TCP/IP

运行在TCP协议上的协议：

- HTTP（Hypertext Transfer Protocol，超文本传输协议），主要用于普通浏览。
- HTTPS（Hypertext Transfer Protocol over Secure Socket Layer, or HTTP over SSL，安全超文本传输协议）,HTTP协议的安全版本。
- FTP（File Transfer Protocol，文件传输协议），由名知义，用于文件传输。
- POP3（Post Office Protocol, version 3，邮局协议），收邮件用。
- SMTP（Simple Mail Transfer Protocol，简单邮件传输协议），用来发送电子邮件 。
- TELNET（Teletype over the Network，网络电传），通过一个终端（terminal）登陆到网络。
- SSH（Secure Shell，用于替代安全性差的TELNET），用于加密安全登陆用。

运行在UDP协议上的协议：

- BOOTP（Boot Protocol，启动协议），应用于无盘设备。
- NTP（Network Time Protocol，网络时间协议），用于网络同步。

其他：

- DNS（Domain Name Service，域名服务），用于完成地址查找，邮件转发等工作（运行在TCP和UDP协议上）。
- ECHO（Echo Protocol，回绕协议），用于查错及测量应答时间（运行在TCP和UDP协议上）。
- SNMP（Simple Network Management Protocol，简单网络管理协议），用于网络信息的收集和网络管理。
- DHCP（Dynamic Host Configuration Protocol，动态主机配置协议），动态配置IP地址。
- ARP（Address Resolution Protocol，地址解析协议），用于动态解析以太网硬件的地址。

TCP（IP协议6）是一个“可靠的”、面向连结的传输机制，它提供一种可靠的字节流保证数据完整、无损并且按顺序到达。TCP尽量连续不断地测试网络的负载并且控制发送数据的速度以避免网络过载。另外，TCP试图将数据按照规定的顺序发送。这是它与UDP不同之处，这在实时数据流或者路由高网络层丢失率应用的时候可能成为一个缺陷。

UDP（IP协议号17）是一个无连结的数据报协议。它是一个“best effort”或者“不可靠”协议—不是因为它特别不可靠，而是因为它不检查数据包是否已经到达目的地，并且不保证它们按顺序到达。如果一个应用程序需要这些特点，它必须自己提供或者使用TCP。

1、建立连接协议（三次握手）

（1）客户端发送一个带SYN标志的TCP报文到服务器。这是三次握手过程中的报文1。 （2） 服务器端回应客户端的，这是三次握手中的第2个报文，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。 （3） 客户必须再次回应服务段一个ACK报文，这是报文段3。 **2**、连接终止协议（四次挥手） 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。 （1） TCP客户端发送一个FIN，用来关闭客户到服务器的数据传送（报文段4）。 （2） 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（报文段5）。和SYN一样，一

个FIN将占用一个序号。 （3） 服务器关闭客户端的连接，发送一个FIN给客户端（报文段6）。 （4） 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（报文段7）。

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。

最后有2个问题的回答，我自己分析后的结论（不一定保证100%正确） 1、 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。 2、 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。

新：

1. **TCP**的三次握手最主要是防止已过期的连接再次传到被连接的主机。(废弃连接问题)

如果采用两次的话，会出现下面这种情况。比如是A机要连到B机，结果发送的连接信息由于某种原因没有到达B机； 于是，A机又发了一次，结果这次B收到了，于是就发信息回来，两机就连接。传完东西后，断开。结果这时候，原先没有到达的连接信息突然又传到了B机，于是B机发信息给A，然后B机就以为和A连上了，这个时候B机就在等待A传东西过去。

2. 三次握手改成仅需要两次握手，死锁是可能发生

考虑计算机A和B之间的通信，假定B给A发送一个连接请求分组，A收到了这个分组，并发送了确认应答分组。按照两次握手的协定，A认为连接已经成功地建立了，可以开始发送数据分组。可是，B在A的应答分组在传输中被丢失的情况下，将不知道A是否已准备好，不知道A建议什么样的序列号，B甚至怀疑A是否收到自己的连接请求分组。在这种情况下，B认为连接还未建立成功，将忽略A发来的任何数据分组，只等待连接确认应答分组。而A在发出的分组超时后，重复发送同样的分组。这样就形成了死锁

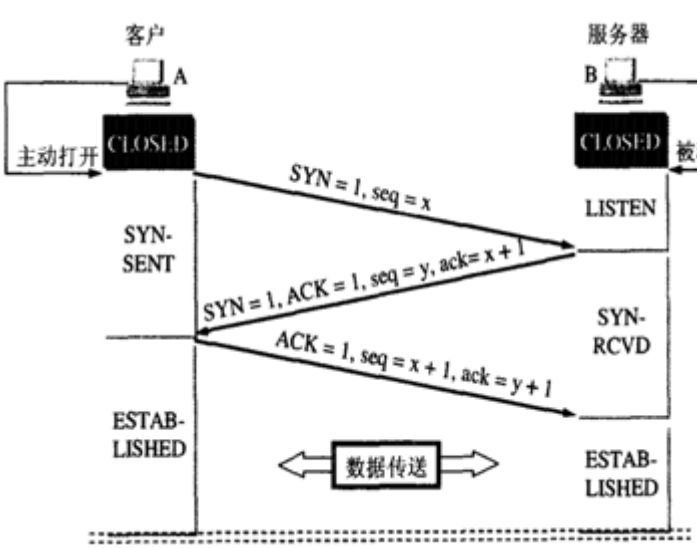
3. TCP通讯中，select到读事件，但是读到的数据量是0，为什么，如何解决????

