


Socket 编程

UDP 阻塞及非阻塞方式

1551265 张伯阳



1

client 和 server 连接后的运行结果

```
[root@RHEL-zby test]# ./udp_server1-1 4000
连接成功
[root@RHEL-zby test]# ./udp_client1-1 192.168.2.231 4000
连接成功
```

如果服务端绑定的 UDP 端口号已被使用,tcp 会出错 显示 address already in use

```
[root@RHEL-zby 01]# ./tcp_server1-1 80
bind: Address already in use
```

udp 不会出错 且能正常连接成功

```
[root@RHEL-zby test]# ./udp_server1-1 80
连接成功
[root@RHEL-zby test]# ./udp_client1-1 192.168.2.231 80
连接成功
```

测试程序 udp\_client1, 运行时带入服务端 IP 地址及端口号, 在发送数据前, 不需要向服务端发起连接

正常的 udp 直接向指定端口发送数据无法确认是否连接成功

如果我们一定要两端确认连接成功 可以在 server 程序中利用一个 while(1)循环进行等待 当 client 端向其发出一个信号后 表示连接成功并回复信息确认 server 连接成功 让 client 得知连接成功

```
//keep listening for data
while(1)
{
    if (recvfrom(s, buf, BUFLen, 0, (struct sockaddr *) &si_other, &slen) > 0)
    {
        sendto(s, buf, strlen(buf), 0, (struct sockaddr *) &si_other, slen);
        cout<<"连接成功"<<endl;
        break;
    }
}
```

client 程序中加入 bind 后的向 server 端发送信号 当收到回复信号后 表示连接成功

```

si_other.sin_addr.s_addr = inet_addr(argv[1]);
while(1)
{
    if (sendto(s, message, strlen(message), 0, (struct sockaddr *) &si_other, slen)==-1)
    {
        exit(-1);
    }
    if (recvfrom(s, buf, BUFLen, 0, (struct sockaddr *) &si_other, &slent) > 0)
    {
        cout<<"连接成功"<<endl;
        break;
    }
}
}

```

如果 client 端连接时的 IP 地址不正确或端口号不正确都不会出错  
 正常的 udp 中 client 会直接向指定的错误 ip 或端口正常发包 只是没有 server 收包 导致数据包的丢失  
 以为我在这道题加入了连接判断 只有确定两端成功连接才会发包  
 所以如果给了错误的端口号或 ip 则会卡在 while(1)循环中 持续向指定的 ip 地址或端口发送连接请求 但是无法得到回应

两边都用 recvfrom 两端都进入阻塞状态  
 此时用 kill -9 杀死 另一端无法检测到

```

[root@RHEL-zby test]# ./udp_client1 192.168.2.231 4000
连接成功
已杀死
[root@RHEL-zby test]#

```

```

[root@RHEL-zby test]# ./udp_server1 4000
连接成功

```

udp\_server1 运行终止后，立即再次启动，绑定相同端口号，可以成功

2

client 和 server 连接后的运行结果

```
[root@RHEL-zby test]# ./udp_server2 4000
连接成功
█

[root@RHEL-zby test]# ./udp_client2 192.168.2.231 4000
连接成功
█
```

如果服务端绑定的 UDP 端口号已被使用

udp 不会出错 且能正常连接成功

测试程序 udp\_client1, 运行时带入服务端 IP 地址及端口号, 在发送数据前, 不需要向服务端发起连接

正常的 udp 直接向指定端口发送数据无法确认是否连接成功

如果我们一定要两端确认连接成功 可以在 server 程序中利用一个 while(1)循环进行等待 当 client 端向其发出一个信号后 表示连接成功并回复信息确认 server 连接成功 让 client 得知连接成功

client 程序中加入 bind 后的向 server 端发送信号 当收到回复信号后 表示连接成功

如果 client 端连接时的 IP 地址不正确或端口号不正确都不会出错

正常的 udp 中 client 会直接向指定的错误 ip 或端口正常发包 只是没有 server 收包 导致数据包的丢失

以为我在这道题加入了连接判断 只有确定两端成功连接才会发包

所以如果给了错误的端口号或 ip 则会卡在 while(1)循环中 持续向指定的 ip 地址或端口发送连接请求 但是无法得到回应

两边都用 recvfrom 两端都进入阻塞状态

此时用 kill -9 杀死 另一端无法检测到

udp\_server2 运行终止后, 立即再次启动, 绑定相同端口号, 可以成功

3-1

client 一次性写入 30 字节 server 一次最多只能接受 20 字节 只一次性接受收到前 20 字节

```
[root@RHEL-zby test]# ./udp_server3 4000
连接成功
recv_len:20
recv:01234567890123456789
[root@RHEL-zby test]#

[root@RHEL-zby test]# ./udp_client3-1 192.168.2.231 4000
连接成功
send:012345678901234567890123456789
[root@RHEL-zby test]#
```

