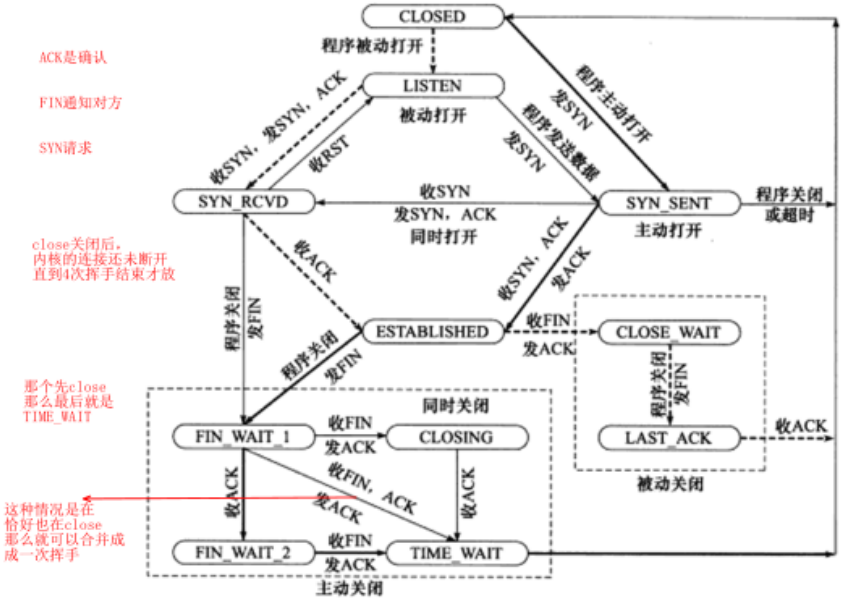
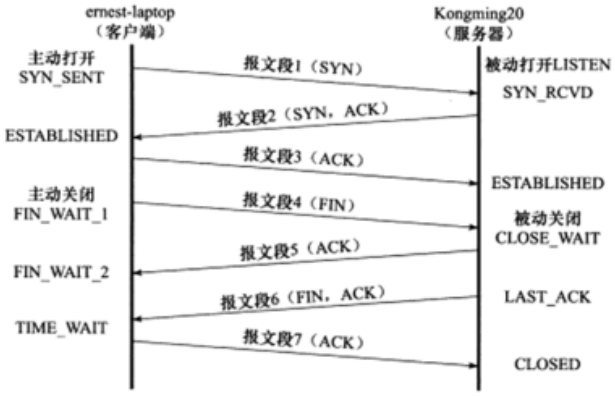
关于握手和挥手问题

首先从状态转移图开始说：





三次握手：

一开始，服务器处于listen状态，等待客户端的链接，客户端开始链接，发送自己的一个SYN，处于SYS_SNET；

服务器接着accept，进入了SYS_RCVD，收到回复一个ACK，并带自己的SYN。

客户端再次进行确认，确认之后发送自己的ACK，三次握手结束，二者都处于了EATABLEISHED；

为什么是三次握手，为什么不能是2次或者4次？

两次握手会发生什么情况 ？因为TCP是一个可靠得传输协议，它自身带有应答机制和重传机制，如果只有两次握手，客

户端向服务器发起连接，只有服务器一端进行ACK回复，服务器在没有收到客户端啊的ACK之前，都认为是ACK丢失，会再次重传，这样的情况只会让系统资源不断消耗

四次握手是没有必要的，因为三次握手之后，两边都已确定对方能接收到自己的数据，所以，四次，五次握手都是没有任何意义的。

四次挥手：

一开始，想结束的一方先发送自己的结束报文FIN，并且进入FIN_WAIT1状态，

另一方服务器端，接收到关闭的请求，进入到close_wait状态，首先会回复一个ACK响应报文，

紧接着，服务器一端也发送上自己的结束报文段，处于了LAST_ACK，等待最后一次的响应。

最后，由客户端并对这个FIn报文进行最后的响应，四次挥手结束。

里面牵扯到了一些问题：

1.为什么有的时候会是三次挥手就完了？

一般关闭的时候，先是客户端先发送FIN关闭信息，此次的FIN的标志只是说不会再有数据发送过来了

而服务器是需要立刻回复一个ACK的，但有可能自己还有数据需要发送给对方，不会立刻也发送出去自己的FIN

当服务器自己的数据处理完了，会发送一个自己的FIN信息

最后还需要一次确认即可。

2.里面刚牵扯到了close_wait这个名词（可以通过查看netstat -ant来查看各个状态信息）

首先，close_wait这个时间段表示着从发送ACK到发送自己的FIN之间的这一段时间，之所以会出现大量close_wait状态，占用着系统的fd，

1.这一般是服务器的问题，没有运行close()，也可能没有机会运行到close（）这一步，可能在close（）的时候，前面还有大量的io操作，导致没能来得及处理。或者死循环

