同济大学计算机网络

实验报告



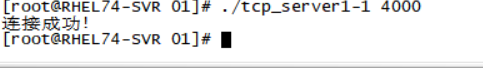
姓名： 涂远鹏-1652262

题目： socket编程非TCP阻塞方式

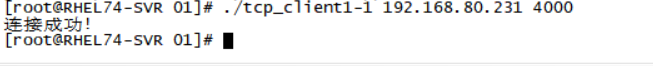
一.(01子目录)写一对TCP socket测试程序，分为client和server，分别运行在不同的虚拟机上

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：

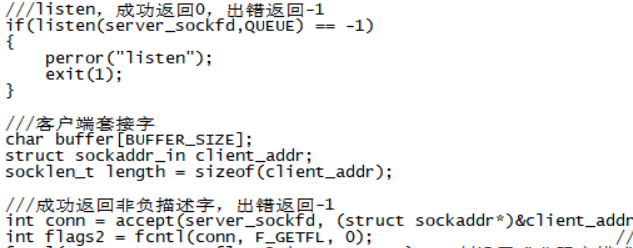


2.测试程序tcp\_client1-1运行时带入服务器IP地址和端口号：



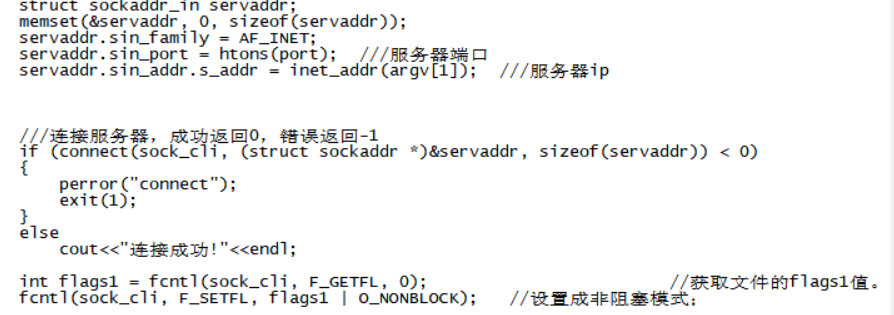
3.Server端用于listen的socket，不设置为非阻塞方式，accept成功后将accept的socket设置为非阻塞:

Listen的socket在accept成功后设置为非阻塞：

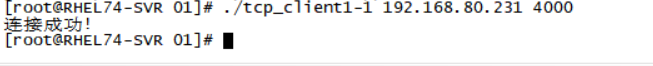


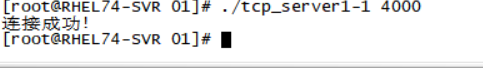
4.client端建立的socket,先不设置为非阻塞方式,accept成功后，将accept的socket设置为非阻塞:

代码如下,连接前不设置连接后设置为非阻塞：



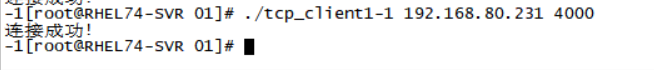
5.连接成功后,双方均进入read(recv)状态,read(recv)函数后直接关闭socket，程序退出:





连接成功后直接退出

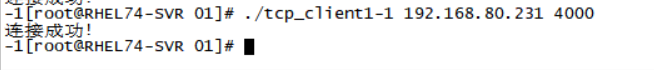
为了查看双方是否已进入read/recv状态，输出返回值为-1：





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?

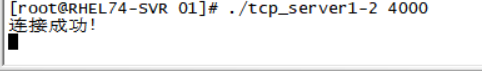
会立即结束，read(recv)函数输出返回值为-1：





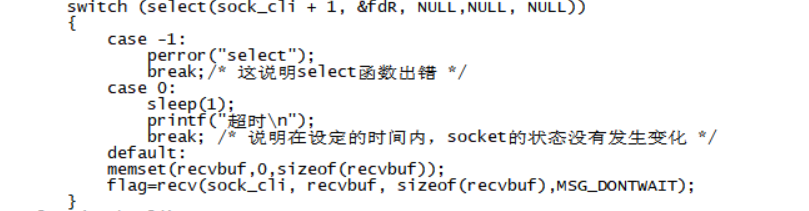
7.测试程序tcp\_server1-2/tcp\_client1-2在1-1基础上，用select使read(recv)停下来而不立即返回:

