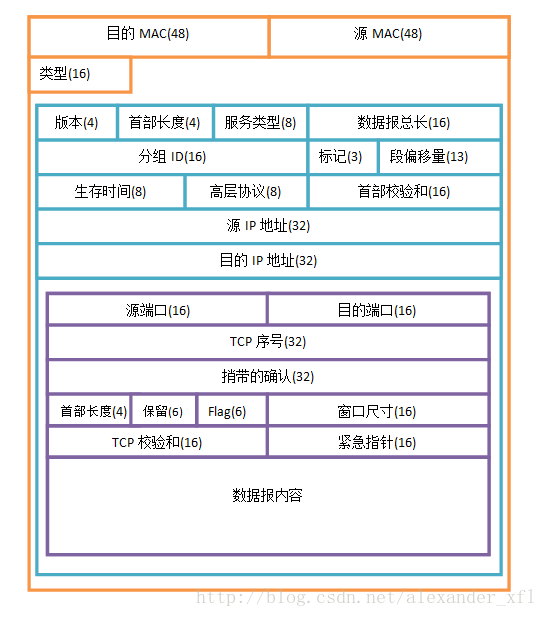
****

****

int socket(int protofamily, int type, int protocol);

参数：

* protofamily：协议族（family）。常用的协议族有 AF\_INET(IPV4)、AF\_INET6(IPV6)等等。协议族决定了socket的地址类型。
* type：指定socket类型。常用的类型有 SOCK\_STREAM、SOCK\_DGRAM、SOCK\_RAW等等。
* protocol：指定协议。常用的协议有 IPPROTO\_TCP、IPPTOTO\_UDP、IPPROTO\_SCTP 等，它们分别对应TCP传输协议、UDP传输协议、STCP传输协议。

返回值：

* sockfd：socket 描述符，与文件描述符使用相同。

int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

参数：

* sockfd：socket描述字，唯一标识一个socket。
* addr：一个const struct sockaddr \* 指针，指向要绑定给sockfd的协议地址。
* addrlen：对应的是地址的长度。

通常服务器在启动的时候都会绑定一个众所周知的地址（如ip地址+端口号），用于提供服务，客户就可以通过它来接连服务器；而客户端就不用指定。这就是为什么通常服务器端在listen之前会调用bind()，而客户端就不会调用，而是在connect()时由系统随机生成一个。

**注意**：在将一个地址绑定到socket的时候，请先将主机字节序转换成为网络字节序。

int listen(int sockfd, int backlog);

int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

listen函数的第一个参数即为要监听的socket描述字，第二个参数为相应socket可以排队的最大连接个数。

connect函数的第一个参数即为客户端的socket描述字，第二参数为服务器的socket地址，第三个参数为socket地址的长度。

int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen); //返回连接connect\_fd

参数

sockfd：监听套接字，这个套接字用来监听一个端口，当有一个客户与服务器连接时，它使用这个一个端口号，而此时这个端口号正与这个套接字关联。当然客户不知道套接字这些细节，它只知道一个地址和一个端口号。

addr：用来接受一个返回值，指定客户端的地址。

len：addr的结构的大小，指明addr结构所占有的字节个数。

**注意：**accept默认会阻塞进程，直到有一个客户连接建立后返回。连接套接字并没有占用新的端口与客户端通信，依然使用的是与监听套接字socketfd一样的端口号。

int close(int fd);

关闭一个套接字描述符。

int open(const char \*pathname , int oflag, mode\_t mode) ;

oflag :

* O\_RDONLY 只读打开。
* O\_WRONLY 只写打开。
* O\_RDWR 读、写打开。
* O\_APPEND 每次写时都加到文件的尾端。
* O\_NONBLOCK 如果pathname指的是一个FIFO、一个块特殊文件或一个字符特殊文件，则此选择项为此文件的本次打开操作和后续的 I/O操作设置非阻塞方式。
* O\_CREAT 若此文件不存在则创建它。使用此选择项时，需同时说明第三个参数 mode。

ssize\_t read(int fd, void \*buff, size\_t nbytes) ;

