同济大学计算机网络

实验报告

姓名：涂远鹏-1652262

题目：socket 编程 TCP 阻塞方式

零.补充知识

1.克隆虚拟机，新的虚拟机如何设置网卡并使得生效：

复制 ifcfg-ens32 为 ifcfg-ens32:0 并修改 IPADDR 为 192.168.80.232，DEVICE 修改为 ens32:0:

```
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
DEFROUTE=yes
IPV4_FAILURE_FATAL=yes
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=ens32
UUID=8e20dcd5-b4ff-440c-97df-750866534f24
DEVICE=ens32:0
ONBOOT=yes()
IPV6_PRIVACY=no
IPADDR=192.168.80.232
GATEWAY=192.168.80.2
PREFIX=24
DNS1=223.5.5.5
DNS2=223.6.6.6
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"ifcfg-ens32:0" 21L, 384C written
[root@RHEL74-SVR network-scripts]# _
```

2.如何在一个网卡中设置多地址：

测试上面新增 IP 地址的结果，用 ifconfig 显示 ens32:0 地址为 192.168.80.232：

```
192.168.80.231 ×
Last login: Sat Oct 20 16:11:04 2018
[root@RHEL74-SVR ~]# systemctl restart network
[root@RHEL74-SVR ~]# ifconfig
ens32: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.80.230  netmask 255.255.255.0  broadcast 192.168.80.255
        inet6 fe80::7f81:ad74:ddee:633e  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:1b:9f:41  txqueuelen 1000  (Ethernet)
        RX packets 348  bytes 33944 (33.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 305  bytes 33320 (32.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens32:0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.80.232  netmask 255.255.255.0  broadcast 192.168.80.255
        ether 00:0c:29:1b:9f:41  txqueuelen 1000  (Ethernet)

ens34: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.1  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::1abc:dcef:5a14:d8d9  prefixlen 64  scopeid 0x20<link>
        inet6 fe80::4933:d69b:e1cf:a671  prefixlen 64  scopeid 0x20<link>
        inet6 fe80::1de3:c3c7:9066:da6c  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:1b:9f:4b  txqueuelen 1000  (Ethernet)
        RX packets 18  bytes 1389 (1.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 33  bytes 2836 (2.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 8  bytes 480 (480.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 480 (480.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Ping192.168.80.231 与 192.168.80.232 都可以 ping 通(一开始 clone 完了没把 clone 机里的 ens32 的地址 192.168.80.230 改掉...)

```
192.168.80.231   192.168.80.230 ×
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]#
[root@RHEL74-SVR ~]# ping 192.168.80.231
PING 192.168.80.231 (192.168.80.231) 56(84) bytes of data.
64 bytes from 192.168.80.231: icmp_seq=1 ttl=64 time=0.343 ms
64 bytes from 192.168.80.231: icmp_seq=2 ttl=64 time=5.81 ms
64 bytes from 192.168.80.231: icmp_seq=3 ttl=64 time=1.16 ms
^Z
[5]+  已停止                 ping 192.168.80.231
[root@RHEL74-SVR ~]# ping 192.168.80.232
PING 192.168.80.232 (192.168.80.232) 56(84) bytes of data.
64 bytes from 192.168.80.232: icmp_seq=1 ttl=64 time=0.449 ms
64 bytes from 192.168.80.232: icmp_seq=2 ttl=64 time=0.968 ms
64 bytes from 192.168.80.232: icmp_seq=3 ttl=64 time=2.31 ms
64 bytes from 192.168.80.232: icmp_seq=4 ttl=64 time=0.950 ms
^Z
[6]+  已停止                 ping 192.168.80.232
[root@RHEL74-SVR ~]#
```

一.设置目录

略....

二.(01 目录)写一对 TCP socket 的测试程序, 分为 client 和 server 分别运行在不同虚拟机上

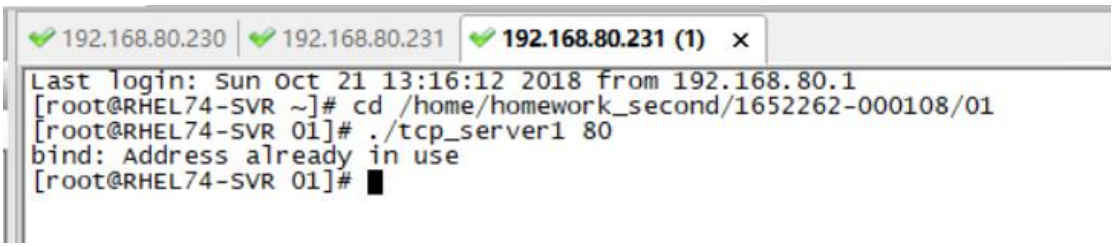
1.测试程序 tcp_server1，运行后绑定某个 TCP 端口号，并进入等待连接状态，要求端口号通过 main 函数带参数的方式传入:

答：进入等待连接状态:

```
[root@RHEL74-SVR 01]# ll
总用量 44
-rw-r--r--. 1 root root  1371 12月 19 2017 client.cpp
-rw-r--r--. 1 root root   132 12月 19 2017 makefile
-rw-r--r--. 1 root root  1708 12月 19 2017 server.cpp
-rwxr-xr-x. 1 root root 13992 10月 21 13:17 tcp_client1
-rwxr-xr-x. 1 root root 13800 10月 21 13:17 tcp_server1
[root@RHEL74-SVR 01]# ./tcp_server1 4000
```

2.如果服务端绑定的端口号已被使用, 则无法进入 LICENSE 状态, 会在那个函数上出错?

答：会在 bind 函数上出错，如图所示显示端口已被占用



3.测试 tcp_client1,连接对应端口显示连接成功:

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

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

4.

IP 地址不对时为 accept 函数返回值出错，返回值为负数，从而显示 No route to host:

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

```
[root@RHEL74-SVR 1652262-000108]# cd 01
[root@RHEL74-SVR 01]# ./tcp_client1 192.168.80.233 4000
connect: No route to host
```

5.

端口号不对时也是 accept 函数返回值出错，返回值为负数，从而显示 No route to host:

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

```
[root@RHEL74-SVR 01]# ./tcp_client1 192.168.80.233 4444
connect: No route to host
[root@RHEL74-SVR 01]#
```

6.显示连接成功提示:

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

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

7.连接成功后, 用 ctrl+C 终端 client 端测试, server 端可以侦察到连接中断, client 端 ctrl+c 之后 server 端直接进入终端了，不再是无法输命令的状态:

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

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

8.连接成功后在 secureCRT 的另一个终端用 kill -9 指令杀死./client 端，server 端可以检测出并退出:

```
[root@RHEL74-SVR 01]# ./tcp_server1 4000
连接成功!
kill -9
已杀死
```

连接成功后在 secureCRT 的另一个终端用 kill -9 指令杀死./server 端，client 端可以检测出并退出:

```
[root@RHEL74-SVR ~]# kill -9 2454
[root@RHEL74-SVR ~]# ps -ef
```

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

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

9.已有 client 与 server 连接成功以后，再登录一个终端 client 连接也会显示连接成功:

```
[root@RHEL74-SVR ~]# cd /home/homework_second/1652262-000108/01
[root@RHEL74-SVR 01]# ./tcp_client1 192.168.80.231 4000
连接成功!
```

10.tcp_server1 运行终止之后，立即再次启动绑定相同端口会显示端口被占用（Address already in use）,连接失败:

```
[root@RHEL74-SVR 01]# ./tcp_server1 4000
bind: Address already in use
```

在 socket 创建函数下接上:

```
if (setsockopt(server_sockfd, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int)) < 0)
    perror("setsockopt(SO_REUSEADDR) failed");
```

随后进行测试发现不会出错，再次运行之后会进行等待连接:

```
[root@RHEL74-SVR 01]# make
g++ -o tcp_server1 server.cpp
g++ -o tcp_client1 client.cpp
[root@RHEL74-SVR 01]# ./tcp_server1 4000
^C
[root@RHEL74-SVR 01]# ./tcp_server1 4000
```

区别即是：

如果定义了 REUSEADDR，只定义一个套接字在一个端口上进行监听，如果服务器出现意外而导致没有将这个端口释放，那么服务器重新启动后，你还可以用这个端口，因为你已经规定可以重用了，如果你没定义的话，你就会得到提示，ADDR 已在使用中，即该端口还在被占用的状态。

11.如果两台虚拟机的 IP 地址为同一网段，且均连接在同一虚拟机的网卡上但网段不同，两台虚拟机能否 ping 通，能否成功？

答：可以成功，因为由于二者连接在同一虚拟机的网卡上，二者在物理上是互通的就相当于一台虚拟机上的，而假如在同一虚拟机内则是无论是否在同一网段均可以 ping 通。

三.(02 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.....

2.测试结果如下，用 01 目录下的 client 程序连接 02 目录下写的 tcp_server2 服务器程序得到结果如下：

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

```
[root@RHEL74-SVR 02]# ./tcp_server2 4000
连接成功!
client IP:192.168.80.230  port:51358
```

3.连接 192.168.80.231 服务器端口 8887 显示测试结果：

```
[root@RHEL74-SVR 02]# ./tcp_client2 8887 192.168.80.231 4000
连接成功!
```

```
[root@RHEL74-SVR 02]# ./tcp_server2 4000
连接成功!
client IP:192.168.80.230  port:8887
```

连接 192.168.80.231 服务器端口 12345 端口显示测试结果：