测试结果如下,双方不立即返回：

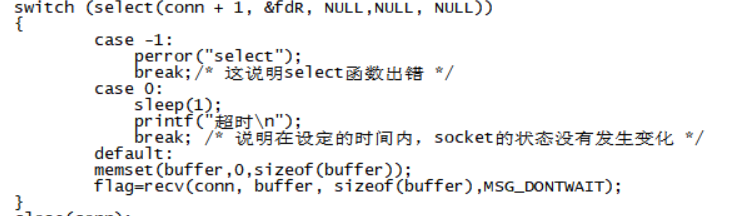
8

代码如下,用switch结构控制select的输出：

Tcp\_client1-2.cpp：

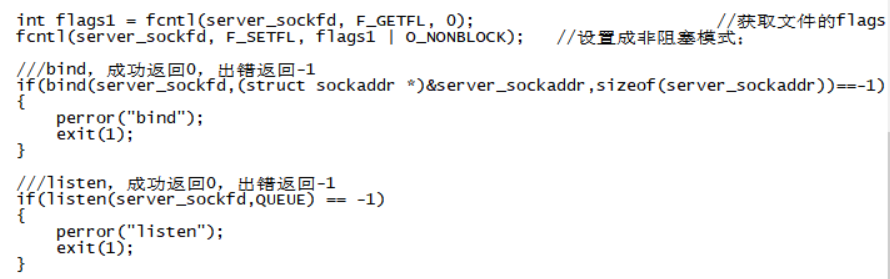


Tcp\_server1-2.cpp:



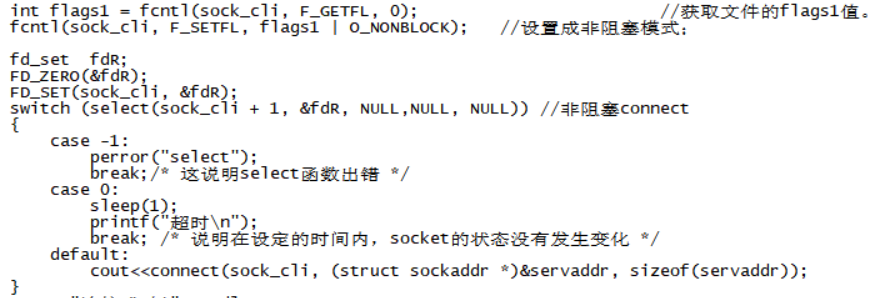
8.测试程序tcp\_server1-3要求socket建立后先设置为非阻塞再进行bind,listen和accept,accept的新的socket：

Tcp\_server1-3设置如下,先设置非阻塞然后进行bind和listen：



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：



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:

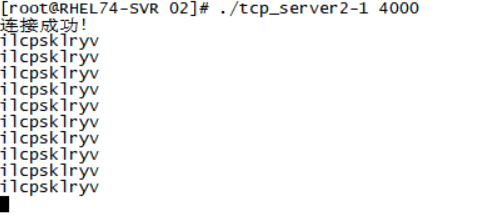
测试结果如下：

12

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:

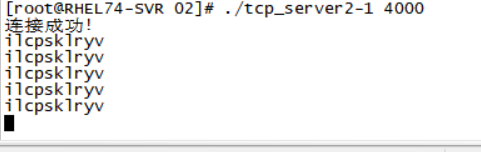
测试显示如下：



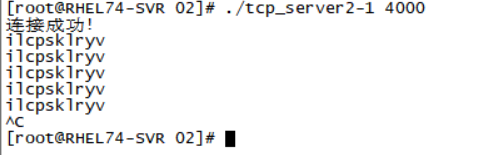
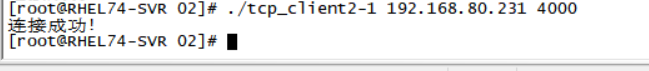


