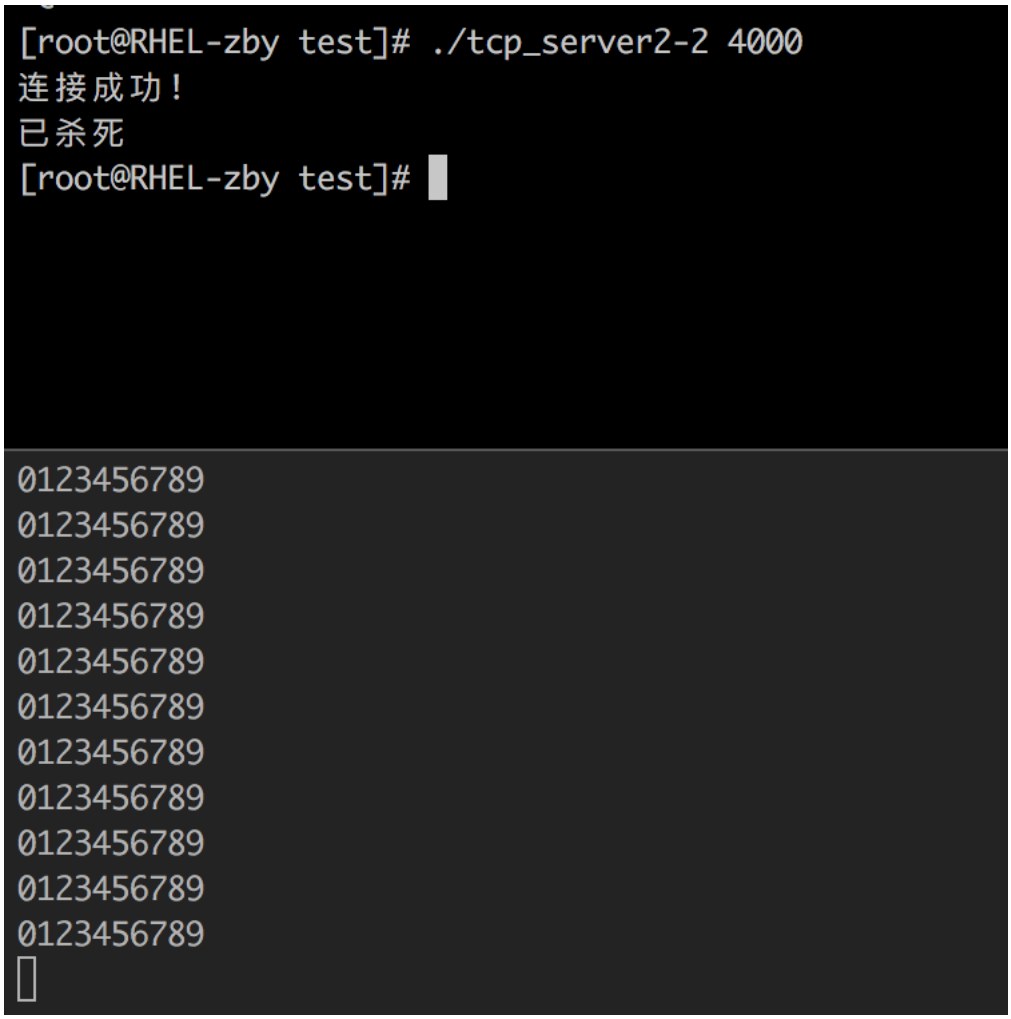
ctrl+c 掉 client server 退出



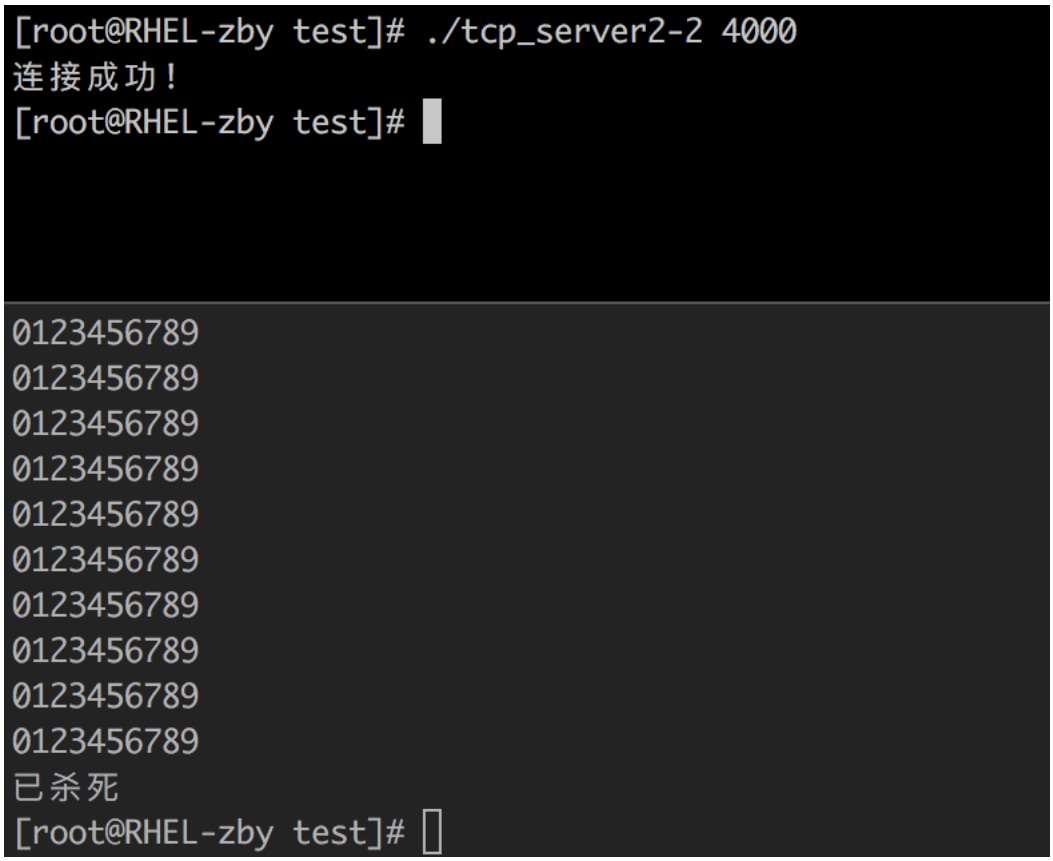
ctrl+c 掉 server client 停止不退出



杀死 server client 停止不退出



杀死 client server 退出


[root@RHEL-zby test]# ./tcp_server2-2 4000
连接成功！
[root@RHEL-zby test]#
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
0123456789
已杀死
[root@RHEL-zby test]#

3-1

由于 sleep 信号与 select 信号是平等的 两个一起运行会有冲突

而定时器 alarm 会发送中断给 select 两个可以并行

故使用 alarm 信号进行定时控制

alarm 不能写在 select 函数内 否则会持续阻塞

很明显发信号周期和收信号周期没有任何关系 所以将 alarm 写在 while 外

通过 alarm 信号执行的函数内部再次执行 alarm 实现循环间隔 TIMEOUT 时间发信号

由于该函数不便于传参 故直接将需要用到的参数设为全局变量

server 端和 client 端实现方式完全一样 只进行一次截图

```
    int flags2 = fcntl(conn, F_GETFL, 0);                       //获取文件的flags2值。
    fcntl(conn, F_SETFL, flags2 | O_NONBLOCK);   //设置成非阻塞模式;

    alarm(3);

    while(1)
    {
        FD_ZERO(&fdR);
        FD_SET(conn, &fdR);
        switch (select(conn + 1, &fdR, NULL,NULL, NULL)) //非阻塞recv
        {
            case 0:
                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));
                int len=recv(conn, buffer, sizeof(buffer),MSG_DONTWAIT);
                //cout<<"len="<<len<<endl;
                if(len>0)
                {
                    cout<<"recv=";
                    for(num=0;num<len;num++)
                        cout<<buffer[num];
                    cout<<endl;
                }
        }
    }
    close(conn);
```

```
#define SEND_SIZE 10
#define TIMEOUT 1
int conn,num;
char buffer[BUFFER_SIZE],send_buf[SEND_SIZE];

void CbSigAlrm(int signo)
{
    send(conn, send_buf, sizeof(send_buf),MSG_DONTWAIT);
    cout<<"send=";
    int i;
    for(i=0;i<sizeof(send_buf);i++)
        cout<<send_buf[i];
    cout<<endl;
    alarm(TIMEOUT);
}

int main(int argc,char* argv[])
{
    ///定义sockfd
    int server_sockfd = socket(AF_INET,SOCK_STREAM, 0);
```