```
[root@RHEL74-SVR 02]# ./tcp_client2 12345 192.168.80.231 4000
连接成功!
```

```
[root@RHEL74-SVR 02]# ./tcp_server2 4000
连接成功!
client IP:192.168.80.230  port:12345
```

四.(03 目录)

1.由于 client 机已设置多地址所以只需设置 server 机的多地址设置，server 地址增加一个 192.168.80.232：

```
[root@RHEL74-SVR network-scripts]# vi ifcfg-ens32:0
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
DEFROUTE=yes
IPV4_FAILURE_FATAL=yes
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=ens32
UUID=8e20dcd5-b4ff-440c-97df-750866534f24
DEVICE=ens32:0
ONBOOT=yes()
IPV6_PRIVACY=no
IPADDR=192.168.80.232
GATEWAY=192.168.80.2
PREFIX=24
DNS1=223.5.5.5
DNS2=223.6.6.6
~
[root@RHEL74-SVR network-scripts]# systemctl restart network
[root@RHEL74-SVR network-scripts]# ifconfig
ens32: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.80.231  netmask 255.255.255.0  broadcast 192.168.80.255
        inet6 fe80::7f81:ad74:ddee:633e  prefixlen 64  scopeid 0x20<link>
        inet6 fe80::aea9:e3db:c241:23cc  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:7e:2d:c3  txqueuelen 1000  (Ethernet)
        RX packets 2354  bytes 253607 (247.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1630  bytes 244332 (238.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens32:0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.80.232  netmask 255.255.255.0  broadcast 192.168.80.255
        ether 00:0c:29:7e:2d:c3  txqueuelen 1000  (Ethernet)

ens34: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.1  netmask 255.255.255.0  broadcast 192.168.2.255
        inet6 fe80::1abc:dcef:5a14:d8d9  prefixlen 64  scopeid 0x20<link>
        inet6 fe80::4933:d69b:e1cf:a671  prefixlen 64  scopeid 0x20<link>
        inet6 fe80::1de3:c3c7:9066:da6c  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:7e:2d:cd  txqueuelen 1000  (Ethernet)
        RX packets 28  bytes 2445 (2.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 55  bytes 4706 (4.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 16  bytes 960 (960.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 16  bytes 960 (960.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@RHEL74-SVR network-scripts]# ■
```

2.运行测试程序 tcp_server3 可以读取到服务端的所有 IP 地址，此处选择绑定的为 192.168.80.231 地址：

```
[root@RHEL74-SVR 03]# ./tcp_server3 192.168.80.231 4000
IP:192.168.2.1
IP:192.168.80.232
IP:192.168.80.231
IP:127.0.0.1
连接成功!
client IP:192.168.80.230  port:59990
■
```

```
[root@RHEL74-SVR 03]# ./tcp_client3 192.168.80.231 4000
连接成功!
```

3.运行测试程序 tcp_client3,运行时带入服务器地址 IP192.168.80.231 以及端口 4000，显示连接成功:

```
[root@RHEL74-SVR 03]# ./tcp_server3 192.168.80.231 4000
IP:192.168.2.1
IP:192.168.80.232
IP:192.168.80.231
IP:127.0.0.1
连接成功!
client IP:192.168.80.230  port:59990
```

```
[root@RHEL74-SVR 03]# ./tcp_client3 192.168.80.231 4000
连接成功!
```

4.如果连接服务端未绑定的 IP 地址时，会出现连接被拒绝的情况，此处服务端为 192.168.80.232 而客户端连接的为未绑定的 192.168.80.231IP 地址，出现以下情况:

```
[root@RHEL74-SVR 03]# ./tcp_client3 192.168.80.231 4000
connect: Connection refused
[root@RHEL74-SVR 03]#
```

五.(04 目录)

1.测试程序 tcp_client4-1-1，连接服务器成功之后，每次向服务端写入 30 字节内容，测试结果如下:

Client 端发送内容:

Server 端接受内容如下，server 端 read 函数返回值和返回内容接受的内容与 client 端写入的内容一致,但 server 端是分两次接收的,第一次 20 字节第二次输出 10 字节,而 client 端，添加了一秒的延时之后输出返回的信号为 30 字节连续字符流:

```
[root@RHEL74-SVR 04]# ./tcp_server4-1 4000
连接成功!
367535629127093606261879202375
922897361293194784503610632061
554765693745254744307868843149
206892664950487172722610615949
091771159776736563948129390885
096385611598481030444476317596
217857418597538831896433386048
889776430309254059469224775481
289368021051108506462586284724
062990813110340391969338056640
046267569872829960276132159914
910758704804296104222055290283
804091962544993605020
```

```
[root@RHEL74-SVR 04]# ./tcp_client4-1-1 192.168.80.231 4000
连接成功!
发送:367535629127093606261879202375
367535629127093606261879202375发送:9228973612931947845036106320 61
发送:554765693745254744307868843149
3610632061发送:20689266495048717272261061594 9
452547443078688431 49发送:09177115977673656394812939088 5
206892664950487172722610615949发送:0963856115984810304444763175 96
发送:217857418597538831896433386048
8129390885发送:889776430309254059469224775481
9848103044476317596发送:289368021051108506462586284724
217857418597538831896433386048发送:062990813110340391969338056640
发送:046267569872829960276132159914
9224775481发送:91075870480429610422205529028 3
511085064625862847 24发送:804091962544993605029435174314
062990813110340391969338056640发送:69422641288928886838333804768 9
发送:068790333732652658796041048708
6132159914发送:624793928301789154925749945935
042961042220552902 83发送:708199782534902019621207311905
804091962544993605029435174314发送:67740647485826066462896070101 3
发送:772645202909945103788604676097
^C
```

2.测试程序 tcp_client4-1-2,连接服务端成功后，每次用 write 写 2 字节，延时一秒再写两字节，测试结果如下：

Client 端 write 写入服务端数据如下:

Server 端接受内容如下，server 端 read 函数返回值和返回内容接受的内容与 client 端写入的内容一致：

```
[root@RHEL74-SVR 04]# ./tcp_client4-1-2 192.168.80.231 4000
连接成功!
发送:00
00发送:01
01发送:02
02发送:03
03发送:04
04发送:05
05发送:06
06发送:07
07发送:08
08发送:09
09发送:10
10发送:11
11发送:12
12发送:13
13发送:14
14发送:15
15发送:16
^C
```

server 端接收数据如下：

```
[root@RHEL74-SVR 04]# ./tcp_server4-1 4000
连接成功!
00
01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
```

由于此处 write 函数每次写入的 2 字节<20 字节，所以 server 端的 read 函数没有读满 20 个字节，直接返回 2 个字节，不用延时一秒。

3.tcp_server4-2/tcp_client_4-2-1 测试：

把 read/write 函数换成 send/recv 函数之后，server 每次读入 20 字节，client 每次发送 30 字节，但是与之前 read/write 函数不同的是，发现会出现每次返回 10 字节、20 字节、30 字节循环的情况：

```
[root@RHEL74-SVR 04]# ./tcp_client_4-2-1 192.168.80.231 4000
连接成功!
发送:367535629127093606261879202375
367535629127093606261879202375发送:92289736129319478450361063206l
发送:554765693745254744307868843149
3610632061发送:206892664950487172722610615949
4525474430786884314 9发送:091771159776736563948129390885
20689266495048717272261061594 9发送:096385611598481030444476317596
发送:217857418597538831896433386048
8129390885发送:889776430309254059469224775481
98481030444476317596发送:289368021051108506462586284724
217857418597538831896433386048发送:062990813110340391969338056640
发送:046267569872829960276132159914
9224775481发送:910758704804296104222055290283
51108506462586284724发送:804091962544993605029435174314
062990813110340391969338056640发送:694226412889288868383338047689
发送:068790333732652658796041048708
6132159914发送:624793928301789154925749945935
042961042220552902 3发送:708199782534902019621207311905
804091962544993605029435174314发送:677406474858260664628960701013
^C
```

```
[root@RHEL74-SVR 04]# chmod +x tcp_server4-2
[root@RHEL74-SVR 04]# ./tcp_server4-2 4000
连接成功!
367535629127093606 26
1879202375
92289736129319478450
3610632061
55476569374525474430
7868843149
20689266495048717272
2610615949
091771159776736563 94
8129390885
096385611598481030 44
4476317596
217857418597538831 89
6433386048
889776430309254059 46
9224775481
289368021051108506 46
2586284724
062990813110340391 96
9338056640
046267569872829960 27
```

4.tcp_server4-2/tcp_client_4-2-2 测试，与 tcp_client4-2 测试结果相同，由于读入小于 20 字节所以也是直接返回 server 端的 read 函数尚未读满的阻塞内容:

```
[root@RHEL74-SVR 04]# ./tcp_server4-2 4000
连接成功!
00
01
02
03
04
05
06
07
08
09
```

```
[root@RHEL74-SVR 04]# ./tcp_client_4-2-2 192.168.80.231 4000
连接成功!
发送:00
00发送:01
01发送:02
02发送:03
03发送:04
04发送:05
05发送:06
06发送:07
07发送:08
08发送:09
^C
[root@RHEL74-SVR 04]#
```

5.可以做到，只要在 server 端的 recv 函数中直接将最后一个参数改为 MSG_WAITALL 即可(recv 函数最后一个参数若改为 MSG_WAITALL 则会一直阻塞到指定的条件满足后才会返回)，显示测试结果如下，只有写入满了 20 字节才返回:

```
[root@RHEL74-SVR 04]# ./tcp_client4-1-2 192.168.80.231 4000
连接成功!
发送:00
纤续L发送:01
发送:02
发送:03
发送:04
发送:05
发送:06
发送:07
发送:08
发送:09
000102030405060708 09发送:10
发送:11
发送:12
发送:13
发送:14
发送:15
发送:16
发送:17
发送:18
发送:19
10111213141516171819发送:20
^Z
[2]+  已停止               ./tcp_client4-1-2 192.168.80.231 4000
[root@RHEL74-SVR 04]#
```

```
[root@RHEL74-SVR 04]# ./tcp_server4-2 4000
连接成功!
00010203040506070809
10111213141516171819
```

在 server1.cpp 中修改 read 函数，修改 read 函数缓冲区下限，在 cpp 中加入

Int recv_min_size=20;

Setsocketopt(server_sockfd,SOL_SOCKET,SO_RCVLOWAT,(void *)&recv_min_size,sizeof(int));

测试结果如下：每次等到 20 字节满了才会返回

```
[root@RHEL74-SVR 04]# ./tcp_client_4-2-2 192.168.80.231 4000
连接成功!
发送:00
姥垦发送:01
发送:02
发送:03
发送:04
发送:05
发送:06
发送:07
发送:08
发送:09
000102030405060708 09发送:10
发送:11
发送:12
发送:13
发送:14
发送:15
发送:16
发送:17
发送:18
发送:19
10111213141516171819发送:20
```

```
[root@RHEL74-SVR 04]# ./tcp_server4-1 4000
连接成功!
00010203040506070809
10111213141516171819
```

7,8.read 与 recv 函数区别，write 函数与 send 函数区别如下：

Write 函数将 buf 中的 nbytes 字节内容写入到文件描述符中，成功返回写的字节数，失败返回-1.并设置 errno 变量。在网络程序中，当我们向套接字文件描述舒服写数据时有两种可能：

1、write 的返回值大于 0，表示写了部分数据或者是全部的数据，这样用一个 while 循环不断的写入数据，但是循环过程中的 buf 参数和 nbytes 参数是我们自己来更新的，也就是说，网络编程中写函数是不负责将全部数据写完之后再返回的，说不定中途就返回了！

2、返回值小于 0，此时出错了，需要根据错误类型进行相应的处理。

如果错误是 EINTR 表示在写的时候出现了中断错误，如果是 EPIPE 表示网络连接出现了问题。

Read 函数是负责从 fd 中读取内容，当读取成功时，read 返回实际读取到的字节数，如果返回值是 0，表示已经读取到文件的结束了，小于 0 表示是读取错误。

如果错误是 EINTR 表示在写的时候出现了中断错误，如果是 EPIPE 表示网络连接出现了问题。

Recv 函数和 read 函数提供了 read 和 write 函数一样的功能，不同的是他们提供了四个参数。前面的三个参数和 read、write 函数是一样的，最后一个参数可以是零或者是以下组合：

MSG_DONTROUTE	不查找表
MSG_OOB	接收或者发送带外数据
MSG_PEEK	查看数据并不从缓冲区移走数据
MSG_WAITALL	等待所有数据

设置为 0 时与 read 和 write 函数一样，如果 recv 最后一个参数改为 MSG_WAITALL 时则会一直阻塞到满足给定的条件或者是发生错误为止：

(1)当读到指定的字节时，函数返回正常值，返回值小于 len 值

(2)当读到文档的结尾时，函数返回正常值，返回值小于 len 值

(3)当操作发生错误时，recv 返回值错误值-1，显示配置错误号，设置 errno 变量

六.(05 目录)

1.测试 tcp_server5-1 ，连接 client 成功之后，另外登录一个 server 的终端用 netstat -t 查看 tcp 连接情况：

服务端查看，连接成功之后，等待 getchar（）输入：

```
[root@RHEL74-SVR 05]# ./tcp_server5-1 4000
连接成功!
```

2.连接成功之后，write 函数向服务器写入数据，大约写入 380000 字节后会使 write 函数不再返回，进入阻塞状态：

```
[root@RHEL74-SVR 05]# ./tcp_client5-1 192.168.80.231 4000
连接成功!
发送字节数:20000
发送字节数:40000
发送字节数:60000
发送字节数:80000
发送字节数:100000
发送字节数:120000
发送字节数:140000
发送字节数:160000
发送字节数:180000
发送字节数:200000
发送字节数:220000
发送字节数:240000
发送字节数:260000
发送字节数:280000
发送字节数:300000
发送字节数:320000
发送字节数:340000
发送字节数:360000
发送字节数:380000
```

Server 服务端 netstat -t 最终结果(tcp 265760):

```
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:49661      ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:50768      ESTABLISHED
tcp   265760      0 RHEL74-SVR:terabase     192.168.80.230:60132    ESTABLISHED
tcp        0      0 RHEL74-SVR:36306        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]#
```

Client 用户端 netstat -t 最终结果(tcp 130320):

```
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State

tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:50831      ESTABLISHED
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:52665      ESTABLISHED
tcp        0      0 RHEL74-SVR:59528        192.168.80:microsoft-ds ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:50828      ESTABLISHED
tcp        0 130320 RHEL74-SVR:60132        192.168.80.231:terabase ESTABLISHED
[root@RHEL74-SVR ~]#
```

注：Recv-Q：表示收到的数据已经在本地接收缓冲，但是还有多少没有被进程取走，recv()，换句话说就是表示程序总共还有多少字节的数据没有从内核空间的套接字缓存拷贝到用户空间。

Send-Q：对方没有收到的数据或者说没有 Ack 的,还是本地缓冲区.

Recv-Q 和 Send-Q 详解

1. Recv-Q 表示网络接收队列

a)此列的值通常应该为 0，也可以短暂的不为 0，如果这个数值过大，说明接收队列有数据堆积。如果常时间不为 0 有可能遭受到了 denial-of-service（拒绝服务）攻击，也就是我们常说的 DoS 攻击。简单的讲就是把你带宽或者资源消耗完，无法正常为真正需要服务的用户提供服务直到崩溃。

b)既然有这个危险存在，那么我们应该如何防范。首先们们应该谨慎的维护我们的 OS，确保无安全隐患和漏洞，安装硬件防火墙设备（推荐），定期查看安全设备的日志

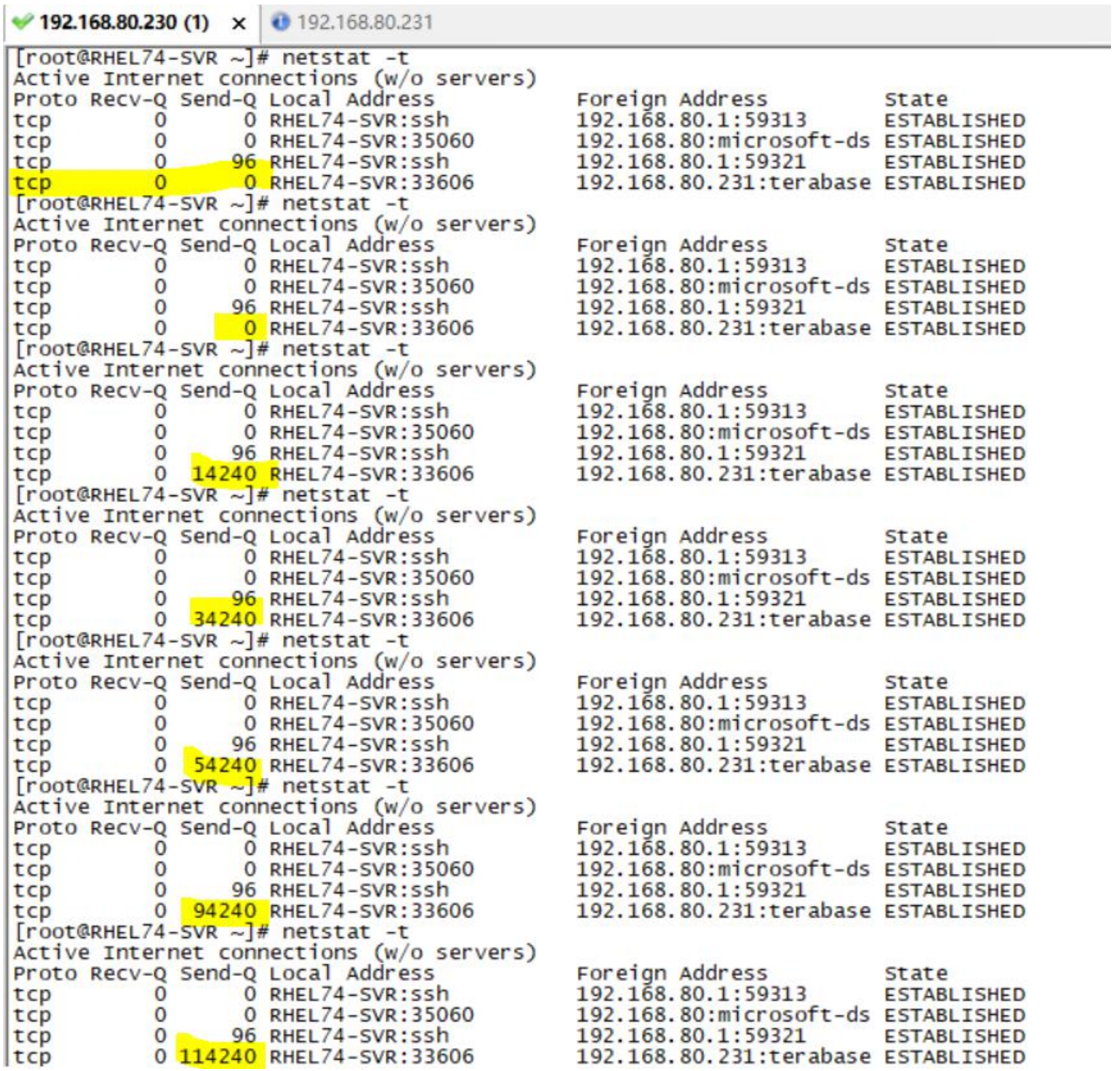
2.Send-Q 表示网络发送队列

a)此列的值也通常应该为 0，也可以短暂的不为 0，如果这个数值过大，发明发送队列有数

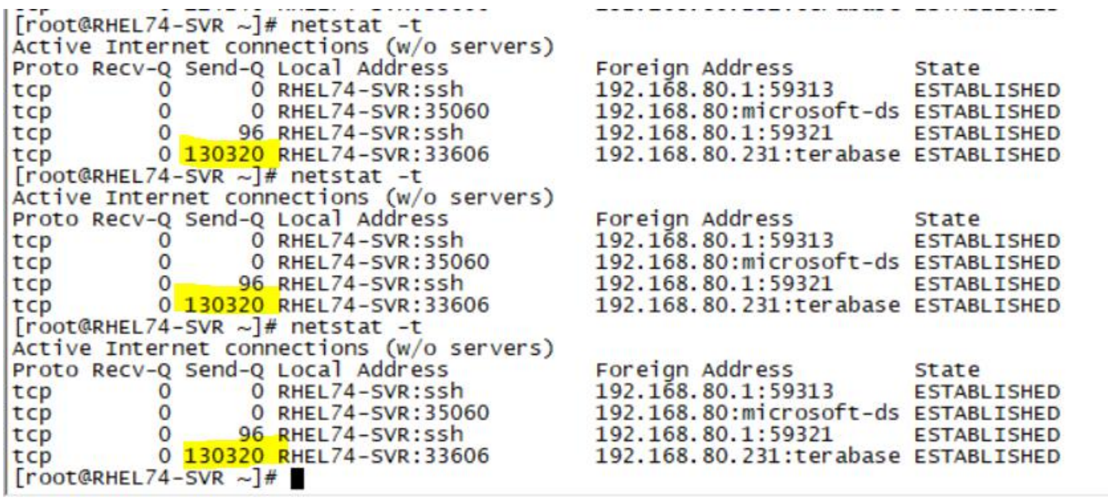
据堆积。如果常时间不为 0 有可能是向外发送数据过快，或者对方接收太慢，导致发出去的数据还停留在本地缓冲区

3.getchar()打开阻塞之后，server 会一直读下去，不会进入阻塞状态,要解释这一情况首先分析上面的情况，getchar()之前，也就是从 0 到 380000 字节过程中 client 端，server 端的 netstat -t 查询结果如下：

Client 端查询结果：