2.此时在client(server)端CTRL+C，server(client)端能否检测到连接中断？

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

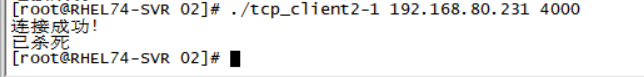
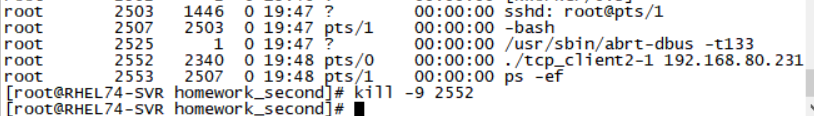
10

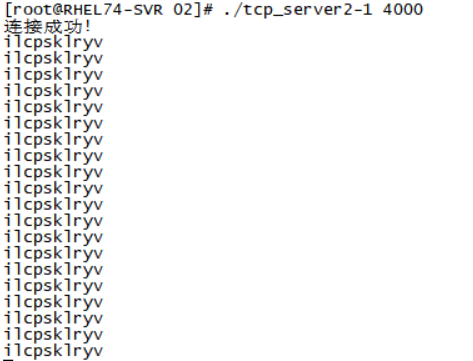
Server端CTRL+C观察client端结果,server端中断以后client端可以检测到中断，自动退出：



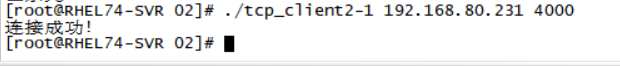
3.如果新开一个会话窗口,kill -9杀client(server)端程序，server(client)端能否检测到连接中断?

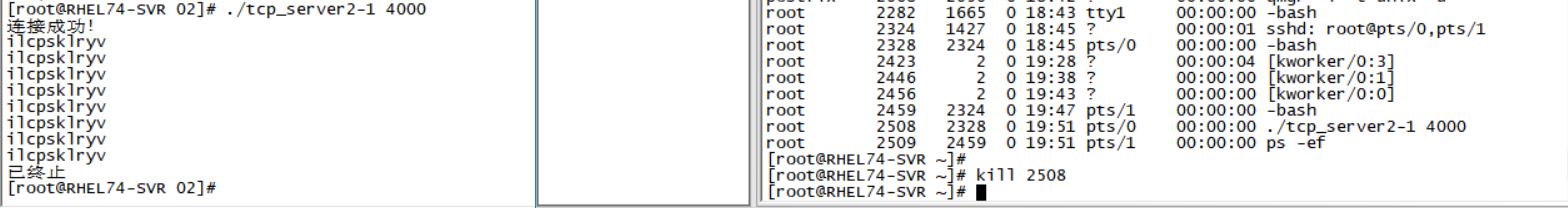
新建窗口杀死client端，server端情况如下，检测不到连接已中断：





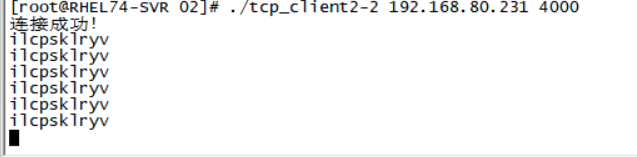
新建窗口杀死server端，client端情况如下，可以检测到连接已中断，自动退出：



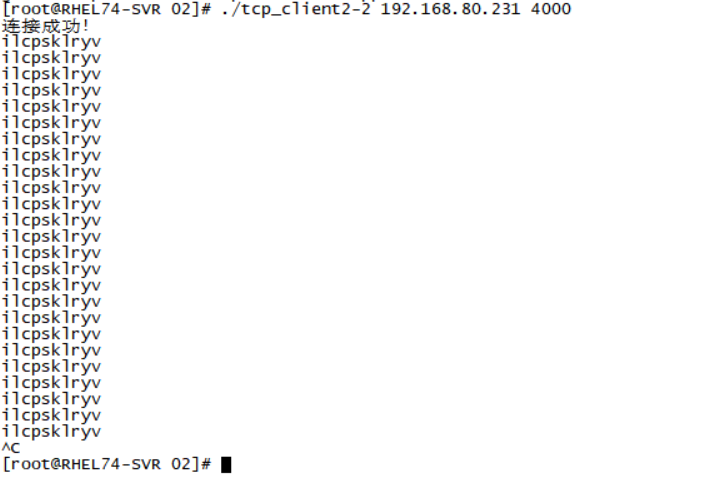


4.测试程序tcp\_server2-2/tcp\_client2-2,server发数据(每次10字节，间隔1秒)，client用大小100的缓冲区收数据，死循环进行:

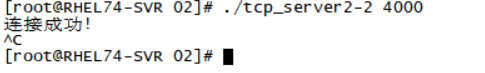
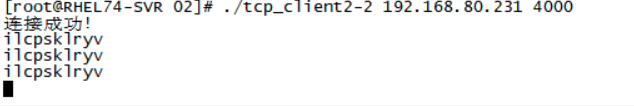
首先展示传输数据测试的结果：

2

Client用CTRL+C中断server端可以发现连接中断：

4

Server用CTRL+C中断client端不能发现连接中断：

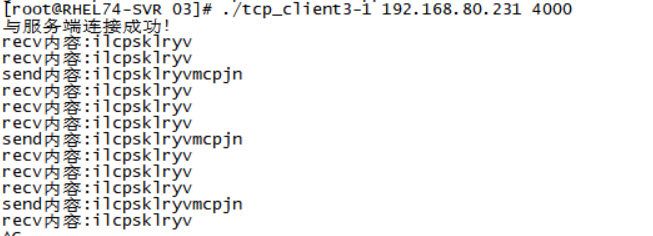


三.(03子目录)写一对TCP socket的测试程序，分为client和server，分别运行在不同的虚拟机上

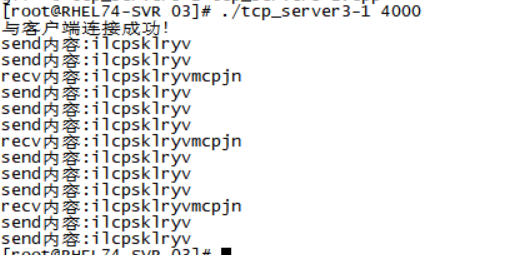
1.测试程序tcp\_server3-1/tcp\_client3-1,server发数据(每次10字节,间隔1秒)并用大小为100的缓冲区收数据，client发数据(每次15字节,间隔3秒)并同时用大小为100的缓冲区收数据，死循环进行:

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：

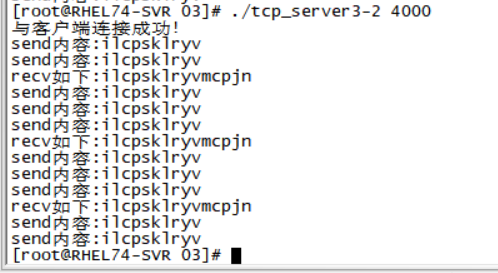
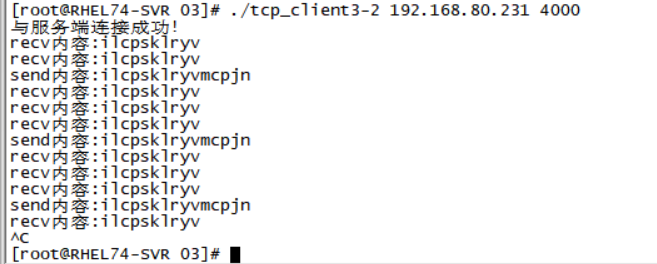


Server端测试结果：



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...：



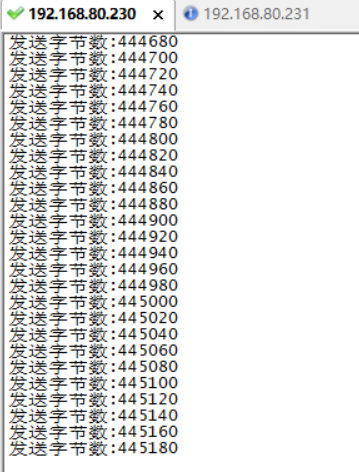
四.(04子目录)写一对TCP socket的测试程序，分为client和server，分别运行在不同的虚拟机上