运行结果符合预期

任意一端退出另一端在发信号时都能检测到并退出(不再截图)

```
[root@RHEL-zby test]# ./tcp_server3-1 4000
连接成功!
send=0123456789
send=0123456789
recv=0123456789:;<=>
send=0123456789
send=0123456789
send=0123456789
recv=0123456789:;<=>
send=0123456789
send=0123456789
send=0123456789
recv=0123456789:;<=>
send=0123456789
```

```
[root@RHEL-zby test]# ./tcp_client3-1 192.168.2.231 4000
连接成功!
recv=0123456789
recv=0123456789
send=0123456789:;<=>
recv=0123456789
recv=0123456789
recv=0123456789
send=0123456789:;<=>
recv=0123456789
recv=0123456789
recv=0123456789
send=0123456789:;<=>
recv=0123456789
```

3-2

非阻塞模式下无法实现阻塞读入 会无视 MSG_WAITALL 的参数

运行结果和之前一样

```
[root@RHEL-zby test]# ./tcp_server3-2 4000
连接成功!
send=0123456789
send=0123456789
recv=0123456789:;<=>
send=0123456789
send=0123456789
send=0123456789
recv=0123456789:;<=>
send=0123456789
send=0123456789
```

```
[root@RHEL-zby test]# ./tcp_client3-2 192.168.2.231 4000
连接成功!
recv=0123456789
recv=0123456789
send=0123456789:;<=>
recv=0123456789
recv=0123456789
recv=0123456789
send=0123456789:;<=>
recv=0123456789
recv=0123456789
```

4-1

运行结果如图所示

```
[root@RHEL-zby test]# ./tcp_server4-1 4000
连接成功!
发送字节数:1010900
发送字节数:1010920
发送字节数:1010940
发送字节数:1010960
发送字节数:1010980
发送字节数:1011000
发送字节数:1011020
发送字节数:1011040
发送字节数:1011060
发送字节数:1011080
发送字节数:1011100
发送字节数:1011120
发送字节数:1011140
发送字节数:1011160
发送字节数:1011180
发送字节数:1011200
发送字节数:1011220
```

2. root@RHEL-zby:~ (ssh)