```
192.168.80.230 (1)  ×   192.168.80.231
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:59313      ESTABLISHED
tcp        0      0 RHEL74-SVR:35060        192.168.80:microsoft-ds ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:59321      ESTABLISHED
tcp        0      0 RHEL74-SVR:33606        192.168.80.231:terabase ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:59313      ESTABLISHED
tcp        0      0 RHEL74-SVR:35060        192.168.80:microsoft-ds ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:59321      ESTABLISHED
tcp        0      0 RHEL74-SVR:33606        192.168.80.231:terabase ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:59313      ESTABLISHED
tcp        0      0 RHEL74-SVR:35060        192.168.80:microsoft-ds ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:59321      ESTABLISHED
tcp        0  14240 RHEL74-SVR:33606        192.168.80.231:terabase ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:59313      ESTABLISHED
tcp        0      0 RHEL74-SVR:35060        192.168.80:microsoft-ds ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:59321      ESTABLISHED
tcp        0  34240 RHEL74-SVR:33606        192.168.80.231:terabase ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:59313      ESTABLISHED
tcp        0      0 RHEL74-SVR:35060        192.168.80:microsoft-ds ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:59321      ESTABLISHED
tcp        0  54240 RHEL74-SVR:33606        192.168.80.231:terabase ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:59313      ESTABLISHED
tcp        0      0 RHEL74-SVR:35060        192.168.80:microsoft-ds ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:59321      ESTABLISHED
tcp        0  94240 RHEL74-SVR:33606        192.168.80.231:terabase ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:59313      ESTABLISHED
tcp        0      0 RHEL74-SVR:35060        192.168.80:microsoft-ds ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:59321      ESTABLISHED
tcp        0 114240 RHEL74-SVR:33606        192.168.80.231:terabase ESTABLISHED
```



```
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:59313      ESTABLISHED
tcp        0      0 RHEL74-SVR:35060        192.168.80:microsoft-ds ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:59321      ESTABLISHED
tcp        0 130320 RHEL74-SVR:33606        192.168.80.231:terabase ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:59313      ESTABLISHED
tcp        0      0 RHEL74-SVR:35060        192.168.80:microsoft-ds ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:59321      ESTABLISHED
tcp        0 130320 RHEL74-SVR:33606        192.168.80.231:terabase ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:59313      ESTABLISHED
tcp        0      0 RHEL74-SVR:35060        192.168.80:microsoft-ds ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:59321      ESTABLISHED
tcp        0 130320 RHEL74-SVR:33606        192.168.80.231:terabase ESTABLISHED
[root@RHEL74-SVR ~]#
```

Server 端查询结果：

```
[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          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51922        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp    40000      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51936        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp    60000      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51936        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp    80000      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51936        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp   100000      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51936        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp   120000      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51936        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp   120000      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51936        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp   140000      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51936        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp   160000      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
```

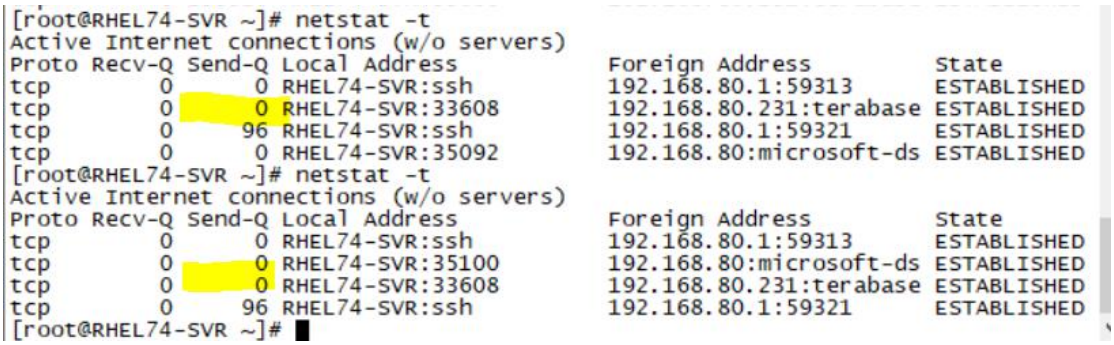
```
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp   200000      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51936        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp   220000      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51936        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp   240000      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51936        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp   260000      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51936        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp   265760      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51936        192.168.80:microsoft-ds ESTABLISHED
```

从上面截图可以看到：

(1) 开始时，client 端 Send -Q(未被内核处理字节数)为 0，此时数据全部被内核接受，无

数据在队列中
(2) server 端 recv -Q 值从一开始便随着 client 端发送数据增加而增加, 然后到达 265760 字节最大值保持不变，即开始 server 端接受来自 client 端数据时是阻塞在了内核并没有被 read 函数读取，内核阻塞区满了之后就开始拒收 client 端来的数据，client 端也继续发送数据直至自己的 send -Q 区（把自己的内核区也塞满了），然后 client 端就也进入了阻塞状态了，即此时 client 端 send -Q 值增加到 130320 后停止发送
(3) write 发出的信息阻塞在内核 直到达到最大值停止 write，两个最大值之和为 396080 在 380000 到 400000 之间 符合预期即 265760 字节阻塞在 server 内核，剩下 130320 字节阻塞在 client 端内核，此时双方的内核区都满了。

然后就是解释 getchar（）之后为什么可以一直读下去，因为之前 server 端是没处理数据的，getchar（）之后，开始 read，server 端把数据读入内核，server 端的 recv -Q 就不会有数据在队列，即没数据阻塞在内核，server 端的 recv -Q 都没满，client 端的 Send -Q 就更不会有数据阻塞在内核了，所以双方可以正常收发数据，不会阻塞。
Getchar（）之后显示如下:
Client 端（接收数据情况，Send -Q 情况均为 0）：



server 端（recv -Q 情况均为 0）：

