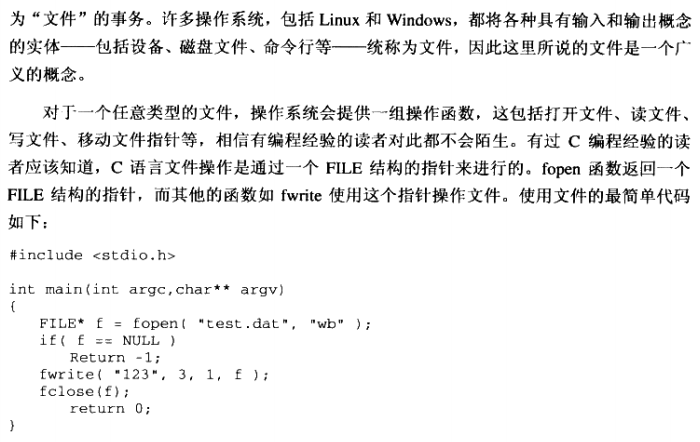
## 必备基础知识：

### 一、互联网协议：

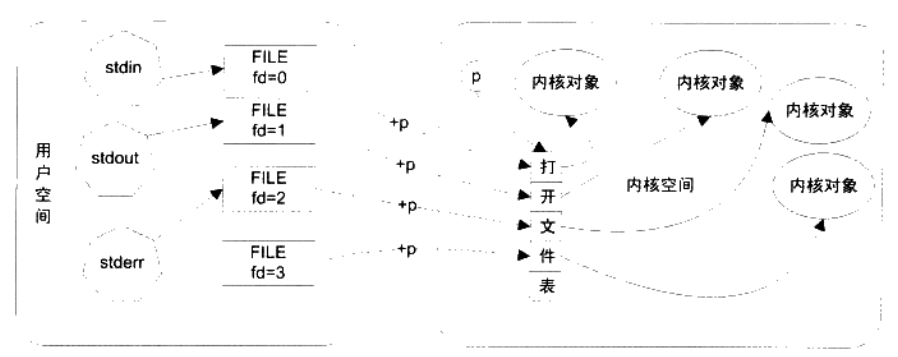
1、<http://www.ruanyifeng.com/blog/2012/05/internet_protocol_suite_part_i.html>

2、<http://www.ruanyifeng.com/blog/2012/06/internet_protocol_suite_part_ii.html>

### 二、文件描述符（fd），文件指针（FILE\*）①摘自（程序员的自我修养）：



文件描述符fd。在内核中，每个进程都有一个私有的“打开文件表”，这个表是一个指针数组，每一个元素都指向一个内核的打开文件对象。而fd，就是这个表的下标。（当用户打开一个文件时，内核会在内部生成一个打开文件对象，并在这个表里找到一个空项，让这一项指向生成的打开文件对象，并返回这一项的下标作为fd）

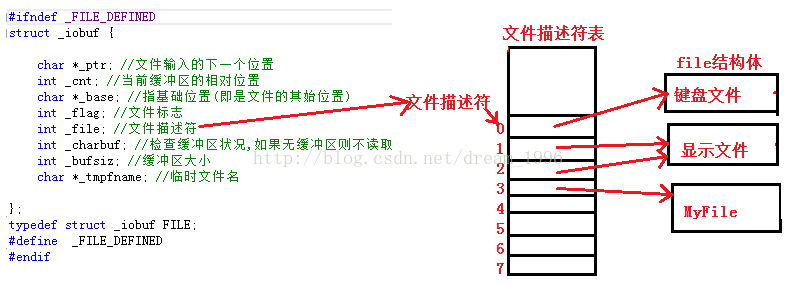


内核指针p指向该进程的打开文件表，只要有fd，通过p+fd就可以得到代开文件表的某一项地址，该地址就是fd的

### 三、文件描述符（fd），文件指针（FILE\*）②来自网络：

**文件指针：**[**C语言**](http://lib.csdn.net/base/c)中使用文件指针做为I/O的句柄。文件指针指向进程用户区中的一个被称为FILE结构的[**数据结构**](http://lib.csdn.net/base/datastructure)。FILE结构包括缓冲区和文件描述符。而文件描述符是文件描述符表的一个索引，也就是说c语言的文件指针是[**Linux**](http://lib.csdn.net/base/linux)系统中对文件描述符的一种封装。

下面看FILE结构体里面都有那些成员：



上面是文件指针指向的结构体，可以发现文件指针结构体里包含有文件描述符，说明文件指针是对文件描述符的一种封装。文件指针是C语言库里的提供的一个结构体，文件描述符是系统调用接口；