3-2

client 分次写入 每次 2 字节

server 端一次性读入 只读到了 2 字节就退出了

```
[root@RHEL-zby test]# ./udp_server3 4000
连接成功
0
recv_len:2
recv:01
[root@RHEL-zby test]#

[root@RHEL-zby test]# ./udp_client3-2 192.168.2.231 4000
连接成功
send:01
send:01
send:01
send:01
```

4-1

初次运行结果

```
[root@RHEL-zby test]# ./udp_server4 4000
连接成功
recv_len:20
@`?
[root@RHEL-zby test]#
```

```
[root@RHEL-zby test]# ./udp_client4-1 192.168.2.231 4000
连接成功
send:012345678901234567890123456789
[root@RHEL-zby test]#
```

输出的并不是我想要的结果

修改代码将 char 的值输出

```
[root@RHEL-zby test]# ./udp_server4 4000
连接成功
recv_len:20
-40 29 96 0 0 0 0 0 -43 13 64 0 0 0 0 0 0 0 0 0
```

for 循环把所有收到的都输出

可以注意到最后输出了正确结果 前面的乱七八糟的数应该没有 memset

```

[root@RHEL-zby test]# ./udp_server4 4000
连接成功
recv_len:20
-40 29 96 0 0 0 0 0 -43 13 64 0 0 0 0 0 0 0 0 0
recv_len:20
-40 29 96 0 0 0 0 0 -43 13 64 0 0 0 0 0 0 0 0 0
recv_len:20
-40 29 96 0 0 0 0 0 -43 13 64 0 0 0 0 0 0 0 0 0
recv_len:20
-40 29 96 0 0 0 0 0 -43 13 64 0 0 0 0 0 0 0 0 0
recv_len:20
-40 29 96 0 0 0 0 0 -43 13 64 0 0 0 0 0 0 0 0 0
recv_len:20
-40 29 96 0 0 0 0 0 -43 13 64 0 0 0 0 0 0 0 0 0
recv_len:20
-40 29 96 0 0 0 0 0 -43 13 64 0 0 0 0 0 0 0 0 0
recv_len:20
-40 29 96 0 0 0 0 0 -43 13 64 0 0 0 0 0 0 0 0 0
48 49 50 51 52 53 54 55 56 57 48 49 50 51 52 53 54 55 56 57
^C

```

memset 后再输出 前面的输出为全 0(不再截图)

可是还是要考虑如何一次输出想要的结果

修改代码 在输出前加一个判定如果接受全 0 的字节流则不输出

```

le(1)
if(select(s + 1, &fdR, NULL, NULL, NULL))
{
    recv_len = recvfrom(s, buf, BUFLen, 0, (struct sockaddr *) &si_other, &slen);
    int flag_zero=0;
    for(i=0;i<BUFLen;i++)
        if(buf[i])
        {
            flag_zero=1;
            break;
        }
    if(flag_zero)
    {
        cout<<"recv_len:"<<recv_len<<endl;
        for(i=0;i<BUFLen;i++)
            cout<<buf[i];
        cout<<endl;
    }
}
}

```

再次输出显示正常

```

[root@RHEL-zby test]# ./udp_server4 4000
连接成功
recv_len:20
01234567890123456789
[root@RHEL-zby test]#

[root@RHEL-zby test]# ./udp_client4-1 192.168.2.231 4000
连接成功
send:012345678901234567890123456789
[root@RHEL-zby test]#

```

UDP 下的读函数能否做到必须读满指定字节数才退出

udp 中使用改变 recv 的最后一个参数为 waitall 不起作用

使用 tcp 中设置低水位的方式也不起作用

在查了很久的资料后 在阻塞状态使用 recvmsg 可达到预期效果

但是非阻塞状态 recvmsg 不起作用 应该就是不能做到吧

4-2

这里也遇到了和之前一样的问题

```

[root@RHEL-zby test]# ./udp_server4 4000
连接成功
recv_len:2

[root@RHEL-zby test]#

[root@RHEL-zby test]# ./udp_client4-2 192.168.2.231 4000
连接成功
send=01
send=01
send=01

```

还是一样的办法

修改代码 在输出前加一个判定如果接受全 0 的字节流则不输出



```

le(1)

if(select(s + 1, &fdR, NULL, NULL, NULL))
{
    recv_len = recvfrom(s, buf, BUFLen, 0, (struct sockaddr *) &si_other, &slen);
    int flag_zero=0;
    for(i=0;i<BUFLen;i++)
        if(buf[i])
        {
            flag_zero=1;
            break;
        }
    if(flag_zero)
    {
        cout<<"recv_len:"<<recv_len<<endl;
        for(i=0;i<BUFLen;i++)
            cout<<buf[i];
        cout<<endl;
    }
}
}

```

结果符合预期

```

[root@RHEL-zby test]# ./udp_server4 4000
连接成功
recv_len:2
01
[root@RHEL-zby test]#

[root@RHEL-zby test]# ./udp_client4-2 192.168.2.231 4000
连接成功
send=01
send=01