```
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51980        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51980        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51980        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51980        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51980        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:terabase     192.168.80.230:33608    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:51980        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# █
```

4.某网站截的 netstat 命令图：

netstat 命令详解

netstat命令是一个监控TCP/IP网络的非常有用的工具，它可以显示路由表、实际的网络连接以及每一个网络接口设备的状态信息。

语法选项

netstat [选项]

-a或—all：显示所有连线中的Socket；

-A<网络类型>或—<网络类型>：列出该网络类型连线中的相关地址；

-c或—continuous：持续列出网络状态；

-C或—cache：显示路由器配置的快取信息；

-e或—extend：显示网络其他相关信息；

-F或—fib：显示FIB；

-g或—groups：显示多重广播功能群组组员名单；

-h或—help：在线帮助；

-i或—interfaces：显示网络界面信息表单；

-l或—listening：显示监控中的服务器的Socket；

-M或—masquerade：显示伪装的网络连线；

-n或—numeric：直接使用ip地址，而不通过域名服务器；

-N或—netlink或—symbolic：显示网络硬件外围设备的符号连接名称；

-o或—timers：显示计时器；

-p或—programs：显示正在使用Socket的程序识别码和程序名称；

-r或—route：显示Routing Table；

-s或—statistice：显示网络工作信息统计表；

-t或—tcp：显示TCP传输协议的连线状况；

-u或—udp：显示UDP传输协议的连线状况；

-v或—verbose：显示指令执行过程；

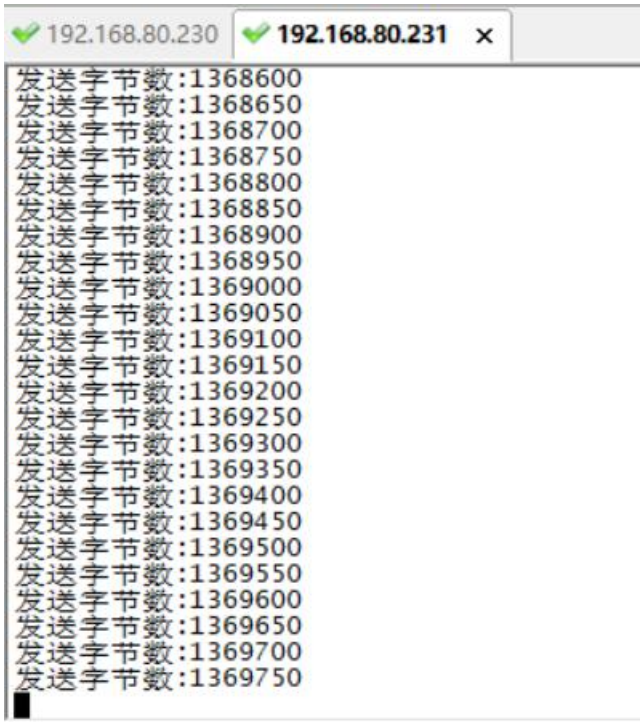
-V或—version：显示版本信息；

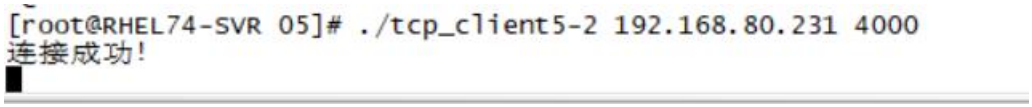
-w或—raw：显示RAW传输协议的连线状况；

-x或—unix：此参数的效果和指定"-A unix"参数相同；

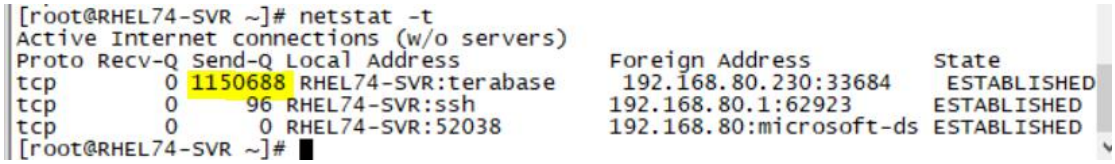
—ip或—inet：此参数的效果和指定"-A inet"参数相同。

Send -Q 和 Recv -Q 上面已解释过…
proto 是协议类型此处为 tcp，local address 为本地地址，foreign address 是连接的地址，state 为连接的状态，连接成功则显示 established
4.测试程序 tcp_server2/tcp_client2，双方角色互换：
Client 与 server 分别收发信号，每次 50 字节，测试结果显示如下：
Server 端到 1369750 字节时进入阻塞状态，等待 getchar():

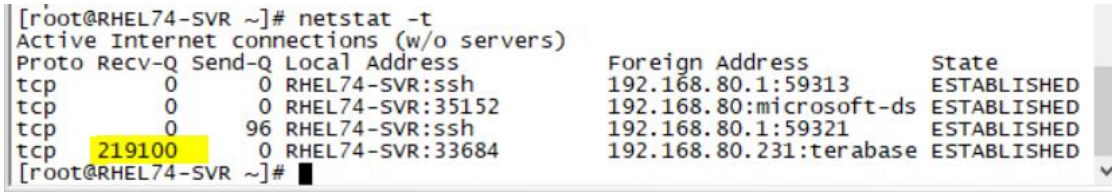




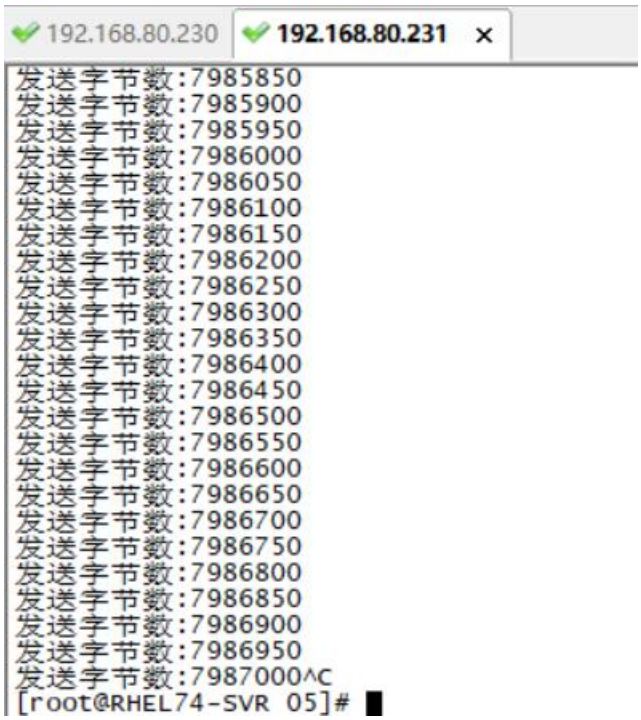
查看此时双方 Recv -Q,Send -Q 值：
(1)server 端：



(2)Client 端：



Getchar()之后打开阻塞再次查看情况：
Server 端发送情况：



Server 端 send -Q 情况均为 0：

```
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:terabase     192.168.80.230:33684    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:52052        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]#
```

Client 端 recv -Q 情况均为 0：

```
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:59313      ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:59321      ESTABLISHED
tcp        0      0 RHEL74-SVR:35166        192.168.80:microsoft-ds ESTABLISHED
tcp        0      0 RHEL74-SVR:33684        192.168.80.231:terabase ESTABLISHED
[root@RHEL74-SVR ~]#
```

5.测试程序 tcp_server3/tcp_client3，设置函数改变 TCP 收发缓冲区大小测试：

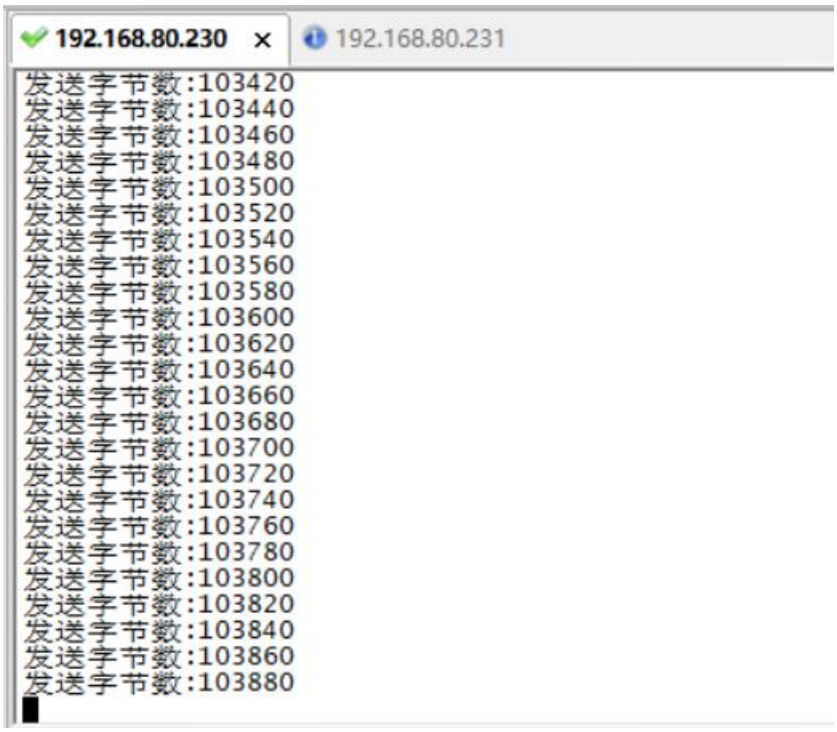
Server 端设置：设置为 32k