为什么系统已经有了文件描述符，库里面还要对其做一层封装呢？也是**优点:**一、方便程序员使用；二、可以提高程序的移植性。

将数据写入硬盘文件中时，缓冲区的刷新方式默认为全缓冲，

将数据写入显示文件中，缓冲区的刷新方式默认为行缓冲；

而系统调用的函数write()写入时，是没有缓冲的，是因缓冲区是C库提供的，在FILE结构体里。文件描述符和缓冲区都是FILE结构体的成员，所以文件描述符指向的file结构体里是没有缓冲区的。

**文件描述符：**在[**Linux**](http://lib.csdn.net/base/linux)系统中打开文件就会获得文件描述符，它是个很小的正整数。每个进程在PCB（Process Control Block）中保存着一份文件描述符表，文件描述符就是这个表的索引，每个表项都有一个指向已打开文件的指针，已打开的文件在内核中用file结构体表示，文件描述符表中的指针指向file结构体。

文件描述符的分配规则：从当前未被分配的最小整数处分匹配。

### 三、socket

Socket通信。（官方翻译：套接字。更好翻译：插座）

服务器就像一个大插排，包含很多插座，客户端就是像一个插头，每一个进程（线程）代表一条电线，客户端将电线的插头插到服务器插排上对应的插座上，就可以开始socket通信了。

socket是一种操作系统提供的**进程间通信机制**。

### socket

我们知道IP层的ip地址可以唯一标示主机，而TCP层协议和端口号可以唯一标示主机的一个进程，这样我们可以利用ip地址＋协议＋端口号唯一标示网络中的一个进程。

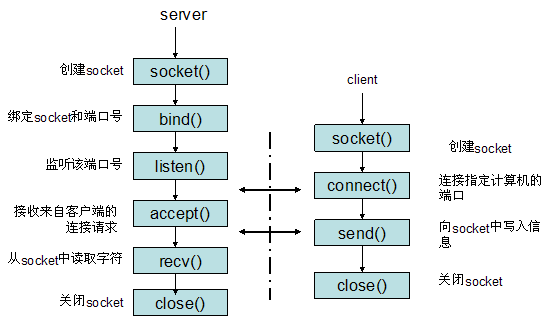
能够唯一标示网络中的进程后，它们就可以利用socket进行通信了，什么是socket呢？我们经常把socket翻译为套接字，socket是在应用层和传输层之间的一个抽象层，它把TCP/IP层复杂的操作抽象为几个简单的接口供应用层调用已实现进程在网络中通信。

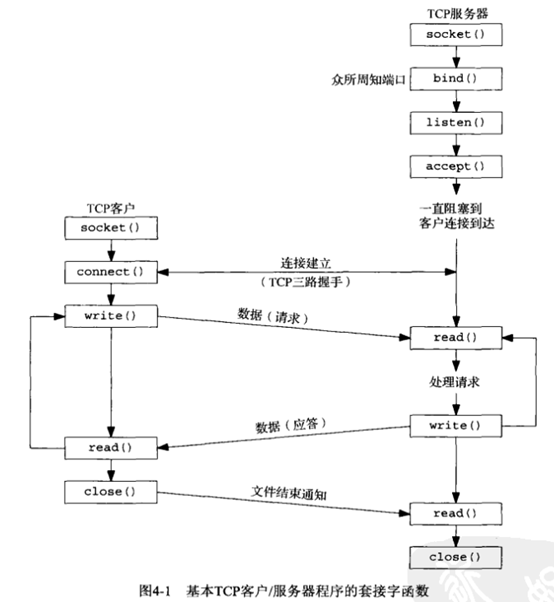


socket起源于UNIX，在Unix一切皆文件哲学的思想下，socket是一种"打开—读/写—关闭"模式的实现，服务器和客户端各自维护一个"文件"，在建立连接打开后，可以向自己文件写入内容供对方读取或者读取对方内容，通讯结束时关闭文件。

### socket通信流程

socket是"打开—读/写—关闭"模式的实现，以使用TCP协议通讯的socket为例，其交互流程大概是这样子的（二者等价）：





在TCP/IP网络应用中，通信的两个进程间相互作用的主要模式是客户/服务器（Client/Server, C/S）模式，即客户向服务器发出服务请求，服务器接收到请求后，提供相应的服务。

（1）客户与服务器进程的作用是非对称的，因此代码不同。

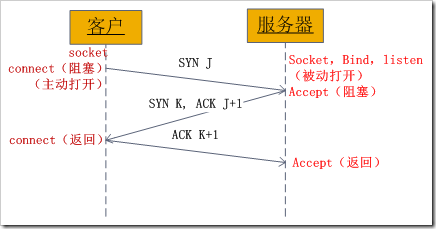
（2）服务器进程一般是先启动的。只要系统运行，该服务进程一直存在，直到正常或强迫终止。

