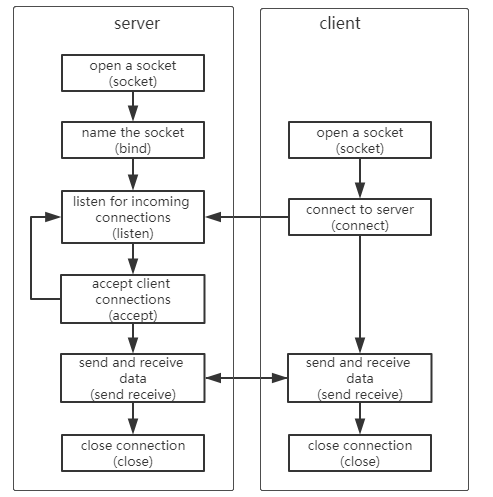
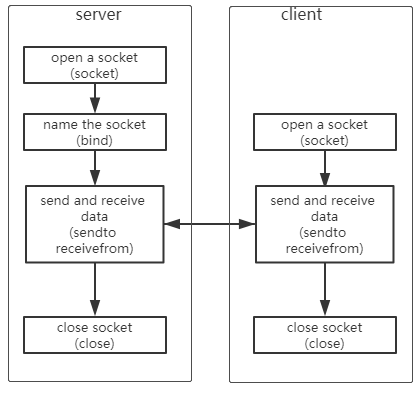
##### TCP/UDP网络通信大概交互图

下面我们分别画出运用TCP协议和运用UDP协议的客户端和服务器大概交互图。



面向连接的TCP流模式



UDP用户数据包模式

### 8.2 网络编程主要函数介绍

#### 8.2.1 socket函数

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

此函数用于创建一个套接字。

**domain**是网络程序所在的主机采用的通讯协族(AF\_UNIX和AF\_INET等)。

AF\_UNIX只能够用于单一的Unix 系统进程间通信，而AF\_INET是针对Internet的，因而可以允许远程通信使用。

**type**是网络程序所采用的通讯协议(SOCK\_STREAM,SOCK\_DGRAM等)。

SOCK\_STREAM表明用的是TCP 协议，这样会提供按顺序的，可靠，双向，面向连接的比特流。

SOCK\_DGRAM 表明用的是UDP协议，这样只会提不可靠，无连接的通信。

关于**protocol**，由于指定了type，所以这个地方一般只要用0来代替就可以了。

此函数执行成功时返回文件描述符，失败时返回-1,看errno可知道出错的详细情况。

#### 8.2.2 bind函数

int bind(int sockfd, struct sockaddr \*my\_addr, int addrlen);

从函数用于将地址绑定到一个套接字。

**sockfd**是由socket函数调用返回的文件描述符。

**my\_addr**是一个指向sockaddr的指针。

**addrlen**是sockaddr结构的长度。

sockaddr的定义：

struct sockaddr{

unisgned short  as\_family;

char sa\_data[14];

};

不过由于系统的兼容性,我们一般使用另外一个结构(struct sockaddr\_in) 来代替。

sockaddr\_in的定义：

struct sockaddr\_in{

unsigned short   sin\_family;

unsigned short sin\_port;

struct in\_addr   sin\_addr;

unsigned char   sin\_zero[8];

}

如果使用Internet所以sin\_family一般为AF\_INET。

sin\_addr设置为INADDR\_ANY表示可以和任何的主机通信。

sin\_port是要监听的端口号。

bind将本地的端口同socket返回的文件描述符捆绑在一起.成功是返回0,失败的情况和socket一样。

#### 8.2.3 listen函数

int listen(int sockfd,int backlog);

此函数宣告服务器可以接受连接请求。

**sockfd**是bind后的文件描述符。

**backlog**设置请求排队的最大长度。当有多个客户端程序和服务端相连时，使用这个表示可以介绍的排队长度。

listen函数将bind的文件描述符变为监听套接字，返回的情况和bind一样。

#### 8.2.4 accept函数

int accept(int sockfd, struct sockaddr \*addr,int \*addrlen);

服务器使用此函数获得连接请求，并且建立连接。

**sockfd**是listen后的文件描述符。

**addr，addrlen**是用来给客户端的程序填写的,服务器端只要传递指针就可以了， bind,listen和accept是服务器端用的函数。

accept调用时，服务器端的程序会一直阻塞到有一个客户程序发出了连接。 accept成功时返回最后的服务器端的文件描述符，这个时候服务器端可以向该描述符写信息了，失败时返回-1 。

#### 8.2.5 connect函数

int connect(int sockfd, struct sockaddr \* serv\_addr,int addrlen);

可以用connect建立一个连接，在connect中所指定的地址是想与之通信的服务器的地址。

**sockfd**是socket函数返回的文件描述符。

**serv\_addr**储存了服务器端的连接信息，其中sin\_add是服务端的地址。

**addrlen**是serv\_addr的长度

connect函数是客户端用来同服务端连接的.成功时返回0，sockfd是同服务端通讯的文件描述符，失败时返回-1。

#### 8.2.6 send函数

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

**sockfd** 指定发送端套接字描述符；

**buf** 指明一个存放应用程序要发送数据的缓冲区；

**len** 指明实际要发送的数据的字节数；

**flags** 一般置0。

客户或者服务器应用程序都用send函数来向TCP连接的另一端发送数据

#### 8.2.7 recv函数

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

**sockfd** 指定接收端套接字描述符；

**buf** 指明一个缓冲区，该缓冲区用来存放recv函数接收到的数据；

**len** 指明buf的长度；

**flags** 一般置0。

客户或者服务器应用程序都用recv函数从TCP连接的另一端接收数据。

#### 8.2.8 recvfrom函数

ssize\_t recvfrom(int sockfd, void \*buf, size\_t len, int flags,

struct sockaddr \*src\_addr, socklen\_t \*addrlen);

recvfrom通常用于无连接套接字，因为此函数可以获得发送者的地址。

**src\_addr** 是一个struct sockaddr类型的变量，该变量保存源机的IP地址及端口号。

**addrlen** 常置为sizeof （struct sockaddr）。

#### 8.2.9 sendto函数

ssize\_t sendto(int sockfd, const void \*buf, size\_t len, int flags,

const struct sockaddr \*dest\_addr, socklen\_t addrlen);

sendto和send相似，区别在于sendto允许在无连接的套接字上指定一个目标地址。

**dest\_addr** 表示目地机的IP地址和端口号信息，

**addrlen** 常常被赋值为sizeof （struct sockaddr）。

**sendto** 函数也返回实际发送的数据字节长度或在出现发送错误时返回-1。