```
int nRecvBuf = 32 * 1024; //设置为32K
setsockopt(server_sockfd,SOL_SOCKET,SO_RCVBUF,(const char*)&nRecvBuf,sizeof(int));
```

client 端设置：设置为 32k

```
if(1)
{
    int nSendBuf = 32*1024; //设置为32K
    setsockopt(sock_cli,SOL_SOCKET,SO_SNDBUF,(const char*)&nSendBuf,sizeof(int));
}
///连接服务器，成功返回0，错误返回-1
```

连接情况，数据发送情况如下,到 103880 字节进入阻塞状态：




```
[root@RHEL74-SVR 05]# ./tcp_server5-3 4000
连接成功!
```

查看此时 client 端 Send -Q:

```
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address               Foreign Address             State
tcp        0      0 RHEL74-SVR:ssh             192.168.80.1:59313          ESTABLISHED
tcp        0  68056 RHEL74-SVR:33752           192.168.80.231:terabase     ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh             192.168.80.1:59321          ESTABLISHED
tcp        0      0 RHEL74-SVR:35214           192.168.80:microsoft-ds     ESTABLISHED
[root@RHEL74-SVR ~]#
```

查看此时 server 端 Recv -Q:

```
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address               Foreign Address             State
tcp        0      0 RHEL74-SVR:52096           192.168.80:microsoft-ds     ESTABLISHED
tcp    35836      0 RHEL74-SVR:terabase        192.168.80.230:33752        ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh             192.168.80.1:62923          ESTABLISHED
[root@RHEL74-SVR ~]#
```

Getchar()之后打开阻塞，数据一直发送不停，再次查看:
数据发送情况:

```
192.168.80.230  ×   192.168.80.231
发送字节数:3820420
发送字节数:3820440
发送字节数:3820460
发送字节数:3820480
发送字节数:3820500
发送字节数:3820520
发送字节数:3820540
发送字节数:3820560
发送字节数:3820580
发送字节数:3820600
发送字节数:3820620
发送字节数:3820640
发送字节数:3820660
发送字节数:3820680
发送字节数:3820700
发送字节数:3820720
发送字节数:3820740
发送字节数:3820760
发送字节数:3820780
发送字节数:3820800
发送字节数:3820820
发送字节数:3820840
发送字节数:3820860
^C
[root@RHEL74-SVR 05]#
```

查看此时 client 端 Send -Q，一直为 0：

```
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:59313      ESTABLISHED
tcp        0      0 RHEL74-SVR:35224        192.168.80:microsoft-ds ESTABLISHED
tcp        0      0 RHEL74-SVR:33752        192.168.80.231:terabase ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:59321      ESTABLISHED
[root@RHEL74-SVR ~]#
```

查看此时 server 端 Recv -Q，一直为 0：

```
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:terabase     192.168.80.230:33752    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:52106        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:terabase     192.168.80.230:33752    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:52106        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:terabase     192.168.80.230:33752    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:62923      ESTABLISHED
tcp        0      0 RHEL74-SVR:52106        192.168.80:microsoft-ds ESTABLISHED
[root@RHEL74-SVR ~]#
```

七.(06 目录)

1.测试程序 tcp_server6-1,连接成功后进入死循环，双方均进入 read 函数阻塞状态：

```
[root@RHEL74-SVR 06]# ./tcp_client6-1 192.168.80.231 4000 1000 1000
连接成功!
```

CRT

Transfer Script Tools Window Help

<Alt+R>

192.168.80.231 ×

```
g++ -o tcp_client6-3 client3.cpp
g++ -o tcp_client6-4 client4.cpp
[root@RHEL74-SVR 06]# ./server6-1 4000 1000 1000
-bash: ./server6-1: 没有那个文件或目录
[root@RHEL74-SVR 06]# ./tcp_server6-1 4000 1000 1000
连接成功!
```

并且 netstat -t 显示没有数据堆积，此时还未传送数据：

192.168.80.230 | 192.168.80.230 (1) | 192.168.80.231 ×

```
Last login: Wed Oct 24 10:52:06 2018 from 192.168.80.1
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:terabase     192.168.80.230:45670    ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:51057      ESTABLISHED
tcp        0      0 RHEL74-SVR:42558        192.168.80:microsoft-ds ESTABLISHED
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:50686      ESTABLISHED
[root@RHEL74-SVR ~]#
```

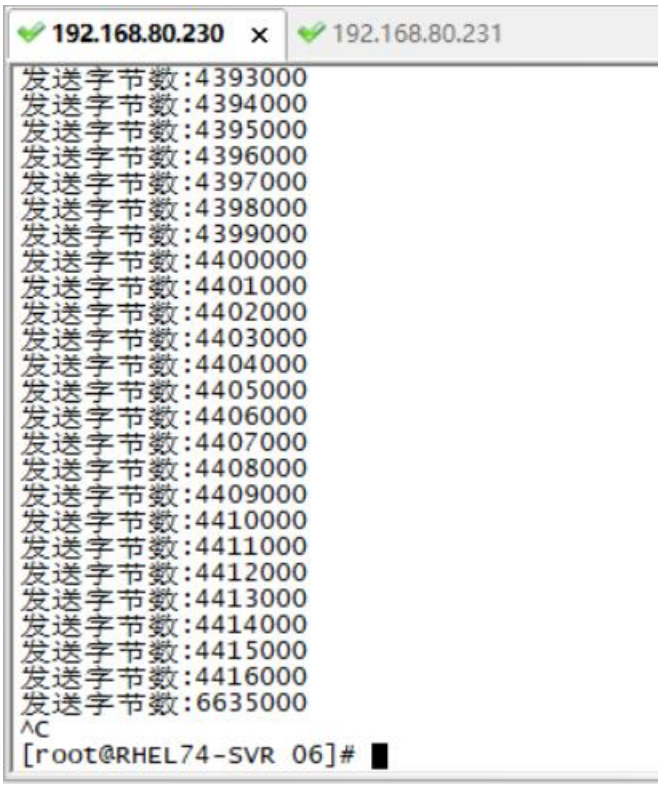
192.168.80.230 | 192.168.80.230 (1) × | 192.168.80.231

