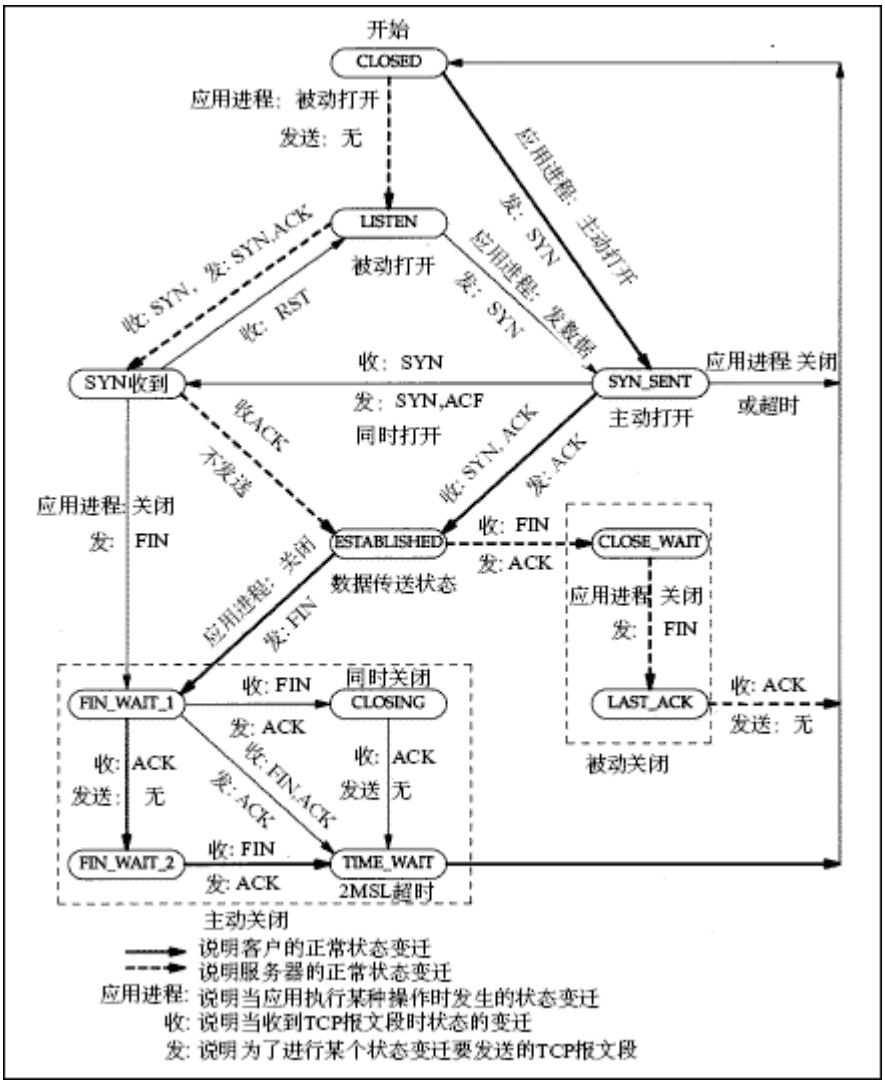
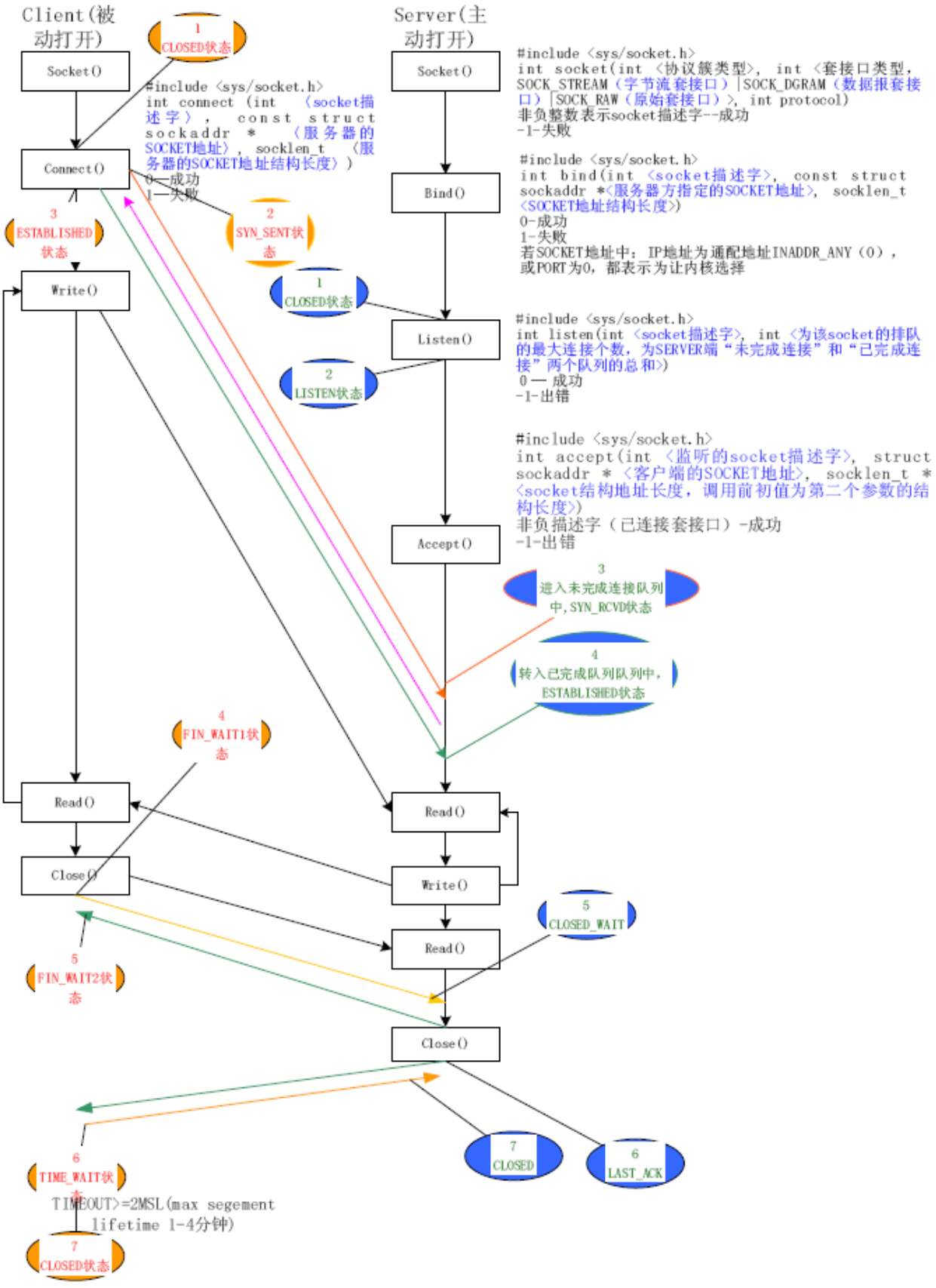
详细介绍 TCP 三次握手建立连接和四次握手断开连接的过程