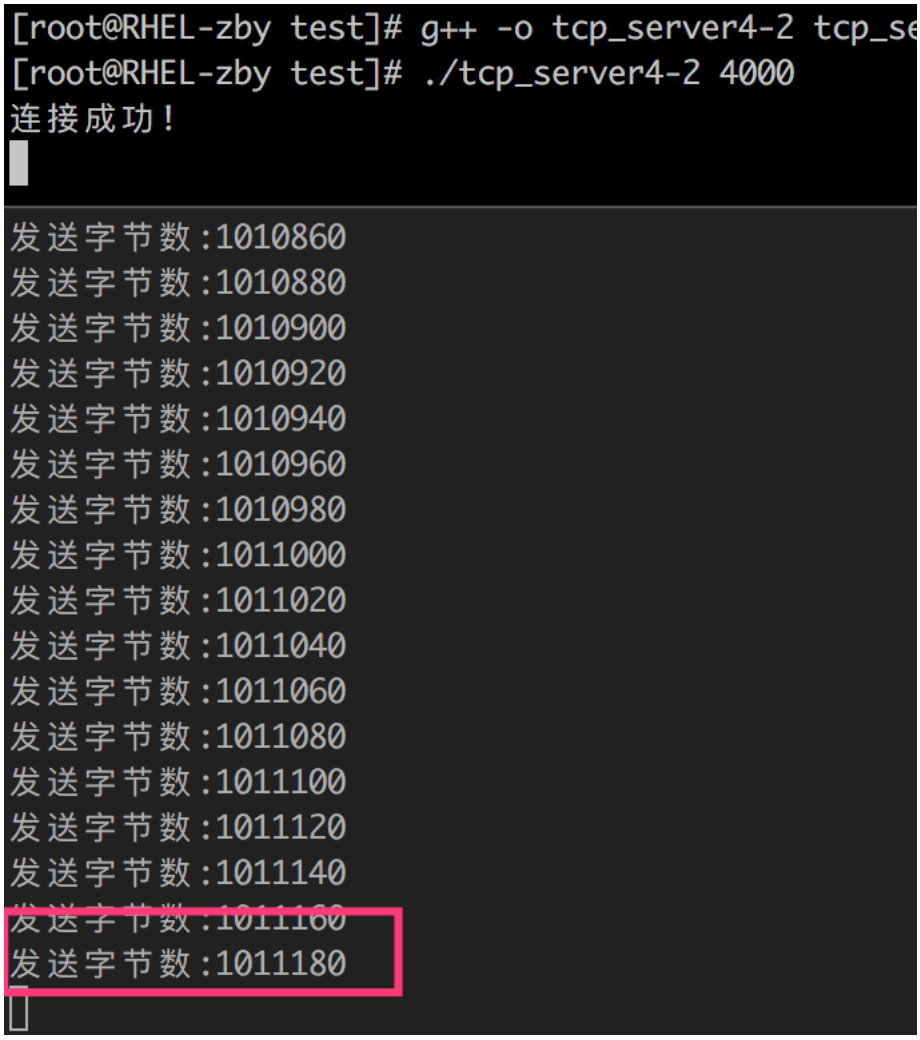
root@RHEL-zby:~ (ssh)

```
root@192.168.2.231's password:
Last login: Thu Nov  2 18:43:51 2017 from 192.168.2.1
[root@RHEL-zby ~]# netstat -t                                                    231
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp   251668      0 RHEL-zby:terabase       192.168.2.232:38260     ESTABLISHED
tcp        0      0 RHEL-zby:ssh            192.168.2.1:54789       ESTABLISHED
tcp        0      0 RHEL-zby:ssh            192.168.2.1:53216       ESTABLISHED
[root@RHEL-zby ~]#
```

```
root@192.168.2.232's password:
Last login: Wed Nov  1 23:43:42 2017 from 192.168.2.1
[root@RHEL-zby ~]# netstat -t                                                    232
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL-zby:ssh            192.168.2.1:54790       ESTABLISHED
tcp        0 759560 RHEL-zby:38260          192.168.2.231:terabase  ESTABLISHED
tcp        0      0 RHEL-zby:ssh            192.168.2.1:54784       ESTABLISHED
tcp        0      0 RHEL-zby:ssh            192.168.2.1:52030       ESTABLISHED
[root@RHEL-zby ~]#
```