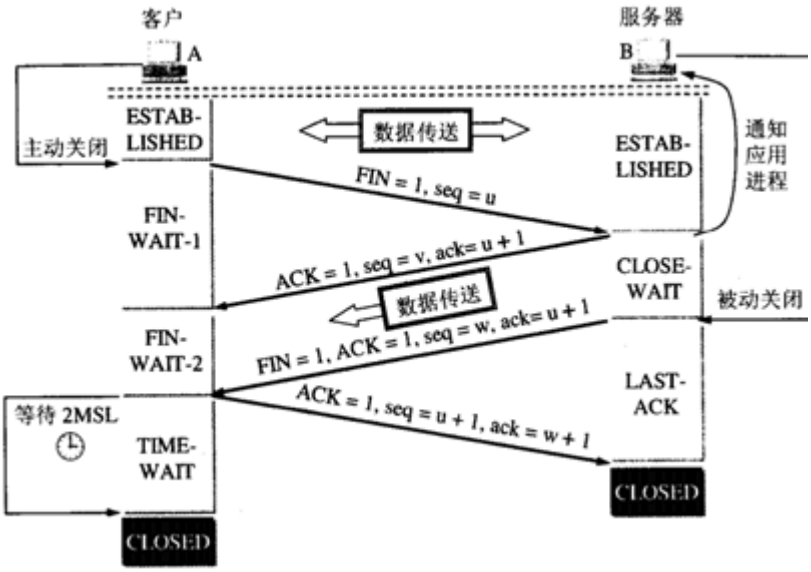
select 返回0代表超时。select出错返回-1。 select到读事件，但是读到的数据量为0，说明对方已经关闭了socket的读端。本端关闭读即可。 当select出错时，会将接口置为可读又可写。这时就要通过判断select的返回值为-1来区分。

4. **2MSL(maximum segment lifetime)** 报文最大生存时间

(1).等待一段时间，防止最后的FIN的ACK包丢失，对方未收到ACK会重发FIN (2).TCP连接在2MSL时间内ip,port不能重新被bind 5.复位报文**(RST)** (1).接收到不存在端口的连接请求，回复RST包(但是udp是响应ICMP端口不可达的error) (2).异常终止一个连接，发送RST包，收到RST的一方终止该连接。 (3).收到一个半开连接的数据包后，回复RST,收到RST的一方终止该连接。 6.几种情况 (1).服务器未开启服务，回复RST (2).服务器连接正常关闭，回复FIN (3).服务器进程异常终止，回复RST。 (4).服务器直接掉电，如果客户端没有"发送数据"或者"设置keepalive选项"，客户端将一直保持此半开连接。 如果客户端重新连接，将新建立一个连接。 (5).服务器重启时，如果收到一个半开连接的数据包，回复RST.

- 常见面试题
  - **TCP**协议和**UDP**协议的区别是什么
    - TCP协议是有连接的，有连接的意思是开始传输实际数据之前TCP的客户端和服务器端必须通过三次握手建立连接，会话结束之后也要结束连接。而UDP是无连接的
    - TCP协议保证数据按序发送，按序到达，提供超时重传来保证可靠性，但是UDP不保证按序到达，甚至不保证到达，只是努力交付，即便是按序发送的序列，也不保证按序送到。
    - TCP协议所需资源多，TCP首部需20个字节（不算可选项），UDP首部字段只需8个字节。
    - TCP有流量控制和拥塞控制，UDP没有，网络拥堵不会影响发送端的发送速率
    - TCP是一对一的连接，而UDP则可以支持一对一，多对多，一对多的通信。
    - TCP面向的是字节流的服务，UDP面向的是报文的服务。
    - TCP介绍和UDP介绍
  - 请详细介绍一下**TCP**协议建立连接和终止连接的过程？
    - 助于理解的一段话：
    - 建立一个TCP连接时，会发生下述情形：