服务器端先初始化Socket，然后与端口绑定(bind)，对端口进行监听(listen)，调用accept阻塞，等待客户端连接。

在这时如果有个客户端初始化一个Socket，然后连接服务器(connect)，如果连接成功，这时客户端与服务器端的连接就建立了。

客户端发送数据请求，服务器端接收请求并处理请求，然后把回应数据发送给客户端，客户端读取数据，最后关闭连接，一次交互结束。

定睛一看，服务器socket与客户端socket建立连接的部分其实就是大名鼎鼎的三次握手：



connect是在接收到服务端响应的SYN+ACK时的返回的，也就是三次握手的第二次动作之后。

### Socket编程api（结合.c源文件看）

1. ***int socket(int domain, int type, int protocol);（返回值：套接字fd）***

为了执行网络IO，一个进程必须做的第一件事就是调用socket函数，socket函数在成功时返回一个小的非负整数值，与文件描述符类似，成为套接字描述符

（根据指定的地址族、数据类型和协议来分配一个socket的描述字及其所用的资源。）

* domain:协议族，常用的有：

AF\_INET、(IPV4 协议)

AF\_INET6、(IPV6协议)

等

* type:socket类型，常用的socket类型有：

SOCK\_STREAM：创建TCP流套接字。提供面向连接的稳定数据传输

SOCK\_DGRAM：创建UDP数据报套接字。使用不连续不可靠的数据包连接。

SOCK\_RAW：创建原始套接字。提供原始网络协议存取。

SOCK\_PACKET：与网络驱动程序直接通信。

SOCK\_SEQPACKET：提供连续可靠的数据包连接。

SOCK\_RDM： 提供可靠的数据包连接

* protocol:协议

通常设为0,通过参数domain指定的协议族和参数type指定套接字类型来确定参数。当创建原始套接字，系统无法惟一的协议，这时候就要使用该参数来指定协议。

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

（把一个地址族中的特定地址赋给socket）

sockfd:socket描述字，

addr:要绑定给sockfd的协议地址

addrlen: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，由系统自动分配一个端口号和自身的ip地址组合。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。*

*bind函数把一个本地协议地址赋予一个套接字，它只是把一个协议地址赋予一个套接字，至于协议地址的含义则取决于协议本身。第二个参数指向协议地址结构的指针，第三个参数是协议地址的长度，对于TCP，调用bind函数可以指定一个端口号，或指定一个IP地址，或两者都指定，也可以两者都不指定。*

*bind函数绑定特定的IP地址必须属于其所在主机的网络接口之一，服务器在启动时绑定它们众所周知的端口，如果一个TCP客户端或服务端未曾调用bind绑定一个端口，当调用connect或listen时，内核就要为响应的套接字选择一个临时端口。让内核选择临时端口对于TCP客户端来说是正常的额，然后对于TCP服务端来说确实罕见的，因为服务端通过他们众所周知的端口被大家认识的。*

*（套接字地址结构，****struct sockaddr，可以先不看，或者对应.c代码看****）*

struct in\_addr {

in\_addr\_t s\_addr; // 32-bit IPv4 address

//network byte ordered

}

struct sockaddr\_in {

sa\_family\_t sin\_family; //AF\_INET

in\_port\_t sin\_port; //16-bit TCP or UDP port nummber, network byte ordered

struct in\_addr sin\_addr; //32-bit IPv4 address, network byte ordered

char sin\_zero[8]; //unused

}

*sockaddr\_in是网络套接字地址结构，大小为16字节，定义在<netinet/in>头文件中，一般我们在程序中是使用该结构体，但是作为参数传递给套接字函数时需要强转为sockaddr类型，注意该结构体中port和addr成员是网络序的(大端结构)。*

struct sockaddr {

sa\_family\_t sa\_family; //address family: AF\_XXX value

char sa\_data[14]; //protocol-specific address

}

sockaddr是套接字地址结构，当作为参数传递给套接字函数时，套接字地址结构总是以指针方式来使用。

（Linux提供了4个函数来完成主机字节序和网络字节序之间的转换。可以先不看，后续自己上网了解一下）

*htons、ntohs、htonl和ntohl函数*

这些函数名字中，h表示host，n表示net，s表示short，l表示long。

百度了解：

1. 主机字节序和网络字节序
2. inet系列函数

3、*htons、ntohs、htonl和ntohl函数*

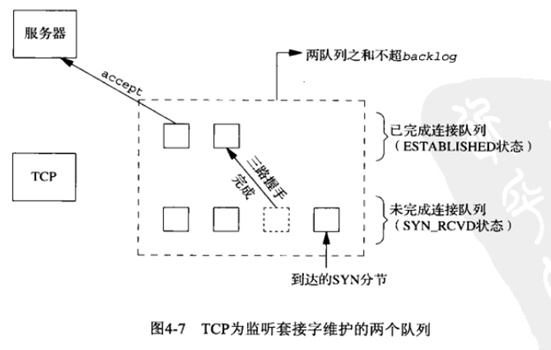
***三、int listen(int sockfd, int backlog);***