4-2

运行结果如下

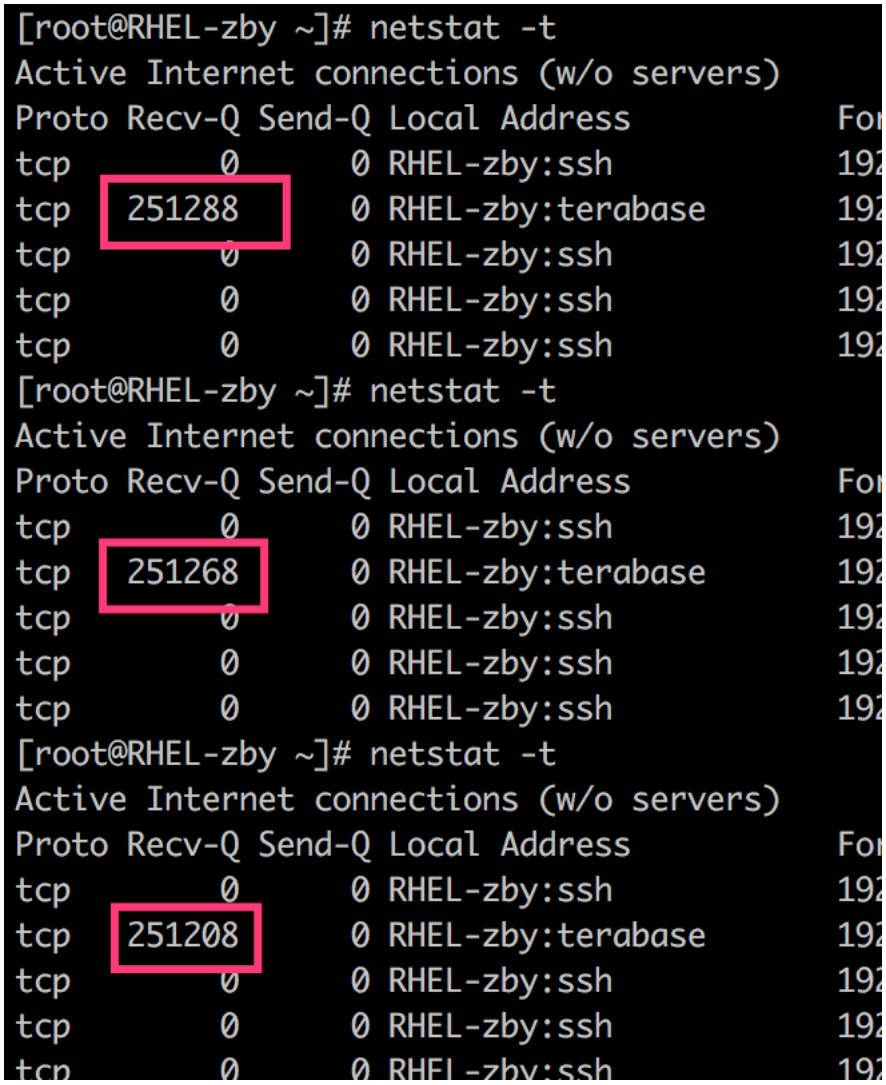


write 很快因为 server 的 Recv_Q 缓冲区和 client 的 Send_Q 都满了而停下来

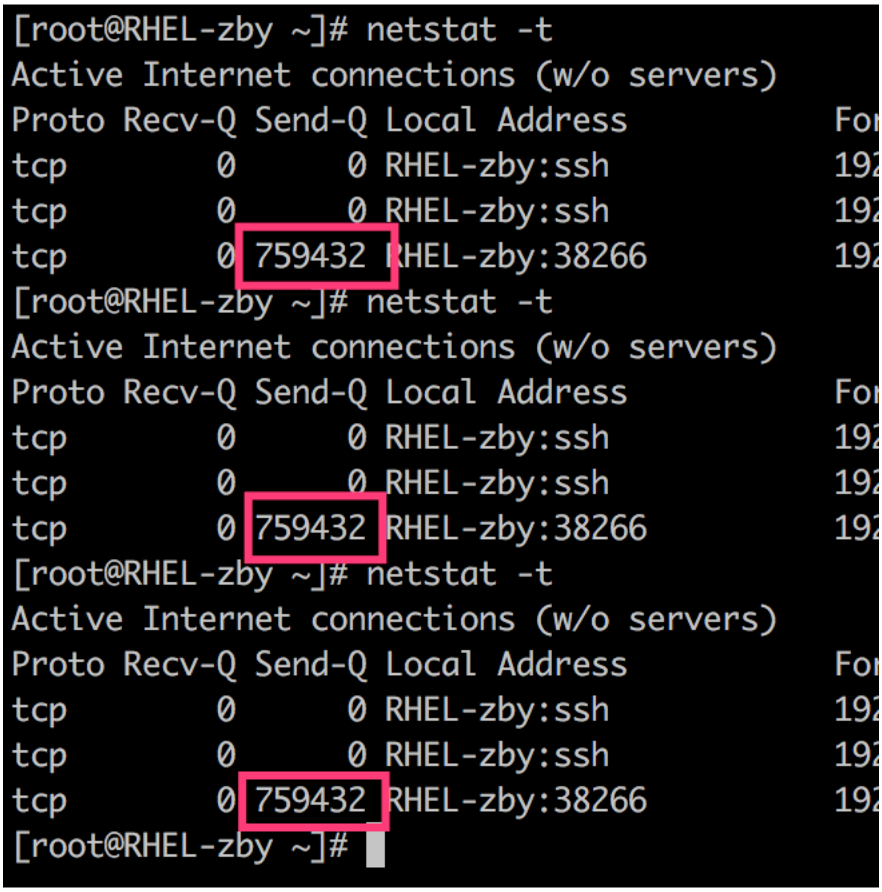
但是由于 server 端还在不断的读取数据

用 netstat -t 查看缓冲区数据变化

Recv_Q 先是慢慢减少



在此过程中 Send_Q 不变



因为 Send_Q 一直不变 Recv_Q 减少过慢 故将 server 每次读取的字节数调整为 5000....