2.响应时间太短，这一边还操作着呢，另一边频频timeout，timeout太小

3.还有就是backlog，这可能设置太大，对方还来不及消费就已经close了，这里就牵扯到listen，在Linux2.2之前，listen的backlog代表的是完成握手和还没有完成握手的总和，而在Linux2.2之后，listen的backlog只代表完成了握手的总和，(半连接的总和是在系统内核设置的，/proc/sys/net/ipv4/tcp_max_syn_backlog)，那么在listen完成握手之后关闭，但却还没有accept去处理这个fd就close，它还是会被accept所接受，会被后面的异常处理所结束掉。

3.time_wait状态

timeout之所以存在是有一定的原因的，

（1）它首先保证了tcp的可靠传输，在timeout时间，它要保证数据已经到达对方，若没有，则会收到对面再次的Fin，然后自己再次发送ack，

（2）一个端口是无法打开多次的，在time_wait状态，我们也不能立刻使用这个刚处于time_wait的端口，这是因为如若我们没有这个状态，我们后面的连接可能会收到来自前面连接的tcp报文。

（3）time_wait存在的时间是两个MSL，最大报文生存周期

但是如果出现大量time_wait怎么办？？？（假想，缩一缩time_wait时间？使用长链接去？多设点服务器去？）

解决方案：

这个time_wait即是友好的，又是让人头疼的。

1.上面说的三个假想，其实都能多少解决一些问题，但是效果和成本的代价。。。分别说一下，修改time_wait时间，那就必须修改tcp内核源码，重编内核，这个是很漫长的。长链接这根据业务需求，而服务器增加，这是最后的计策，代价成本太大；还有一种解决方案是用 setsockopt的SO_LINGER选项强制关闭，选择发送RST而不是FIN，直接越过time_wait，直接进入close，

2。其实更多的是修改内核参数

在这个/etc/sysctl./conf

```
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_fin_timeout = 30
```

然后执行 /sbin/sysctl -p 让参数生效。

net.ipv4.tcp_syncookies = 1 表示开启SYN Cookies。当出现SYN等待队列溢出时，启用cookies来处理，可防范少量SYN攻击，默认为0，表示关闭；我的系统是默认开启的
net.ipv4.tcp_tw_reuse = 1 表示开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接，默认为0，表示关闭；
net.ipv4.tcp_tw_recycle = 1 表示开启TCP连接中TIME-WAIT sockets的快速回收，默认为0，表示关闭。
net.ipv4.tcp_fin_timeout 修改系默认的 TIMEOUT 时间

然后执行 /sbisn/sysctl -p 让参数生效。

统计tcp连接情况：

```
netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'
```

主要说一下这个重用和快速回收

1.首先这个重用并不是使用原先的那个端口，比如服务器的80端口，当断开重连的时候，它会启用一个其他端口来进行操作，刚才的端口还处于time_wait

2.而快速回收牵扯的问题会多一些

tcp_tw_recycle 问题：首先快速回收会碰到之前的一些数据报，也可能会收到之前的Fin，为此，linux在打开tcp_tw_recycle的情况下，会记录下TIME_WAIT连接的对端（peer）信息，包括IP地址、时间戳等：这样，当内核收到同一个IP的SYN包时，就会去比较时间戳，检查SYN包的时间戳是否滞后，如果滞后，就将其丢掉（认为是旧连接的数据

最后，把半连接说一说.

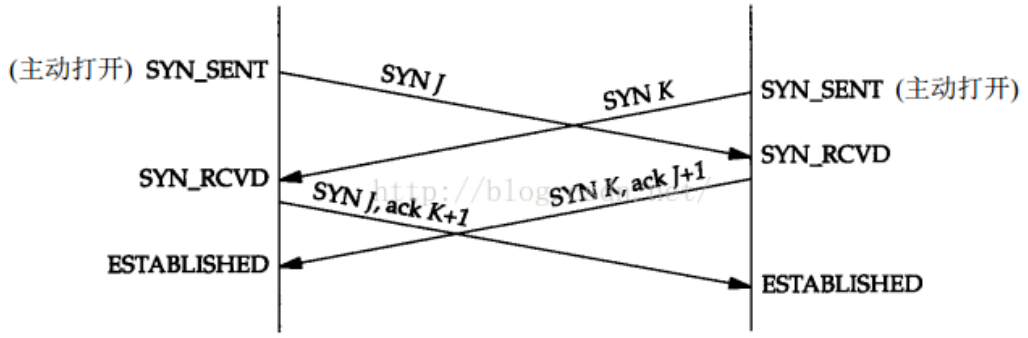
半连接的应用场景是在一端发出FIN，告诉对方自己已经没有什么需要发送了，但仍可以继续接收对方发来的消息，直到对方也发来自己的FIN, read系统调用返回0，认为是对方也关闭了。、

还有一个就是半打开状态：

一端可能会因为一些情况以外断开，当向另一端发送信息，会收到一个复位报文段，

解决办法是接收端发送自己的复位报文段做应答，重新连接进行通信

还有一种就是两端同时打开：（4次握手）



同时关闭：