1. 结合下面的两张图来分析:




Client(被动打开)
Server(主动打开)
1 CLOSED状态
Socket()
#include <sys/socket.h>
int connect (int 〈socket描述字〉, const struct sockaddr * 〈服务器的SOCKET地址〉, socklen_t 〈服务器的SOCKET地址结构长度〉)
0—成功
1—失败
Connect()
3 ESTABLISHED状态
2 SYN_SENT状态
Write()
Socket()
#include <sys/socket.h>
int socket(int 〈协议簇类型〉, int 〈套接口类型, SOCK_STREAM（字节流套接口）|SOCK_DGRAM（数据报套接口）|SOCK_RAW（原始套接口）>, int protocol)
非负整数表示socket描述字--成功
-1-失败
Bind()
#include <sys/socket.h>
int bind(int 〈socket描述字〉, const struct sockaddr *〈服务器方指定的SOCKET地址〉, socklen_t 〈SOCKET地址结构长度〉)
0-成功
1-失败
若SOCKET地址中：IP地址为通配地址INADDR_ANY（0），或PORT为0，都表示为让内核选择
1 CLOSED状态
Listen()
2 LISTEN状态
#include <sys/socket.h>
int listen(int 〈socket描述字〉, int 〈为该socket的排队的最大连接个数，为SERVER端“未完成连接”和“已完成连接”两个队列的总和〉)
0 — 成功
-1-出错
#include <sys/socket.h>
int accept(int 〈监听的socket描述字〉, struct sockaddr *〈客户端的SOCKET地址〉, socklen_t *〈socket结构地址长度，调用前初值为第二个参数的结构长度〉)
非负描述字（已连接套接口）-成功
-1-出错
Accept()
3 进入未完成连接队列中,SYN_RCVD状态
4 转入已完成队列队列中，ESTABLISHED状态
4 FIN_WAIT1状态
Read()
Close()
Read()
Write()
Read()
5 CLOSED_WAIT
5 FIN_WAIT2状态
Close()
6 TIME_WAIT状态
TIMEOUT>=2MSL(max segement lifetime 1-4分钟)
7 CLOSED状态
7 CLOSED
6 LAST_ACK