从打开的文件中读数据。

read 成功，返回读到的字节数。已到文件末尾，返回0。出错时返回-1，并设置error。

ssize\_t write(int fd, const void \* buff, size\_tn bytes) ;

返回：若成功为已写的字节数，若出错为- 1。

int fcntl(int fd, int cmd,.../\* int arg \* / ) ;

改变已经打开文件的性质。

功能1：获得/设置文件描述符标记（cmd = F\_G E T F D或F \_ S E T F D）。

int val = fcntl(fd,F\_GETFL,0);

val |= O\_NONBLOCK;

fcntl(fd,F\_SETFL,val); //设置文件描述符为非阻塞

r e a d v和w r i t e v函数用于在一个函数调用中读、写多个非连续缓存。

**OSI、TCP/IP，五层协议**

OSI（7层）：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层。

TCP/IP（4层）：网络接口层、 网络层、传输层、 应用层。

五层协议：物理层、数据链路层、网络层、传输层、 应用层。

每一层的协议如下：

物理层：RJ45、IEEE802.3

数据链路：VLAN、MAC

网络层：IP、ARP

传输层：TCP、UDP

应用层：DNS、HTTP、WWW

每一层的作用如下：

物理层：通过媒介传输比特

数据链路层：提供点到点的帧传输

网络层：提供主机之间数据报文的传输

传输层：提供主机之间数据传送服务

应用层：提供用户接口，负责两个进程之间的信息交换。

会话层：建立、管理和终止会话

表示层：数据加密解密、压缩等

**IP协议**

提供在源地址和目的地址之间传输数据报的服务。它是无连接、不可靠的。

**IP地址的分类**

**IP地址分类思想：把32bit划分为两个字段：网络号和主机号**

A类：网络号0，1.0.0.0 - 126.255.255.255

B类：网络号10，128.0.0.0 - 191.255.255.255

C类：网络号110，192.0.0.0 - 223.255.255.255

D类：网络号1110，用于多播

E类：网络号1111，保留

子网掩码

将某个IP地址划分成[网络地址](http://baike.baidu.com/view/547479.htm)和[主机地址](http://baike.baidu.com/view/547482.htm)两部分。

用于子网掩码的位数决定于可能的子网数目和每个子网的主机数目。

IP地址与子网掩码相与得到主机号

**ARP地址解析协议**

通过IP地址获得对应的MAC地址。

1：每个主机的ARP缓冲区都有一个ARP列表，保存IP地址和MAC地址之间的对应关系。

2：当源主机要发送数据时，首先检查ARP列表中是否有IP地址对应主机的MAC地址，如果有，则直接发送数据，如果没有，就向本网段的所有主机发送ARP数据包，该数据包内容有：源主机 IP地址，源主机MAC地址，目的主机的IP 地址。

3：当本网络的所有主机收到该ARP数据包，检查数据包中目的IP地址是否是自己的IP地址，如果不是，则忽略该数据包，如果是，则从数据包中取出源主机的IP和MAC地址写入到ARP列表中，然后将自己的MAC地址写入ARP响应包中，告诉源主机自己是它想要找的MAC地址。

4：源主机收到ARP响应包后，将目的主机的IP和MAC地址写入ARP列表，并利用此信息发送数据。如果源主机一直没有收到ARP响应数据包，表示ARP查询失败。

**TCP和UDP**

TCP和UDP都是在传输层完成指定的功能。

TCP提供面向连接的、可靠的数据流传输，而UDP提供的是非面向连接的、不可靠的数据流传输。

TCP传输单位称为TCP报文段，UDP传输单位称为用户数据报。

TCP注重数据安全性，UDP数据传输快。

**TCP连接三次握手：**

第一次握手：客户端发送syn包(seq=x)到服务器，并进入SYN\_SEND状态，等待服务器确认；

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=x+1），同时自己也发送一个SYN包（seq=y），即SYN+ACK包，此时服务器进入SYN\_RECV状态；