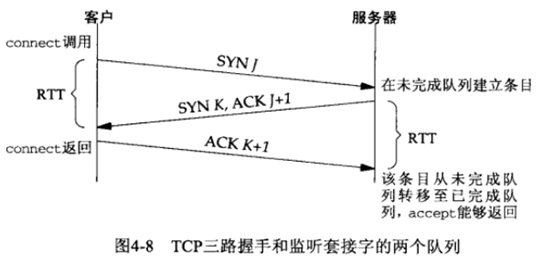
（监听socket）

sockfd:要监听的socket描述字

backlog:相应socket可以排队的最大连接个数

*socket创建一个套接字时，它被假设为一个主动套接字，也就是说，它假设是一个将调用connect发起连接的一个客户套接字。listen函数把一个未连接的套接字转换为一个被动套接字，指示内核应接受指向该套接字的连接请求，第二个参数规定了内核应为相应套接字排队的最大连接个数。*





Listen时期内核维护两个队列：未完成队列和已完成连接队列。

1. **未完成连接队列**：已由某个客户发出SYN并到达服务器，而服务器正在等待完成相应的TCP三路握手过程。这些套接字处于SYN\_RCVD状态。
2. **已完成连接队列**：每个完成TCP三路握手过程的客户对应其中一项，这些套接字处于ESTABLISHED状态。

listen()的backlog（第二个参数）应该指定某个给定套接字上内核为之排队的最大已完成连接数。

当一个客户端SYN达到时，若这些队列是满的，TCP就忽略该SYN，也即是不发送RST，这样做是暂时的，客户端过一段时间将重新发送SYN，期望能得到服务。

在三路握手完成之后，但在服务端调用accept之前到达的数据应由服务端TCP排队，最大数据量为相应已连接套接字的接收缓冲区大小（backlog）。

在TCP服务端套接字编程中，执行完listen后，而没有执行accept，客户端是可以成功建立连接的，只不过是该连接被加入到了已连接队列中，当调用accept时会被提取出来。

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

（连接某个socket）

sockfd:客户端的socket描述字

addr:服务器的socket地址

addrlen:socket地址的长度

***五、int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);***

（ TCP服务器监听到客户端请求之后，调用accept()函数取接收请求）

sockfd:服务器的socket描述字（参数sockfd就是监听套接字）

addr:客户端的socket地址

addrlen:socket地址的长度

返回值是一个连接套接字。

accept函数有TCP服务器调用，用于从已完成队列中列头返回下一个已完成连接，如果已完成队列为空，则进程被投入睡眠（如果该套接字为阻塞方式的话）。如果accept成功，那么其返回值是由内核自动生成的一个全新套接字，代表与返回客户的TCP连接，函数的第一个参数为监听套接字，返回值为已连接套接字。

此时我们需要区分两种套接字，

监听套接字: 监听套接字正如accept的参数sockfd，它是监听套接字，是服务器开始调用socket()函数生成的，在调用listen函数之后绑定完毕，称为监听socket描述字。

连接套接字：一个套接字会从主动连接的套接字变身为一个监听套接字；而accept函数返回的是已连接socket描述字(一个连接套接字)，它代表着一个网络已经存在的点和点的连接。

一个服务器通常通常仅仅只创建一个监听socket描述字，它在该服务器的生命周期内一直存在。内核为每个由服务器进程接受的客户连接创建了一个已连接socket描述字（多个，一个connect对应一个），当服务器完成了对某个客户的服务，相应的已连接socket描述字就被关闭。

（自己搜索了解下面几个函数）

建立好了TCP连接之后，我们就可以把得到的套接字当做文件描述符来使用，由此，想到了网络程序里面的基本的读写函数read和write函数。

***六、ssize\_t read(int fd, void \*buf, size\_t count);***

（读取socket内容）

fd:socket描述字

buf：缓冲区

count：缓冲区长度

***七、ssize\_t write(int fd, const void \*buf, size\_t count);***

（向socket写入内容，其实就是发送内容）

fd:socket描述字

buf：缓冲区

count：缓冲区长度  
recv和**send**函数提供了和read和**write**差不多的功能.不过它们提供了第四个参数来控制读写操作. （自己搜索了解这几个函数）

***八、int close(int fd);***

socket标记为以关闭 ，使相应socket描述字的引用计数-1，当引用计数为0的时候，触发TCP客户端向服务器发送终止连接请求。