```

不能做到读满指定字节输出

```
recv1_len:20
01234567890123456789
send=01234567890123456789
recv2_len:20
01234567890123456789
recv1_len:20
01234567890123456789
send=01234567890123456789
recv2_len:20
01234567890123456789
recv1_len:20
01234567890123456789
send=01234567890123456789
recv2_len:20
01234567890123456789
recv1_len:20
01234567890123456789
^C
[root@RHEL-zby test]#
```

## 7.其他

UDP 的 Server 端能否在本机的多个 IP 地址中只选择其中的某一个绑定？如何做到？

把图示框中内容改为 IP 地址

```
si_me.sin_family = AF_INET;  
si_me.sin_port = htons(port);  
si_me.sin_addr.s_addr = htonl(INADDR_ANY);
```

两个 UDP Server 之间能否收发数据？

可以

在 bind 后加上这样的代码 将对方的 IP 地址和端口号绑定

另一个 server 端代码不变

```
//keep listening for data  
  
memset((char *) &si_other, 0, sizeof(si_other));  
si_other.sin_family = AF_INET;  
si_other.sin_port = htons(4000);  
si_other.sin_addr.s_addr = inet_addr("192.168.2.231");  
while(1)  
{  
    if (sendto(s, message, sizeof(message), 0, (struct sockaddr *) &si_other, slen)==-1)  
    {  
        exit(-1);  
    }  
    if (recvfrom(s, buf, BUFLen, 0, (struct sockaddr *) &si_other, &slenn) > 0)  
    {  
        cout<<"连接成功"<<endl;  
        break;  
    }  
}  
for(i=0;i<BUFLen;i++)  
    message[i]=i%10+'0';  
int send_len=sendto(s, message, sizeof(message), 0, (struct sockaddr *) &si_other, slenn);  
cout<<"send:";  
for(i=0;i<BUFLen;i++)  
    cout<<message[i];  
cout<<endl;  
return 0;
```

```

recv:01234567890123456789
[root@RHEL-zby test]# ./udp_server3 4000
连接成功
0
recv_len:20
recv:01234567890123456789
[root@RHEL-zby test]# █

[root@RHEL-zby test]# ./udp_server_test 4000
连接成功
send:01234567890123456789
[root@RHEL-zby test]# █

```

UDP 的 Client 端能否绑定端口号？

可以 之前的每个程序都用了 bind 不再贴实验结果图

```

// zero out the structure
memset((char *) &si_me, 0, sizeof(si_me));

si_me.sin_family = AF_INET;
si_me.sin_port = htons(port);
si_me.sin_addr.s_addr = htonl(INADDR_ANY);

//bind socket to port
if( bind(s , (struct sockaddr*)&si_me, sizeof(si_me) ) == -1)
{
    exit(-1);
}

```

UDP 绑定端口号，是否能用 netstat 看到？

可以

```

[root@RHEL-zby ~]# netstat -anp | grep udp
udp      0      0 0.0.0.0:4000          0.0.0.0:*           3992/./udp_server3
udp      0      0 0.127.0.0.1:323      0.0.0.0:*           1013/chronyd
udp6     0      0 :::1:323             :::*                 1013/chronyd
[root@RHEL-zby ~]# █

```

231

UDP 的 Client 端是否一定需要 connect 服务端？使用/不使用 connect 的情况下，Client 和 Server 间收发数据有什么不同？

不一定。在使用 connect 的情况下，socket 与已连接的 IP 和端口进行通信，而不是在发送数据时再指定 IP 和端口，写到 UDP 的缓冲区里的数据，将自动发送到调用 connect 指定的 IP 和端口，同时，远端地址不是该 socket 早先 connect 到的协议地址的数据，不会投递到该套接字。在不使用 connect 的情况下，client 端在用 sendto 发送数据时指定 IP 和端口，同时可以接收传送到自己地址的数据。

UDP 一端 getchar 暂停；另一端持续写，能否把缓冲区填满？阻塞/非阻塞有什么区别？

答：不能 没有区别

```
while(1)
{
    int send_len=sendto(s, message, sizeof(message) , 0 , (struct sockaddr *) &si_other, slen);
    cout<<"send_size:";
    send_size+=send_len;
    cout<<send_size<<endl;
    // sleep(1);
}
return 0;
```

```
send_size:14727920
send_size:14727940
send_size:14727960
send_size:14727980
send_size:14728000
send_size:14728020
send_size:14728040
send_size:14728060
send_size:14728080
send_size:14728100
```

```
Last login: Sun Nov 12 19:17:24 2017 from 192.168.2.1
[root@RHEL-zby ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 RHEL-zby:ssh           192.168.2.1:53675      ESTABLISHED
tcp        0      0 RHEL-zby:ssh           192.168.2.1:51616      ESTABLISHED
tcp        0      0 RHEL-zby:ssh           192.168.2.1:53563      ESTABLISHED
[root@RHEL-zby ~]#
```

非阻塞输出结果完全一样 不再截图

UDP 的收发缓冲区是否可设置？如何设置？  
可以设置。

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

```
int nSendBuf = 32*1024; //设置为32K
setsockopt(sock_cli, SOL_SOCKET, SO_SNDBUF, (const char*)&nSendBuf, sizeof(int));
```