```
Last login: Wed Oct 24 10:31:08 2018 from 192.168.80.1
[root@RHEL74-SVR ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL74-SVR:ssh          192.168.80.1:50684      ESTABLISHED
tcp        0     96 RHEL74-SVR:ssh          192.168.80.1:51054      ESTABLISHED
tcp        0      0 RHEL74-SVR:56386        192.168.80:microsoft-ds ESTABLISHED
tcp        0      0 RHEL74-SVR:45670        192.168.80.231:terabase ESTABLISHED
[root@RHEL74-SVR ~]#
```

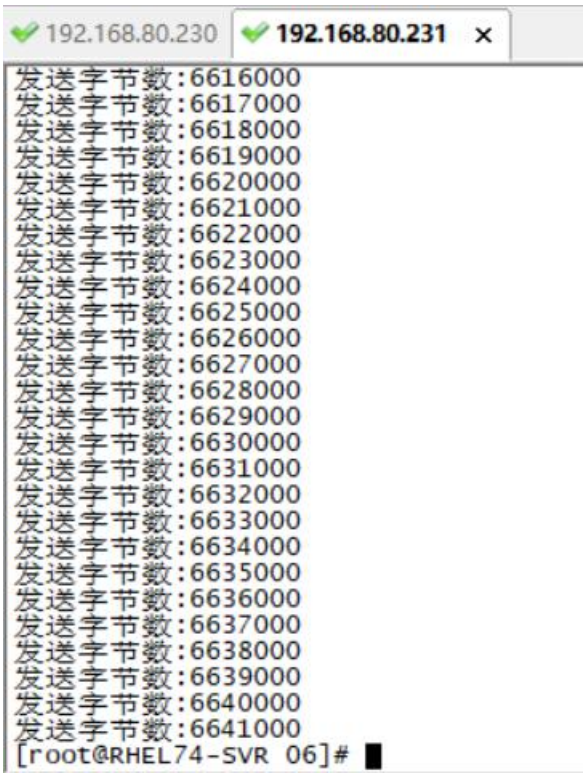
2.tcp_server6-2 测试，收发数据情况测试如下：

(1)双方每次均 read/write 1000 字节情况：

client 端情况：



server 端情况：

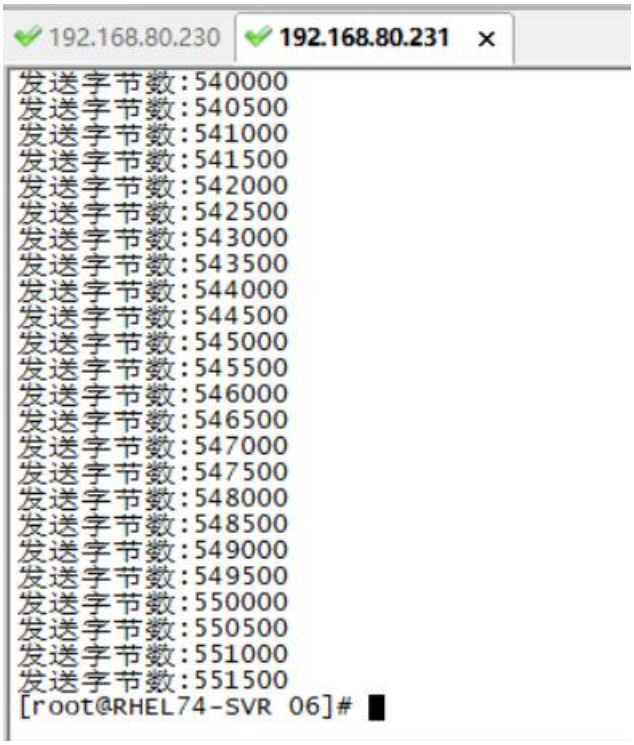


Server 端最终发送为 6641000，client 端为 6635000，双方收发消息字节数近似相等，可以正常收发数据

(2)server 端每次 read1000 字节，write500 字节，client 端每次 read500 字节，write 1000 字节测试：

Client 端收发数据情况：

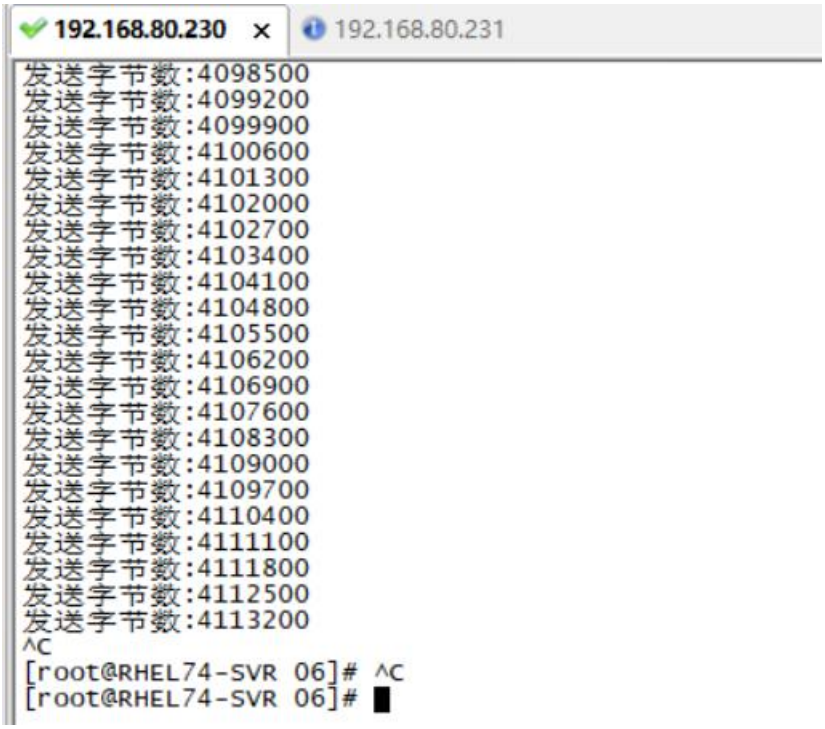
Server 端收发数据情况：



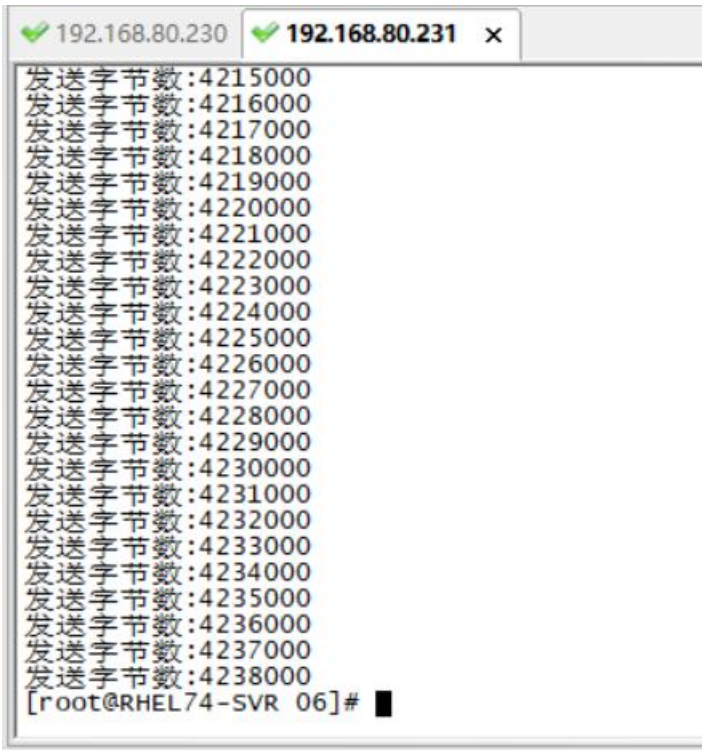
Client 端最终发送为 1102000，而 server 端只有 551500，双方可以正常收发数据但是 client 端发送的数据字节总数为 server 端的两倍

(3)server 端每次 read1000 字节，write1000 字节，client 端每次 read700 字节，write 700 字节测试：

client 端收发数据情况：



Server 端收发数据情况：



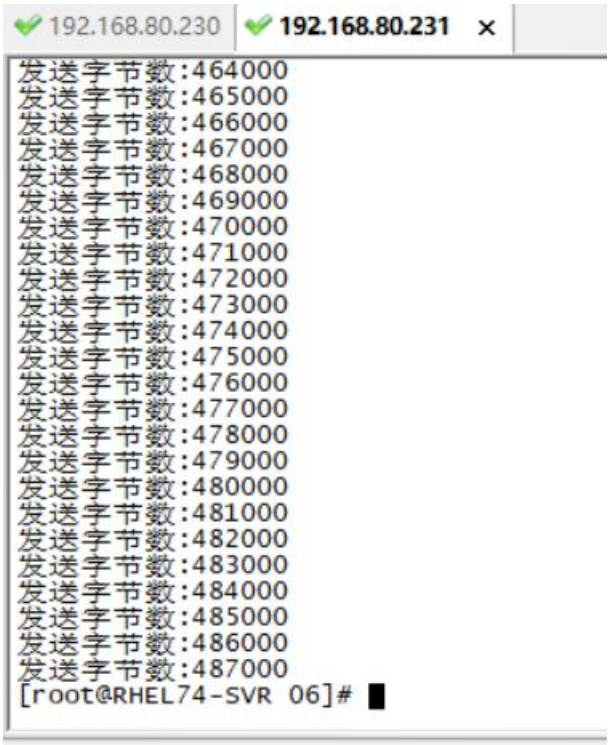
Client 端最终发送为 4113200 字节，而 server 端有 4238000 字节，双方收发消息字节数近似相等，可以正常收发数据

3.

(1)server 端每次 read1000 字节，write1000 字节，client 端每次 read1000 字节，write 1000 字节测试：