第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=y+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。

**断开四次握手**

第一次挥手：主动关闭方发送一个FIN，用来关闭主动方到被动关闭方的数据传送，不会再发数据(在fin包之前发送出去的数据，如果没有收到对应的ack确认报文，主动关闭方依然会重发这些数据)，此时主动关闭方还可以接受数据。

第二次挥手：被动关闭方收到FIN包后，发送一个ACK给对方，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号）。  
第三次挥手：被动关闭方发送一个FIN，用来关闭被动关闭方到主动关闭方的数据传送，不会再发数据。  
第四次挥手：主动关闭方收到FIN后，发送一个ACK给被动关闭方，确认序号为收到序号+1，完成四次挥手。



**TCP建立连接为什么需要三次握手？**

**主要是防止已经过期的连接再次到达被连接的主机。**如果A向B发送的连接信息没有到达，于是A重新发生一次，B收到并回应，建立了连接。数据交换完成之后，断开连接。

如果这个时候，第一次发生的连接信息又到达了B，B就会以为A要建立连接，发送应答后就建立了新的连接，它会一直等待A发送数据。

**TCP断开连接为什么需要四次挥手？**

**主要原因是被动关闭方是否还有数据发送**。FIN信号表示不再发送数据，如果被动关闭方没有数据发送，它也会发送FIN给对方。如果还有数据发送，就要等数据发送完之后才给对方发送FIN信号。

**MSL（最大分段生存期）**

指明TCP报文在Internet上最长生存时间。每个具体的TCP实现都必须选择一个确定的MSL值。RFC 1122建议是2分钟,但BSD传统实现采用了30秒。TIME\_WAIT 状态最大保持时间是2 \* MSL，也就是1-4分钟。

**TIME\_WAIT状态**

主动关闭方发送最后一个ACK信号之后，会进入TIME\_WAIT状态。停留2个MSL时间后进入CLOSED状态。

**TIME\_WAIT状态存在的理由：**

**可靠地关闭TCP连接**

TCP协议在关闭连接的四次握手过程中，最终的ACK是由主动关闭方发出的，如果这个ACK丢失，被动关闭方将重发送FIN。如果主动关闭方处于CLOSED 状态，就会响应RST而不是ACK。所以主动关闭方应该维持TIME\_WAIT状态，再一次发送ACK。

**防止上一次连接中包重新出现**

TCP协议不允许处于TIME\_WAIT状态的连接启动一个新的可用连接，因为TIME\_WAIT状态持续2MSL，就可以保证当成功建立一个新TCP连接的时候，来自旧连接重复分组已经在网络中消逝。

**TIME\_WAIT过多的原因、危害及如何避免**

TIME\_WAIT状态是主动关闭方保存的状态。如果服务端有大量的连接处于TIME\_WAIT状态，则说明是由服务端主动关闭连接的，而且服务端没有对TIME\_WAIT状态的连接快速重用或回收。

危害：占用系统的socket资源

解决方法：

保证由客户端主动发起关闭；

让服务端能够快速回收和重用那些TIME\_WAIT的资源。

/etc/sysctl.conf

开启重用   net.ipv4.tcp\_tw\_reuse = 1

快速回收 net.ipv4.tcp\_tw\_recycle = 1

**TCP的可靠性如何保证：**

1. TCP连接和断开都需要双方握手确认；
2. 超时重发：当TCP发出一个段后，它启动一个定时器，等待接收端确认这个报文段。如果超过规定时间，将重发这个报文段。
3. 接收端收到一个段后，将发送一个ACK确认这个段（延迟一段时间发送，完整校验）
4. 接收端进行首部和数据的检验和。如果检验和有差错，TCP将丢弃这个报文段和不确认收到此报文段。
5. 接收端丢弃重复的数据。
6. 流量控制。根据双方设备性能和网络状况确定发送速率。