server 端的 Recv_Q 先减后增 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 不变 Recv_Q 增加时 Send_Q 减少

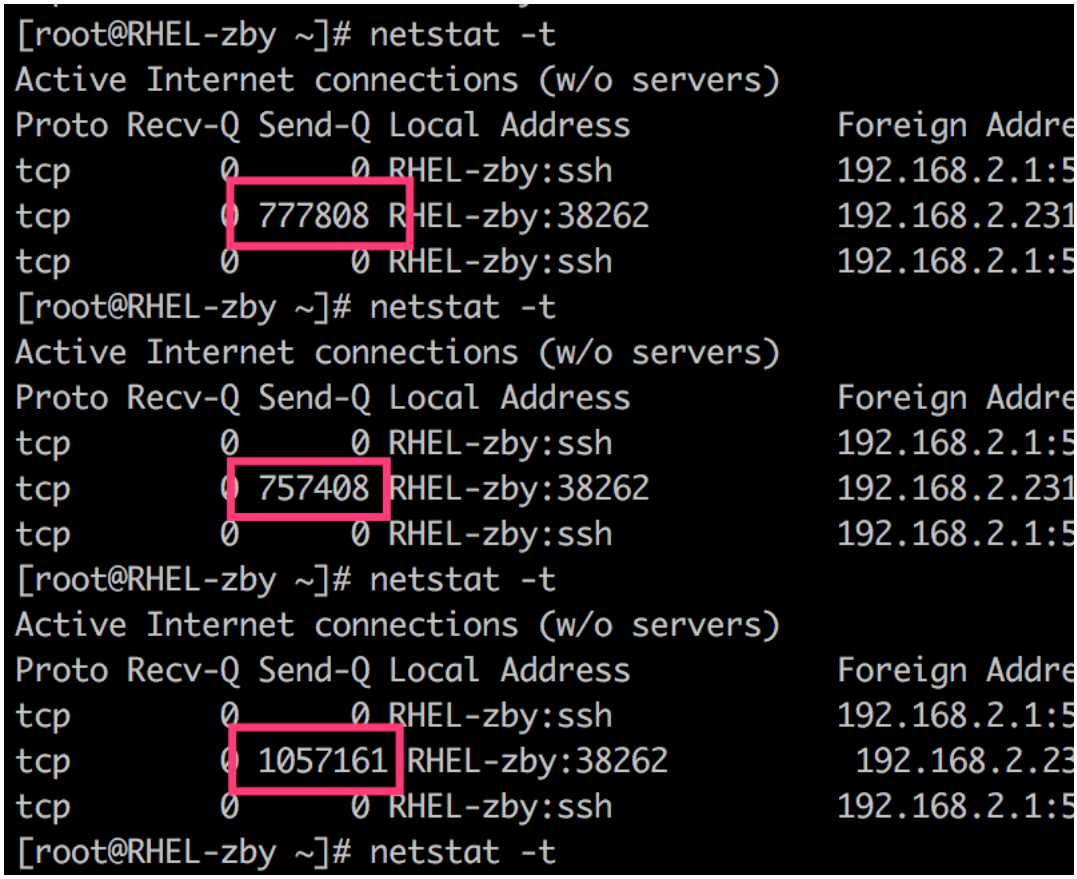
可以理解为 server 的读缓冲区因读取一点点减少 减少到一定量时 client 的写缓冲区将一部分数据移至 server 读缓冲区将其填满 此时 client 不会马上执行 write 测试过程发生这样的移动发生几次后 client 的 write 会恢复写入状态 很快又将自己的写缓冲区填满 然后这样的循环重新执行(手速慢 只截了少量的图)