1) 建立连接协议(三次握手)
a. 客户端发送一个带 SYN 标志的 TCP 报文到服务器.
b. 服务器端回应客户端的, 这个报文同时带 ACK 标志和 SYN 标志. 因此它表示对刚才客户端 SYN 报文的回应; 同时又标 志 SYN 给客户端, 询问客户端是否准备好进行数据通讯.
c. 客户必须再次回应服务段一个 ACK 报文.

2) 连接终止协议(四次握手)
由于 TCP 连接是[全双工]的, 因此每个方向都必须单独进行关闭. 这原则是当一方完成它的数据发送任务后就能发送一个 FIN 来终止这个方向的连接. 收到一个 FIN 只意味着这一方向上没有数据流动, 一个 TCP 连接在收到一个 FIN 后仍能发送数据, 首先进行关闭的一方将执行主动关闭, 而另一方执行被动关闭.
a. TCP 客户端发送一个 FIN, 用来关闭客户到服务器的数据传送.
b. 服务器收到这个 FIN，它发回一个 ACK，确认序号为收到的序号加 1，和 SYN 一样, 一个 FIN 将占用一个序号,
c. 服务器关闭客户端的连接, 发送一个 FIN 给客户端
d. 客户段发回 ACK 报文确认, 并将确认序号设置为收到序号加 1 .

3) 状态分析:
CLOSED: 这个没什么好说的了, 表示初始状态.
LISTEN: 这个也是非常容易理解的一个状态, 表示服务器端的某个 SOCKET 处于监听状态, 可以接受连接了.
SYN_RCVD: 这个状态表示接受到了 SYN 报文, 在正常情况下, 这个状态是服务器端的 SOCKET 在建立 TCP 连接时的三次握手会话 过程中的一个中间状态, 很短暂, 基本上用 netstat 你是很难看到这种状态的, 除非你特意写了一个客户端测试程序, 故意将三次 TCP 握手过程中最后一个 ACK 报文不予发送. 因此这 种状态时, 当收到客户端的 ACK 报文后, 它会进入到 ESTABLISHED 状态.
SYN_SENT: 这个状态与 SYN_RCVD 遥想呼应, 当客户端 SOCKET 执行 CONNECT 连接时, 它首先发送 SYN 报文，因此也随即它会 进入到了 SYN_SENT 状态, 并等待服务端的发送三次握手中的第 2 个报文, SYN_SENT 状态表示客户端已发送 SYN 报文.
ESTABLISHED：这个容易理解了，表示连接已经建立了.
FIN_WAIT_1: 这个状态要好好解释一下，其实 FIN_WAIT_1 和 FIN_WAIT_2 状态的真正含义都是表示等待对方的 FIN 报文.而这两种状态的区别是： FIN_WAIT_1 状态实际上是当 SOCKET 在 ESTABLISHED 状态时, 它想主动关闭连接, 向对方发送了 FIN 报文，此时该 SOCKET 即进入到 FIN_WAIT_1 状态. 而当对方回应 ACK 报文后, 则进入到 FIN_WAIT_2 状态, 当然在实际的正 常情况下, 无论对方何种情况下, 都应该马上回应 ACK 报文，所以 FIN_WAIT_1 状态一般是比较难见到的, 而 FIN_WAIT_2 状态还有时常常可以用 netstat 看到.
FIN_WAIT_2：上面已经详细解释了这种状态，实际上 FIN_WAIT_2 状态下的 SOCKET，表示半连接，也即有一方要求 close 连接，但另外还告诉对方，我暂时还有点数据需要传送给你，稍后再关闭连接.
TIME_WAIT: 表示收到了对方的 FIN 报文，并发送出了 ACK 报文，就等 2MSL 后即可回到 CLOSED 可用状态了。如果 FIN_WAIT_1 状态下，收到了对方同 时带 FIN 标志和 ACK 标志的报文时，可以直接进入到 TIME_WAIT 状态，而无须经过 FIN_WAIT_2 状态.
CLOSING: 这种状态比较特殊，实际情况中应该是很少见，属于一种比较罕见的例外状态.正常情况下，当你发送 FIN 报文后，按理来说是应 该先收到（或同时收到）对方的 ACK 报文，再收到对方的 FIN 报文。但是 CLOSING 状态表示你发送 FIN 报文后，并没有收到对 方的 ACK 报文，反而却也收到了对方的 FIN 报文.什么情况下会出现 此种情况呢？其实细想一下，也不难得出结论：那就是如果双方几乎在同时 close 一个 SOCKET 的话，那么就出现了双 方同时发送 FIN 报文的情况，也即会出现 CLOSING 状态，表示双方都正在关闭 SOCKET 连接.
CLOSE_WAIT: 这种状态的含义其实是表示在等待关闭。怎么理解呢？当对方 close 一个 SOCKET 后发送 FIN 报文给自己，你系统毫 无疑问地会回应一个 ACK 报文给对方，此时则进入到 CLOSE_WAIT 状态. 接下来呢，实际上 你真正需要考虑的事情是察看你是否还有数据发送给对方，如果没有的话, 那么你也就可以 close 这个 SOCKET，发送 FIN 报文给对方，也即关闭 连接。所以你在 CLOSE_WAIT 状态下，需要完成的事情是等待你去关闭连接.