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:

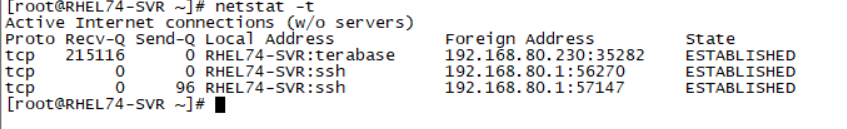
测试结果如下，服务端连接成功后等待运行:



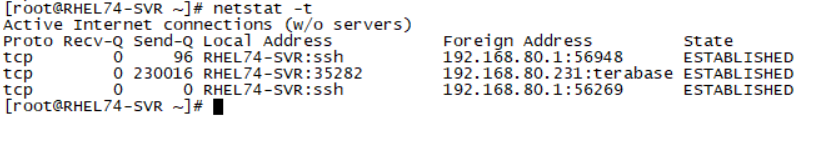
2.测试程序tcp\_client4-1,用write向服务器不断写入，直到write失败为止,用netstat观察读写队列:



Server端netstat -t结果：

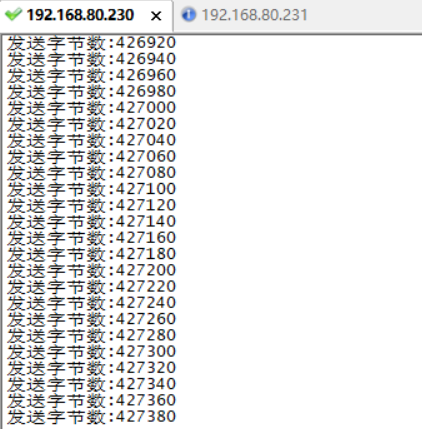


Client端netstat -t结果：

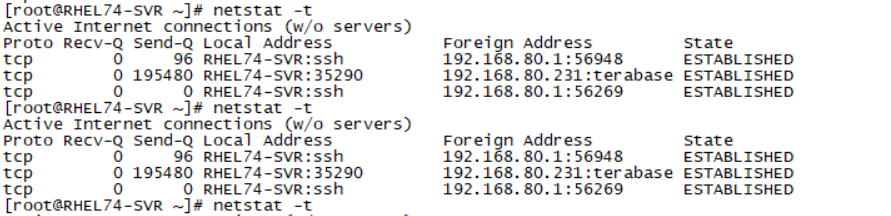


3.测试程序tcp\_server4-2，接受client的连接成功后，用每读20字节就延时一秒的方法循环读数据:

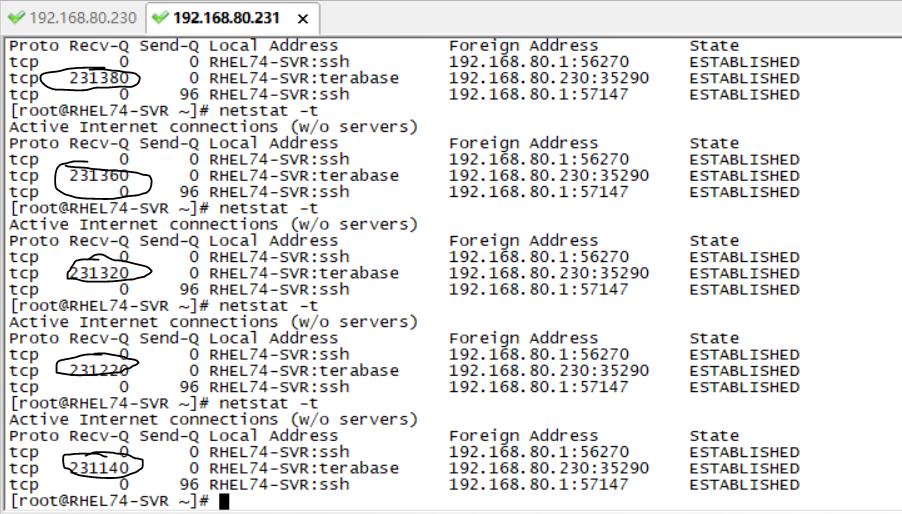
测试结果如下：



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)：



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：



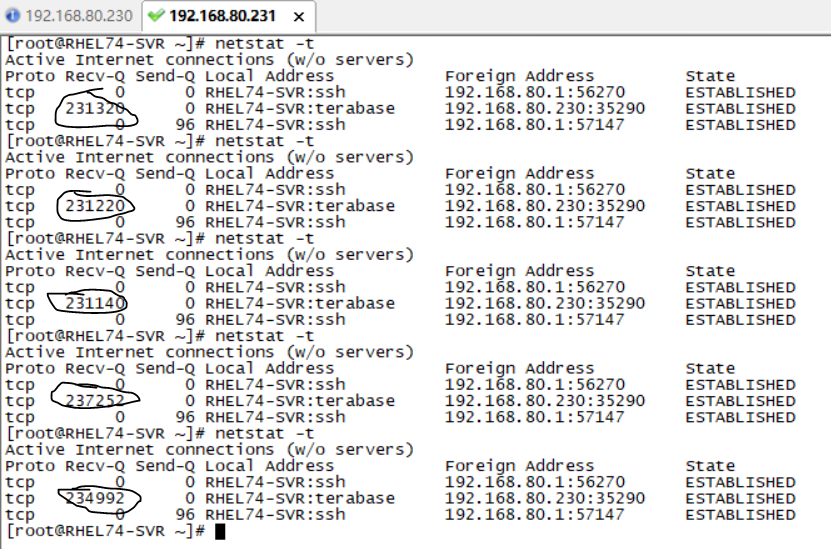
4.测试程序tcp\_client4-2，连接服务端成功后,用write向服务端大量写入,直到write失败为止,用netstat观察读写队列,write失败后如何恢复为写状态?

答：

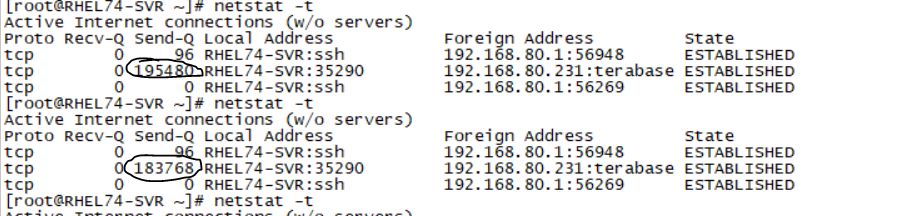
server 端的 Recv-Q 在Recv -Q和Send -Q均满了之后会出现先减后增的情况，增到一个缓冲区上限左右又开始慢慢减少，在 Recv-Q 减少的过程中 Send-Q基本不变， Recv-Q数据量增加时Send -Q减少。

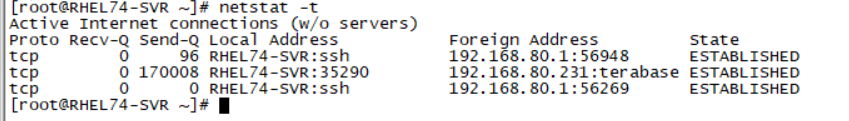
解释：Server 的读缓冲区因读取一点点减少，减少到一定量时 client端 的写缓冲区将一部分数据移至 server 读缓冲区将其填满，此时 client不会马上执行 write，测试过程发生这样的移动发生几次后 client的 write 会恢复写入状态，很快又将自己的写缓冲区填满，然后这样的循环重新执行，测试的结果如下：

