<https://blog.csdn.net/qq_14821541/article/details/52021634>

## 链接地址1. TCP发送接口：send()

        TCP发送数据的接口有send，write，sendmsg。在系统内核中这些函数有一个统一的入口，即sock\_sendmsg()。由于TCP是可靠传输，所以对TCP的发送接口很容易产生误解，比如sn = send(...); 错误的认为sn的值是表示有sn个字节的数据已经发送到了接收端。其实真相并非如此。

        我们知道，TCP的发送和接收在在内核(linux)中是有对应的缓冲的：

**[cpp]** view plain copy

1. **struct** sock{
2. ...
3. **struct** sk\_buff\_head receive\_queue;  //接收的数据报队列
4. **struct** sk\_buff\_head write\_queue;    //即将发送的数据报队列
5. ...
6. }

|  |  |
| --- | --- |
| 1 | struct sock{ |
| 2 | ... |

|  |  |
| --- | --- |
| 3 | struct sk\_buff\_head receive\_queue;  //接收的数据报队列 |
| 4 | struct sk\_buff\_head write\_queue;    //即将发送的数据报队列 |

|  |  |
| --- | --- |
| 5 | ... |
| 6 | } |

        对于发送端而言，用户空间调用send(data)等发送接口将数据发送，内核会将data拷贝到内核空间的socket对应的缓冲中，即sock.write\_queue。而send()函数的返回值仅仅是表示本次send()调用中成功拷贝的字节数（用户空间->内核空间对应的sock缓冲队列）。具体发送和接收端的接收就由TCP协议完成，虽然TCP是可靠传输，但是这个前提是发送端和接收端的网络是连接的，否则你懂得。这样，对于调用send()发送的用户而言，如果想要确定接收方是否成功接受数据，就得需要靠其他的办法查询。

## 链接地址2. send()阻塞和非阻塞

前面已说过，sn=send(data)负责将数据拷贝到内核空间，而具体发送是有tcp后续完成（这里面就包括流量控制等）。对已阻塞和非阻塞仅仅是对于数据从用空空间拷贝到内核空间的发送缓冲这段期间而言的。

        1. 如果发送缓冲剩余空间大于欲发送data的字节数，对于阻塞和非阻塞而言都能成功将数据拷贝至内核缓冲，返回的sn即等于欲发送的数据的字节数。

        2.如果发送缓冲剩余空间小于欲发送data的字节数，

                    a. 那么阻塞模式会进行阻塞，等待内核缓冲的空闲空间，然后继续将数据拷贝到内核缓冲中，直到全部拷贝为止。返回rn即为data的数据量。

                    b. 非阻塞模式，当内核发送缓冲被填满后，立即返回。rn等于data中已被拷贝到发送缓冲的数据量。

## 链接地址3. 如何判定接收成功接收数据？

        1.接收端回复应答信息。

                这个方法比较土，但也是比较实用的。

        2.计算发送端sock已发送数据量

           可参考：如何判断TCP数据包是否发送成功 作者提到可以使用 ioctl接口：ioctl(tcp\_socket, SIOCOUTQ, &value);

## 链接地址4. 参考：

<http://www.aiuxian.com/article/p-936663.html>

<http://www.aiuxian.com/article/p-1653218.html>

如何判断TCP数据包是否发送成功

转自：

链接地址

# linux send与recv函数详解

1 #include <sys/socket.h>

2 ssize\_t recv(int sockfd, void \*buff, size\_t nbytes, int flags);

3 ssize\_t send(int sockfd, const void \*buff, size\_t nbytes, int flags);

recv 和send的前3个参数等同于read和write。

flags参数值为0或：

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| flags | 说明 | recv | send |
| MSG\_DONTROUTE | 绕过路由表查找 |  | • |
| MSG\_DONTWAIT | 仅本操作非阻塞 | **•** | • |
| MSG\_OOB | 发送或接收带外数据 | • | • |
| MSG\_PEEK | 窥看外来消息 | • |  |
| MSG\_WAITALL | 等待所有数据 | • |  |

 1. send解析

 sockfd：指定发送端套接字描述符。

 buff：    存放要发送数据的缓冲区

 nbytes:  实际要改善的数据的字节数

 flags：   一般设置为0

 1) send先比较发送数据的长度nbytes和套接字sockfd的发送缓冲区的长度，如果nbytes > 套接字sockfd的发送缓冲区的长度, 该函数返回SOCKET\_ERROR;

 2) 如果nbtyes <= 套接字sockfd的发送缓冲区的长度，那么send先检查协议是否正在发送sockfd的发送缓冲区中的数据，如果是就等待协议把数据发送完，如果协议还没有开始发送sockfd的发送缓冲区中的数据或者sockfd的发送缓冲区中没有数据，那么send就比较sockfd的发送缓冲区的剩余空间和nbytes

 3) 如果 nbytes > 套接字sockfd的发送缓冲区剩余空间的长度，send就一起等待协议把套接字sockfd的发送缓冲区中的数据发送完

 4) 如果 nbytes < 套接字sockfd的发送缓冲区剩余空间大小，send就仅仅把buf中的数据copy到剩余空间里(注意并不是send把套接字sockfd的发送缓冲区中的数据传到连接的另一端的，而是协议传送的，send仅仅是把buf中的数据copy到套接字sockfd的发送缓冲区的剩余空间里)。

 5) 如果send函数copy成功，就返回实际copy的字节数，如果send在copy数据时出现错误，那么send就返回SOCKET\_ERROR; 如果在等待协议传送数据时网络断开，send函数也返回SOCKET\_ERROR。

 6) send函数把buff中的数据成功copy到sockfd的改善缓冲区的剩余空间后它就返回了，但是此时这些数据并不一定马上被传到连接的另一端。如果协议在后续的传送过程中出现网络错误的话，那么下一个socket函数就会返回SOCKET\_ERROR。（每一个除send的socket函数在执行的最开始总要先等待套接字的发送缓冲区中的数据被协议传递完毕才能继续，如果在等待时出现网络错误那么该socket函数就返回SOCKET\_ERROR）

 7) 在unix系统下，如果send在等待协议传送数据时网络断开，调用send的进程会接收到一个SIGPIPE信号，进程对该信号的处理是进程终止。

2.recv函数

sockfd: 接收端套接字描述符

buff：   用来存放recv函数接收到的数据的缓冲区

nbytes: 指明buff的长度

flags:   一般置为0

 1) recv先等待s的发送缓冲区的数据被协议传送完毕，如果协议在传送sock的发送缓冲区中的数据时出现网络错误，那么recv函数返回SOCKET\_ERROR

 2) 如果套接字sockfd的发送缓冲区中没有数据或者数据被协议成功发送完毕后，recv先检查套接字sockfd的接收缓冲区，如果sockfd的接收缓冲区中没有数据或者协议正在接收数据，那么recv就一起等待，直到把数据接收完毕。当协议把数据接收完毕，recv函数就把s的接收缓冲区中的数据copy到buff中（注意协议接收到的数据可能大于buff的长度，所以在这种情况下要调用几次recv函数才能把sockfd的接收缓冲区中的数据copy完。recv函数仅仅是copy数据，真正的接收数据是协议来完成的）

 3) recv函数返回其实际copy的字节数，如果recv在copy时出错，那么它返回SOCKET\_ERROR。如果recv函数在等待协议接收数据时网络中断了，那么它返回0。

 4) 在unix系统下，如果recv函数在等待协议接收数据时网络断开了，那么调用 recv的进程会接收到一个SIGPIPE信号，进程对该信号的默认处理是进程终止。