LAST_ACK: 这个状态还是比较容易好理解的，它是被动关闭一方在发送 FIN 报文后，最后等待对方的 ACK 报文. 当收到 ACK 报文后，也即可以进入 到 CLOSED 可用状态了.

补充说明:
a. FIN_WAIT_2 是一种半连接状态, 在这种状态下, 主动断开连接的一方不能发送数据给对方, 而另一方仍然可以发送数据过来.
b. TIME_WAIT 主动断开连接的一方可能进入这个状态, 就是当它发送 FIN 数据包给对方之后， 收到了对方同时带 FIN 标志和 ACK 标志的报文时，直接进入到 TIME_WAIT 状态，而无须经过 FIN_WAIT_2 状态.
c. CLOSING 是如果双方几乎在同时 close 一个 SOCKET 的话，那么就出现了双 方同时发送 FIN 报文的情况，也即会出现 CLOSING 状态，表示双方都正在关闭 SOCKET 连接.
d. 为什么建立连接协议是三次握手，而关闭连接却是四次握手呢？
这是 因为服务端的 LISTEN 状态下的 SOCKET 当收到 SYN 报文的建连请求后，它 可以把 ACK 和 SYN （ ACK 起应答作用，而 SYN 起同步作用) 放在一个 报文里来发送. 但关闭连接时，当收到对方的 FIN 报文通知时，它仅仅表示对方没有数据发送给你了; 但未必你所有的数据都全部发送给对方了，所以你可以未必会马上会关闭 SOCKET, 也即你可能还需要发送一些数据给对方之后，再发送 FIN 报文给对方来表示你同 意现在可以关闭连接了, 所以它这里的 ACK 报文和 FIN 报文多数情况下都是分开 发送的. 当然也可以一起发送, 这样主动断开连接的一方会跳过 FIN_WAIT_2 状态直接进入 TIME_WAIT 状态.
e. 为什么 TIME_WAIT 状态还需要等 2MSL 后才能返回到 CLOSED 状态？
这是因为：虽然双方都同意关闭连接了，而且握手的 4 个报文也都协调和发送完毕，按理可以直接回到 CLOSED 状态 (就好比从 SYN_SEND 状态到 ESTABLISH 状态那样) ; 但是因为 我们必须要假想网络是不可靠的，你无法保证你最后发送的 ACK 报文会一定被对方收到，因此对方处于 LAST_ACK 状态下的 SOCKET 可能会因为超时未收到 ACK 报文, 而重发 FIN 报文，所以这个 TIME_WAIT 状态的作用就是用来重发可能丢失的 ACK 报文.
2MSL (Maximum Segment Lifetime) 时间, 大约 2 分钟左右, 主动发起关闭连接的一方会进入 time_wait 状态, 这个时候, 进程所占用的端口号不能被释放. 除非在你的程序中用 setsockopt 设置端口可重用(SOCK_REUSE)的选项.