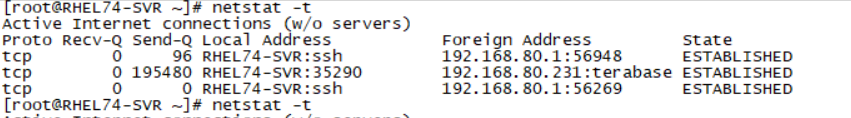
(1)server 端的recv -Q读缓冲区先减后增的情况如下：



(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

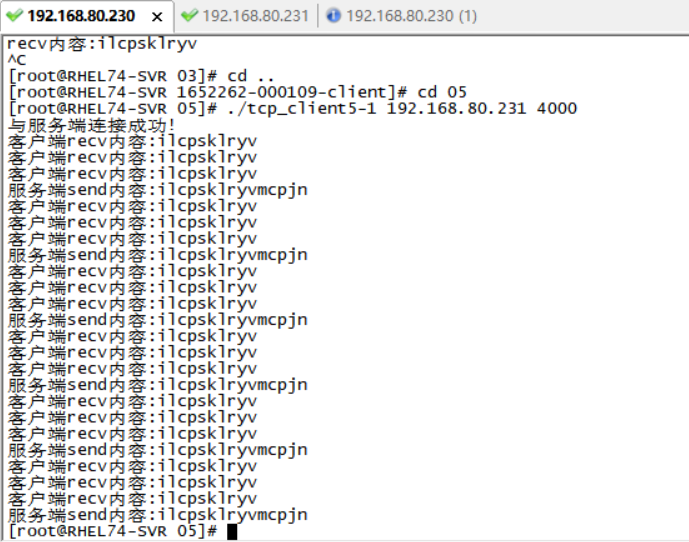
1.测试程序tcp\_server5，接受连接成功后，server发数据(每次10字节，间隔1秒)并同时用大小100的缓冲区收数据，死循环运行测试程序tcp\_client5-1，连接成功后，client发数据(每次15字节，间隔3秒)并同时用大小100的缓冲区收数据，死循环运行,server端先接受一个client的连接，进入死循环读写状态,要求此时server能接受一个新的client端的连接，也进入死循环读写状态:用两个会话窗口分别启动两个tcp\_server5:

Tcp\_server5-1与tcp\_client5-1测试结果如下：

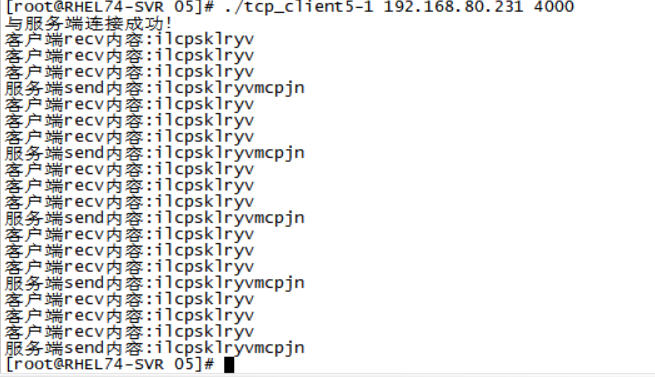
Server端：



第一个client端连接：



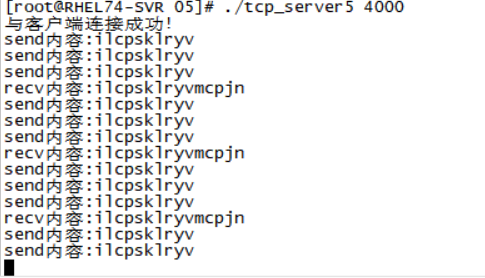
第二个client端连接：



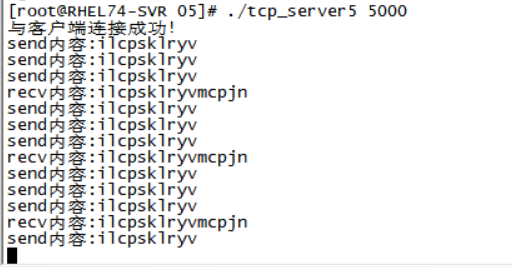
2.测试程序tcp\_client5-2,运行时带两个端口，建立两个socket,分别连接两个server端，client发数据(每次15字节，每隔3秒)并用大小100缓冲区收数据，死循环进行：

Tcp\_server5-2与tcp\_client5-2测试结果如下：

第一个Server端，端口为4000：



第二个server端连接，端口为5000：



client端连接：

