1、int socket(int domain, int type, int protocol)；

domain：协议域，包括：AF\_INET、AF\_INET6、AF\_UNIX等。协议域决定了socket的地址类型，如，AF\_INET要用ipv4的地址和一个端口号，AF\_UNIX要用绝对路径名作为地址。

type：socket类型，常用的有：SOCK\_STREAM、SOCK\_DGRAM、SOCK\_RAW等。

protocal：协议，IPPROTO\_TCP、IPPTOTO\_UDP等。

type和protocal是不能随意组合的，当protocal为0时，会选择type对应的默认协议。

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

struct sockaddr

{

unsigned short sa\_family; //地址族， 一般为AF\_INET

char sa\_data[14]; //14字节的协议地址

}

地址结构是根据定义socket时的domain域决定的，sockaddr是一个通用的地址结构，Ipv4用下面的地址结构，传参时强制转换成sockaddr就行。

struct sockaddr\_in

{

short int sin\_family; //地址族

unsigned short int sin\_port; //端口号

struct in\_addr sin\_addr; //ip地址

unsigned char sin\_zero[8]; //填充

}

struct in\_addr {

uint32\_t s\_addr; /\* address in network byte order \*/

};

定义socket是打开了一个文件描述符，bind是为这个文件描述符绑定地址（IP+16位端口号）的。一般服务器需要绑定地址，以便客户端连接。而客户端就没必要绑定地址，

直接调用connect，随机分配一个端口和自己主机IP，客户端更关心的是目的IP和端口号。服务器在绑定的时候，in\_addr要填本机的IP地址，也可用INADDR\_ANY代替。

不过当本机有多个网卡时，需要选择一个IP进行绑定。

主机序和网络序：网络序是大端模式（高位字节在内存的低位地址，低位字节在内存的高位地址），而主机序可能是小段模式也可能是大端模式。

字节序是以字节为单位的，多个字节才有字节序的。如int型数据有4个字节，int 1；大端模式存成 0,0,0,1；小端模式存成 1,0,0,0。地址 低-->高。

比如用unsighed int s\_addr存IP地址：127.0.0.1，这是IP的点分十进制表示，转为int型大端模式表示，则s\_addr从低到高的4个字节分别存的是127,0,0,1.如果直接输出s\_addr，其值为：16777343

所以，在绑定地址时，要将地址从主机字节序转换成网络字节序(即使他们有可能相同)。

所以在netinet/in.h文件中

uint32\_t ntohl (uint32\_t \_\_netlong)；

uint16\_t ntohs (uint16\_t \_\_netshort)；

uint32\_t htonl (uint32\_t \_\_hostlong)；//将主机一个32位整数(4个字节的字节序)转换成网络的一个32位整数。

uint16\_t htons (uint16\_t \_\_hostshort)；//将主机一个16位整数(2个字节的字节序)转换成网络的一个16位整数。比如给端口号赋值时。

这4个函数用于主机和网络的整数之间的转换。

arpa/inet.h

int inet\_pton (int \_\_af, const char \* src,void \*des);

这个函数是将IP地址转换成一个32位整数，即网络序的IP地址。

\_\_af为AF\_INET时：src为点分十进制的IP地址，如127.0.0.1。des为sockaddr\_in.in\_addr类型(网络序)。

netdb.h

struct hostent \*gethostbyname (const char \*\_\_name);通过域名获取IP地址

struct hostent

{

char \*h\_name;

char \*\*h\_aliases;

int h\_addrtype;

int h\_length;

char \*\*h\_addr\_list;

#define h\_addr h\_addr\_list[0]

};

h\_addr\_list存放的就是网络字节序的IP地址(可能有多个，一般选择第一个)

sockaddr\_in server\_addr;

server\_addr.sin\_addr = \*((struct in\_addr\*)hostent.h\_addr);

3、

int listen(int sockfd, int backlog);

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

listen将socket变为被动监听，backlog为最大连接数。

connect 主动连接目的IP+端口号。客户端使用。

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

返回值是一个连接套接字，而sockfd参数是监听套接字。后两个参数是客户端的地址和大小，如果对客户端地址不感兴趣可设为NULL。

accept是一个阻塞函数，用于完成TCP连接。

5、数据通信函数

#include <unistd.h>

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

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

#include <sys/types.h>

#include <sys/socket.h>

ssize\_t send(int sockfd, const void \*buf, size\_t len, int flags);

ssize\_t recv(int sockfd, void \*buf, size\_t len, int flags);

6、

#include <unistd.h>

int close(int fd);

一般是客户端主动关闭连接。

头文件小结：

#include <sys/types.h> 一些常用类型的定义

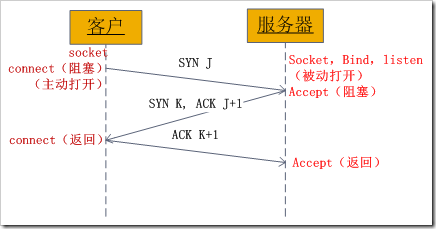
#include <sys/socket.h> 包含socket常用函数socket，bind，listen，connect，accept，send，recv等

#include <netinet/in.h> 包含sockaddr\_in、in\_addr数据结构和htonl、htons等函数。

#include <arpa/inet.h> 有inet\_pton等。

#include <netdb.h> 有gethostbyname等。

三次握手连接



调用listen之后，监听端口就打开了，这时客户端只要连接就能显示连接成功，即只要客户端发过来SYN，服务器就回SYN、ACK。然后客户端再回ACK，连接建立。所以，服务器掉没掉用accept，客户端都认为已经建立连接了。然后客户端开始发送数据。

我觉得上图有些问题，看着好像是Accept返回的SYN、ACK似得，其实TCP三次握手和Accept没关系。Accept的作用是通过监听套接字产生一个连接套接字，所以，调用Accept时，Accept会检查监听套接字监听的端口有没有TCP连接，如果有的话，就产生连接套接字，连接套接字用来与客户端传输数据；没有的话，就阻塞。只不过传输数据用的都是连接套接字，所以觉得调用Accept才是完成TCP连接，其实不是，Accept只是从已经存在的TCP连接上产生一个连接套接字，用于进行数据传输。

Socket的工作模式是，listen之后就能接受TCP连接，Accept之后可以进行数据传输。

四次握手断开