client 端收发数据情况：

Server 端收发数据情况：

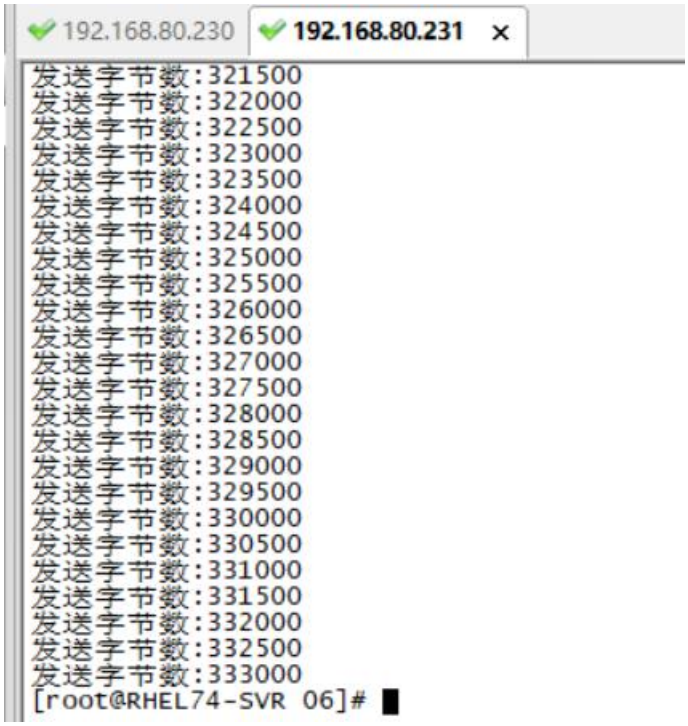


Server 端最终发送为 487000，client 端为 484000，双方收发消息字节数近似相等，可以正常收发数据

(2)server 端每次 read1000 字节，write500 字节，client 端每次 read500 字节，write 1000 字节测试：

client 端收发数据情况：

Server 端收发数据情况:

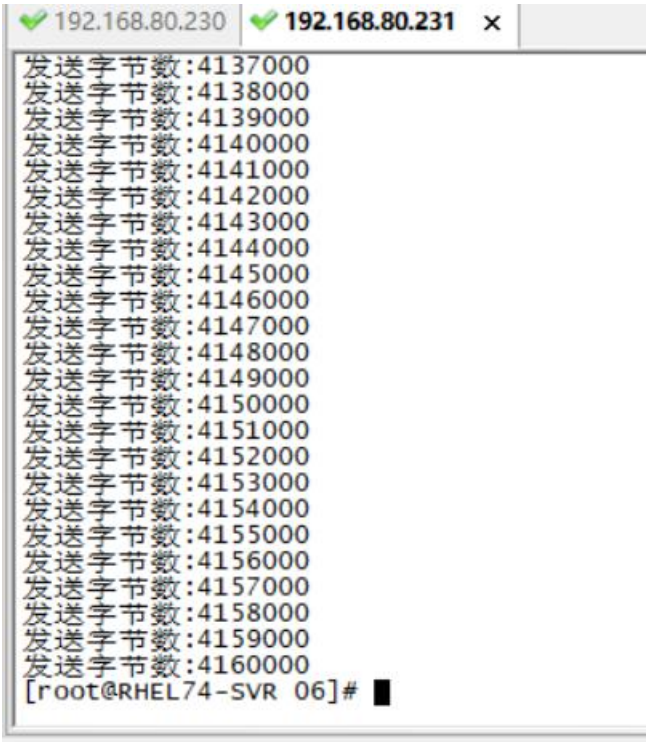


Client 端最终发送为 657000, 而 server 端只有 333000, 双方可以正常收发数据但是 client 端发送的数据字节总数为 server 端的两倍

(3)server 端每次 read1000 字节，write1000 字节，client 端每次 read700 字节，write 700 字节测试:

client 端收发数据情况：

Server 端收发数据情况：



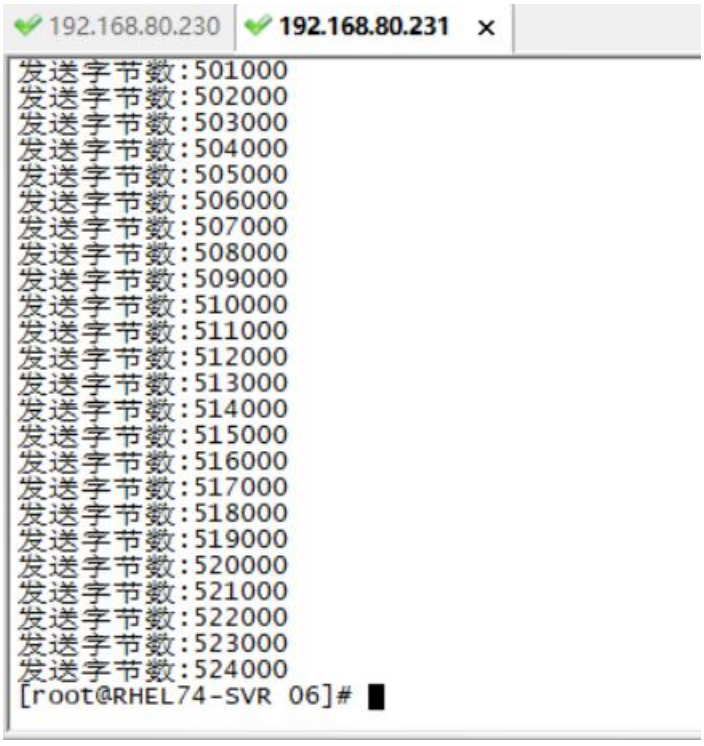
Client 端最终发送为 40040000 字节，而 server 端有 4160000 字节，双方收发消息字节数近似相等，可以正常收发数据

4.

(1)server 端每次 read1000 字节，write1000 字节，client 端每次 read1000 字节，write 1000 字节测试：

client 端收发数据情况：

Server 端收发数据情况：

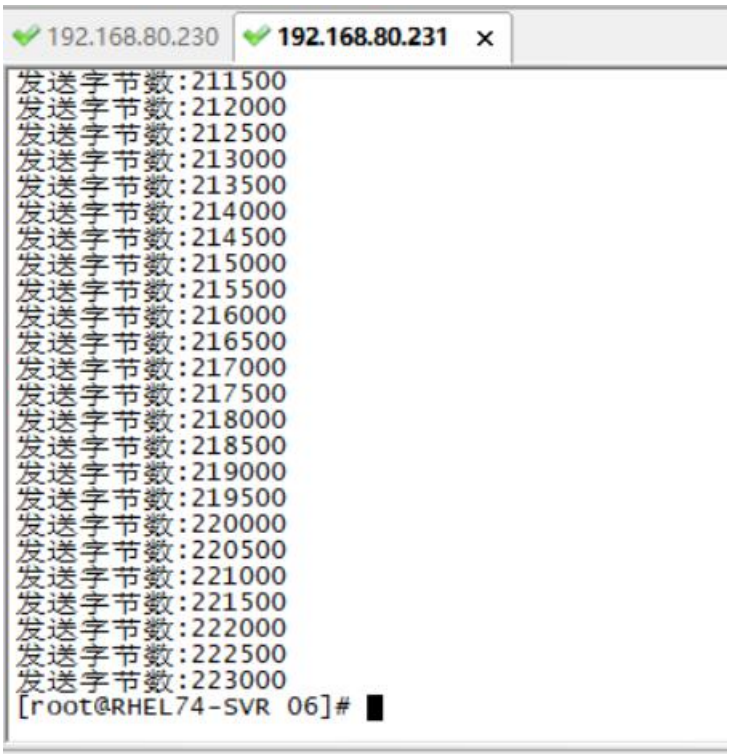


Server 端最终发送为 518000，client 端为 524000，双方收发消息字节数近似相等，可以正常收发数据

(2)server 端每次 read1000 字节，write500 字节，client 端每次 read500 字节，write 1000 字节测试：

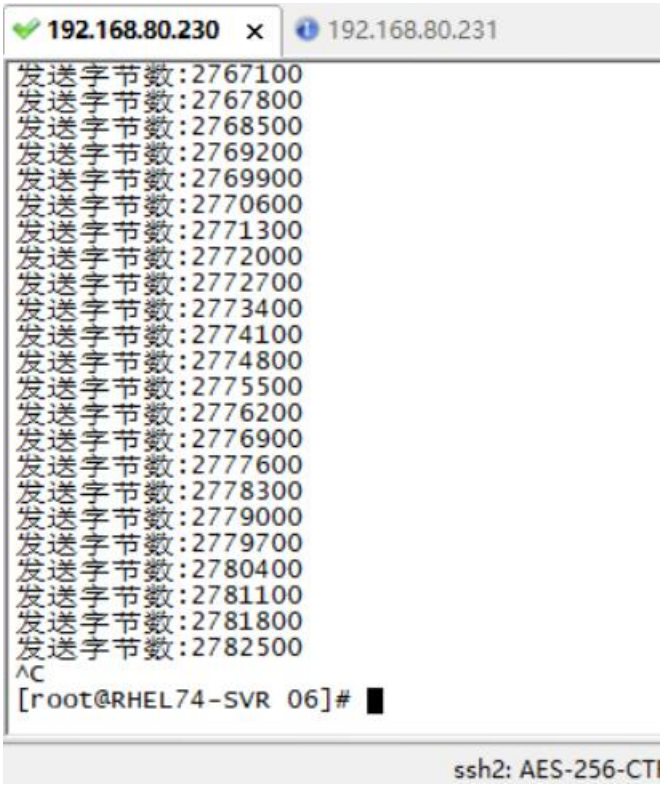
client 端收发数据情况：

Server 端收发数据情况：

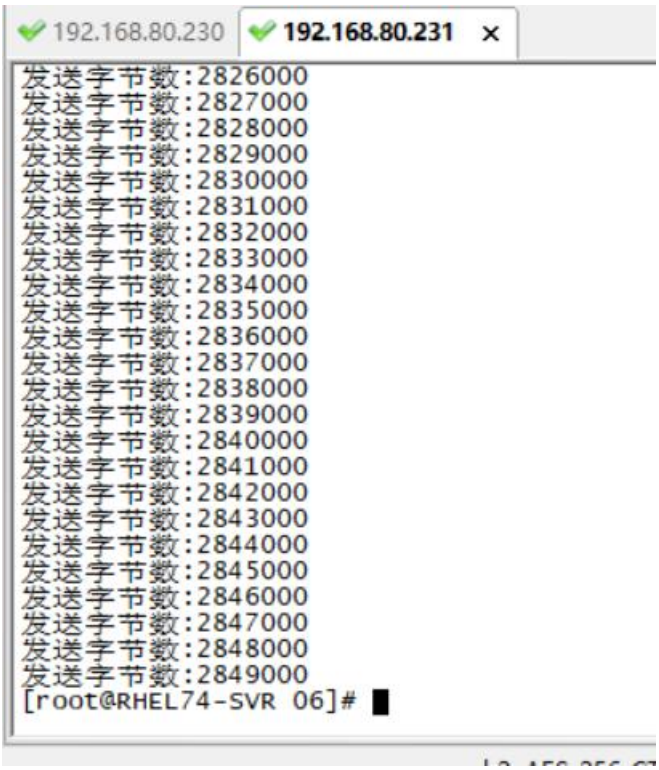


Client 端最终发送为 442000, 而 server 端只有 223000, 双方可以正常收发数据但是 client 端发送的数据字节总数为 server 端的两倍

(3)server 端每次 read1000 字节，write1000 字节，client 端每次 read700 字节，write 700 字节测试：
client 端收发数据情况：



Server 端收发数据情况：



Client 端最终发送为 2782500 字节，而 server 端有 2849000 字节，双方收发消息字节数近似相等，可以正常收发数据