server 的读缓冲区先减后增

```
[root@RHEL-zby ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Add
tcp        0      0 RHEL-zby:ssh           192.168.2.1
tcp        0      0 RHEL-zby:ssh           192.168.2.1
tcp        0      0 RHEL-zby:ssh           192.168.2.1
tcp   250084      0 RHEL-zby:terabase      192.168.2.2
tcp        0      0 RHEL-zby:ssh           192.168.2.1
[root@RHEL-zby ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Add
tcp        0      0 RHEL-zby:ssh           192.168.2.1
tcp        0      0 RHEL-zby:ssh           192.168.2.1
tcp        0      0 RHEL-zby:ssh           192.168.2.1
tcp   240084      0 RHEL-zby:terabase      192.168.2.2
tcp        0      0 RHEL-zby:ssh           192.168.2.1
[root@RHEL-zby ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Add
tcp        0      0 RHEL-zby:ssh           192.168.2.1
tcp        0      0 RHEL-zby:ssh           192.168.2.1
tcp        0      0 RHEL-zby:ssh           192.168.2.1
tcp   252620      0 RHEL-zby:terabase      192.168.2.2
tcp        0      0 RHEL-zby:ssh           192.168.2.1
```

client 的写缓冲区先减后增

```
[root@RHEL-zby ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Forei
tcp        0      0 RHEL-zby:ssh            192.1
tcp        0 974840 RHEL-zby:38262          192.1
tcp        0      0 RHEL-zby:ssh            192.1
[root@RHEL-zby ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Forei
tcp        0      0 RHEL-zby:ssh            192.1
tcp        0 892256 RHEL-zby:38262          192.1
tcp        0      0 RHEL-zby:ssh            192.1
[root@RHEL-zby ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Forei
tcp        0      0 RHEL-zby:ssh            192.1
tcp        0 892256 RHEL-zby:38262          192.1
tcp        0      0 RHEL-zby:ssh            192.1
[root@RHEL-zby ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Forei
tcp        0      0 RHEL-zby:ssh            192.1
tcp        0 846184 RHEL-zby:38262          192.1
tcp        0      0 RHEL-zby:ssh            192.1
```


[root@RHEL-zby ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address Foreign Addre
tcp 0 0 RHEL-zby:ssh 192.168.2.1:5
tcp 0 777808 RHEL-zby:38262 192.168.2.231
tcp 0 0 RHEL-zby:ssh 192.168.2.1:5
[root@RHEL-zby ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address Foreign Addre
tcp 0 0 RHEL-zby:ssh 192.168.2.1:5
tcp 0 757408 RHEL-zby:38262 192.168.2.231
tcp 0 0 RHEL-zby:ssh 192.168.2.1:5
[root@RHEL-zby ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address Foreign Addre
tcp 0 0 RHEL-zby:ssh 192.168.2.1:5
tcp 0 1057161 RHEL-zby:38262 192.168.2.23
tcp 0 0 RHEL-zby:ssh 192.168.2.1:5
[root@RHEL-zby ~]# netstat -t

5-1

运行结果

```
[root@RHEL-zby test]# ./tcp_server5-1 4000
连接成功!
send=0123456789
send=0123456789
send=0123456789
recv=0123456789:;<=>
send=0123456789
send=0123456789
send=0123456789
recv=0123456789:;<=>
send=0123456789
```

```
[root@RHEL-zby test]# ./tcp_client5-1 192.168.2.231 4000
连接成功!
recv=0123456789
recv=0123456789
recv=0123456789
send=0123456789:;<=>
recv=0123456789
recv=0123456789
recv=0123456789
send=0123456789:;<=>
recv=0123456789
```

第二个 server 连接

```
[root@RHEL-zby test]# ./tcp_client5-1 192.168.2.231 4000
连接成功!
recv=0123456789
recv=0123456789
recv=0123456789
send=0123456789:;<=>
recv=0123456789
```

5-2

运行结果

```
[root@RHEL-zby test]# ./tcp_server5-2 4000
连接成功!
send=0123456789
send=0123456789
recv=0123456789:;<=>
send=0123456789
```

```
[root@RHEL-zby test]# ./tcp_client5-2 192.168.2.231 4000 5000
1连接成功!
2连接成功!
1recv=0123456789
2recv=0123456789
1recv=0123456789
2recv=0123456789
send=0123456789:;<=>
2recv=0123456789
1recv=0123456789
```

另一个 server

```
[root@RHEL-zby test]# ./tcp_server5-2 5000
连接成功!
send=0123456789
send=0123456789
recv=0123456789:;<=>
send=0123456789
send=0123456789
send=0123456789
recv=0123456789:;<=>
send=0123456789
send=0123456789
send=0123456789
recv=0123456789:;<=>
send=0123456789
send=0123456789
send=0123456789
recv=0123456789:;<=>
```