- 1、服务器端必须做好准备接受外来的连接。这通常通过 socket(), bind(), listen() 三个函数来完成的。我们称之为 被动打开(passive open). 2、客户端通过调用connect发起主动打开(active open)。这导致客户端TCP发送SYN同步分节。它告诉服务器客户端在(待建立的)连接中发送的数据的初始化序列号。通用SYN分节不携带数据， 3、服务器必须确认(ACK) 客户端的SYN，同时自己也得发送一个SYN分节，它含有服务器将在统一连接中发送的数据的初始化序号。服务器在单个分节中发送SYN和对客户端SYN的ACK确认。 4、客户端必须确认服务器的SYN。 TCP连接的终止： 1、某个应用程序首先调用close，主动关闭(active close) 该端的TCP于是发送一个FIN分节，表示数据发送完毕。 2、接收到这个FIN的对端执行被动关闭(passive close)。这个FIN是TCP确认。它的接收也作为一个文件结束符(end of file) 传递给接收端的应用程序(放在排队等候应用进程接收的任何其他数据之后)，因为FIN的接收意味着接收端应用程序在相应连接上再无额外数据可以接收。 3、一段时间以后，接收到这个文件结束符的应用进程将调用close关闭它的套接字。这导致它的TCP也发送一个FIN。 4、接收这个最终FINd额原发送端TCP(即执行主动关闭的一端)确认这个FIN
- 两幅图（来源）：
  - 建立连接：三次握手
  - 

  - 关闭连接：四次挥手
  - 

- 三次握手建立连接时，发送方再次发送确认的必要性？

        - 主要是为了防止已失效的连接请求报文段突然又传到了B,因而产生错误。假定出现一种异常情况，即A发出的第一个连接请求报文段并没有丢失，而是在某些网络结点长时间滞留了，一直延迟到连接释放以后的某个时间才到达B，本来这是一个早已失效的报文段。但B收到此失效的连接请求报文段后，就误认为是A又发出一次新的连接请求，于是就向A发出确认报文段，同意建立连接。假定不采用三次握手，那么只要B发出确认，新的连接就建立了，这样一直等待A发来数据，B的许多资源就这样白白浪费了。
    - 四次挥手释放连接时，等待**2MSL**的意义？
        - 第一，为了保证A发送的最有一个ACK报文段能够到达B。这个ACK报文段有可能丢失，因而使处在LAST-ACK状态的B收不到对已发送的FIN和ACK报文段的确认。B会超时重传这个FIN和ACK报文段，而A就能在2MSL时间内收到这个重传的ACK+FIN报文段。接着A重传一次确认。
        - 第二，就是防止上面提到的已失效的连接请求报文段出现在本连接中，A在发送完最有一个ACK报文段后，再经过2MSL，就可以使本连接持续的时间内所产生的所有报文段都从网络中消失。
-
    - 常见的应用中有哪些是应用**TCP**协议的，哪些又是应用**UDP**协议的，为什么它们被如此设计？
        - 以下应用一般或必须用udp实现？
            - 多播的信息一定要用udp实现，因为tcp只支持一对一通信。
            - 如果一个应用场景中大多是简短的信息，适合用udp实现，因为udp是基于报文段的，它直接对上层应用的数据封装成报文段，然后丢在网络中，如果信息量太大，会在链路层中被分片，影响传输效率。
            - 如果一个应用场景重性能甚于重完整性和安全性，那么适合于udp，比如多媒体应用，缺一两帧不影响用户体验，但是需要流媒体到达的速度快，因此比较适合用udp
            - 如果要求快速响应，那么udp听起来比较合适
            - 如果又要利用udp的快速响应优点，又想可靠传输，那么只能考上层应用自己制定规则了。
            - 常见的使用udp的例子：ICQ,QQ的聊天模块。
        - 以qq为例的一个说明（转载自知乎）

          登陆采用TCP协议和HTTP协议，你和好友之间发送消息，主要采用UDP协议，内网传文件采用了P2P技术。总来的说： 1.登陆过程，客户端client 采用TCP协议向服务器server发送信息，HTTP协议下载信息。登陆之后，会有一个TCP连接来保持在线状态。 2.和好友发消息，客户端client采用UDP协议，但是需要通过服务器转发。腾讯为了确保传输消息的可靠，采用上层协议来保证可靠传输。如果消息发送失败，客户端会提示消息发送失败，并可重新发送。 3.如果是在内网里面的两个客户端传文件，QQ采用的是P2P技术，不需要服务器中转。

因特网应用	IP	ICMP	UDP	TCP	SCTP
ping		•			
traceroute		•	•		
OSPF（路由协议）	•				
RIP（路由协议）			•		
BGP（路由协议）				•	
BOOTP（引导协议）			•		
DHCP（引导协议）			•		
NTP（时间协议）			•		
TFTP（低级FTP）			•		
SNMP（网络管理）			•		
SMTP（电子邮件）				•	
Telnet（远程登录）				•	
SSH（安全的远程登录）				•	
FTP（文件传送）				•	
HTTP（Web）				•	
NNTP（网络新闻）				•	
LPR（远程打印）				•	
DNS（域名系统）			•	•	
NFS（网络文件系统）			•	•	
Sun RPC（远程过程调用）			•	•	
DCE RPC（远程过程调用）			•	•	
IUA（IP之上的ISDN）					•
M2UA和M3UA（SS7电话信令）					•
H.248（媒体网关控制）			•	•	•
H.323（IP电话）			•	•	•
SIP（IP电话）			•	•	•

图2-19　各种常见因特网应用的协议使用情况