**HTTP协议（超文本传输协议）**

是用于从万维网服务器传输超文本到本地浏览器的传送协议。

HTTP是一个属于应用层的面向对象的协议，基于TCP/IP通信协议来传递数据（HTML文件, 图片文件, 查询结果等）。

浏览器通过URL向服务端发送请求。Web服务器根据接收到的请求，向客户端发送响应信息。

**HTTP协议请求**

GET：请求读取由URL所标志的信息。

POST：给服务器添加信息（如注释）。

PUT：在给定的URL下存储一个文档。

DELETE：删除给定的URL所标志的资源。

**DNS域名系统**

一个域名和[IP地址](http://baike.baidu.com/item/IP%E5%9C%B0%E5%9D%80)相互映射的[分布式数据库](http://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E6%95%B0%E6%8D%AE%E5%BA%93)，提供通过[主机](http://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA)名得到该主机名对应IP地址的服务。

当客户机需要使用主机名时，向DNS服务器发送请求，查询主机所对应的[IP地址](http://baike.baidu.com/item/IP%E5%9C%B0%E5%9D%80)。

**浏览器输入网址后执行的过程：**

1. 通过DNS解析到域名对应的IP地址。

依次查找浏览器缓存、系统缓存、路由器缓存、ISP NDS缓存、根域名服务器。

1. 浏览器与服务器建立TCP连接。
2. 浏览器向web服务器发送一个HTTP请求
3. 服务器响应HTTP请求，处理并返回一个响应。
4. 浏览器解析网页内容。

**涉及到的协议：**

应用层：HTTP(WWW访问协议)，DNS(域名解析服务) 。

传输层：TCP(为HTTP提供可靠的数据传输)，UDP(DNS使用UDP传输) 。

网络层：IP(IP数据数据包传输和路由选择)，ICMP(提供网络传输过程中的差错检测)，ARP(将本机的默认网关IP地址映射成物理MAC地址)。

**TCP滑动窗口**

每个主机都有一个发送缓存和接收缓存。同时每个主机都有一个发送窗口和一个接收窗口。对于发送方，发送缓存中的已发送但未收到确认和未发送但允许发送的这两部分就是发送窗口。

对于接收方，接收缓存中未接收准备接收的部分成为接收窗口。

TCP利用这个窗口，慢慢的从数据的左边移动到右边，把处于窗口范围内的数据发送出去。

流量控制：就是让发送速率不要过快，让接收方来得及接收。利用滑动窗口机制就可以实施流量控制。发送方根据接收方接收窗口的大小来动态调整自己发送窗口的大小。如果接收方没有缓存可以用，发送窗口的就会设置为0， 停止发送数据。

拥塞控制：防止过多的数据流入到网络中。

方法：

发送方维持一个拥塞窗口，拥塞窗口的大小取决于网络的拥塞程度而动态变化。发送方让自己的发送窗口大小等于拥塞窗口的大小。同时还要设置一个慢开始门限。

拥塞窗口 > 慢开始门限，用慢开始

拥塞窗口 < 慢开始门限，改用拥塞避免

拥塞窗口 = 慢开始门限，都可使用

1. 慢开始

主机开始发送数据时，并不清楚网络的负荷情况。因此，较好的方法是先探测一下，由小到大逐渐增大发送窗口。首先设置拥塞窗口为最大报文段长度，每次增加这个长度。

1. 拥塞避免

让拥塞窗口缓慢地增大，每次只增加1，而不是加倍。拥塞窗口按线性规律缓慢增长，比慢开始算法的拥塞窗口增长速率缓慢得多。

慢开始和拥塞避免算法，当发送方判断网络出现拥塞（根据是否收到确认），将慢开始门限调整为发送窗口一半，拥塞窗口调整为1 。然后重新开始。

**MTU**

**最大传输单元**

如果发送的数据包大于MTU，则该数据包在传输过程中会被IP协议分片传输，使得每片数据报的长度小于MTU。分片传输的IP数据报不一定按序到达，但IP首部中的信息能让这些数据报片按序组装。IP数据报的分片与重组是在网络层进完成的。