第9章 Linux网络编程

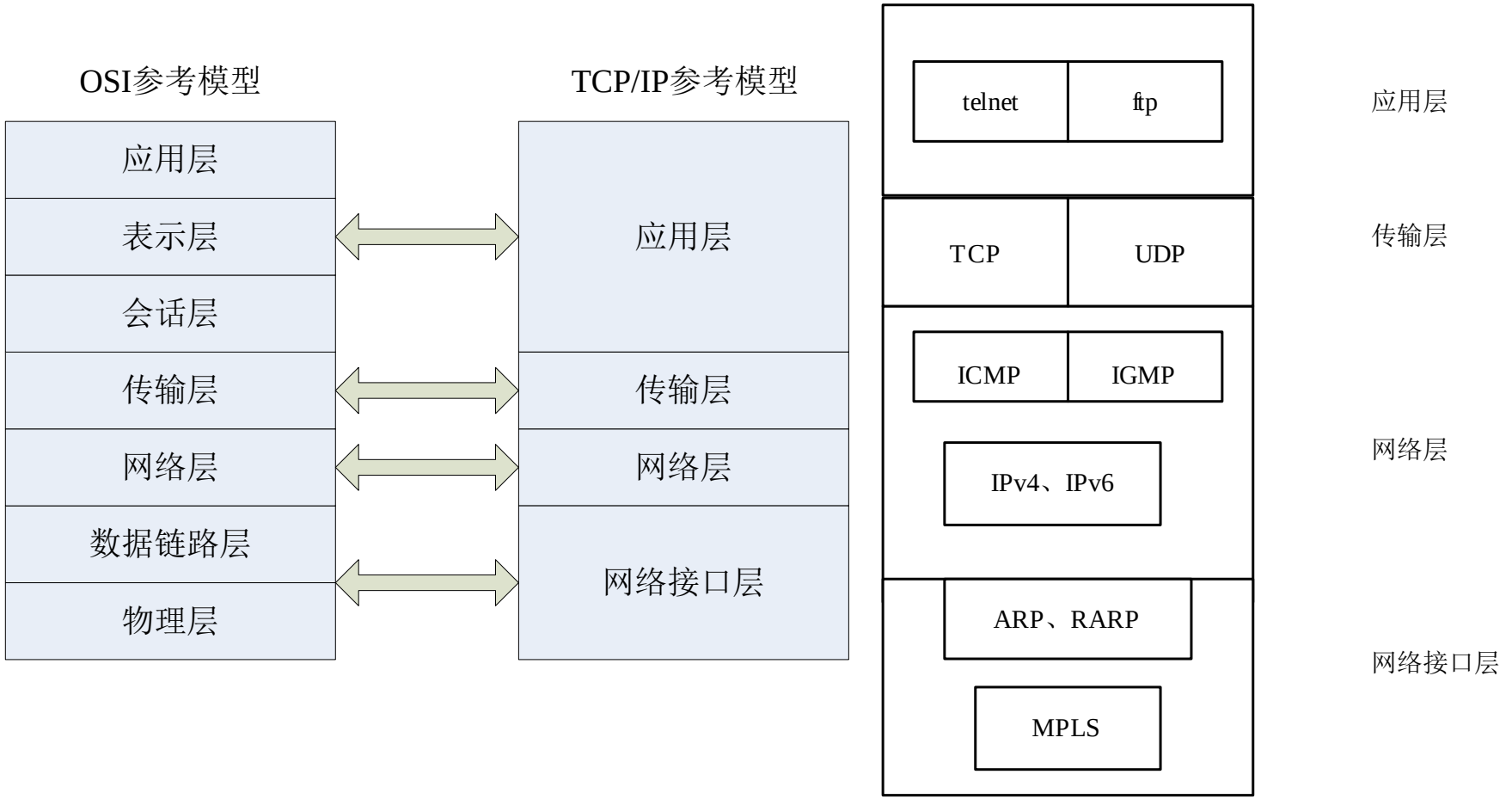
9.1 TCP/IP协议概述

9.2 网络基础编程

9.1 TCP/IP协议概述

- OSI参考模型及TCP/IP参考模型
- TCP
- UDP
- 协议的选择

OSI参考模型及TCP/IP参考模型



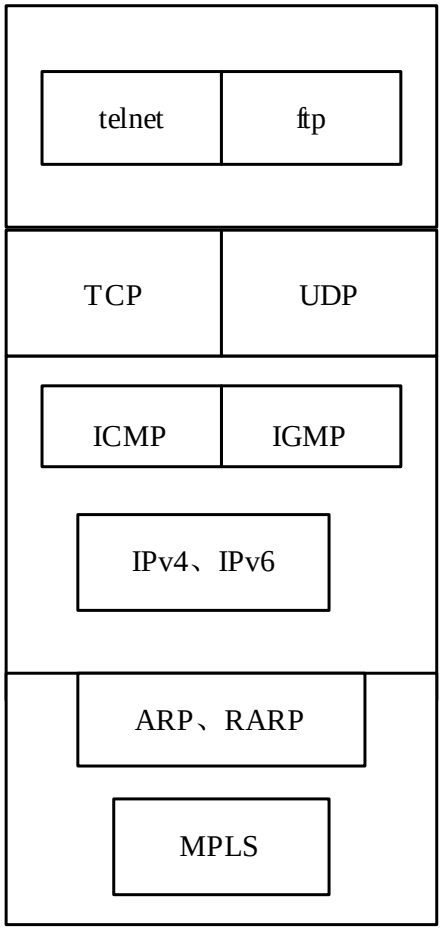
OSI参考模型及TCP/IP参考模型

负责应用程序的网络访问，通过端口号来识别各个不同的进程

负责端对端之间的通信会话连接与建立。传输协议的选择根据数据传输方式而定

负责将数据帧封装成IP数据包，并运行必要的路由算法。

负责将二进制流转换为数据帧，并进行数据帧的发送和接收。数据帧是独立的网络信息传输单元



应用层

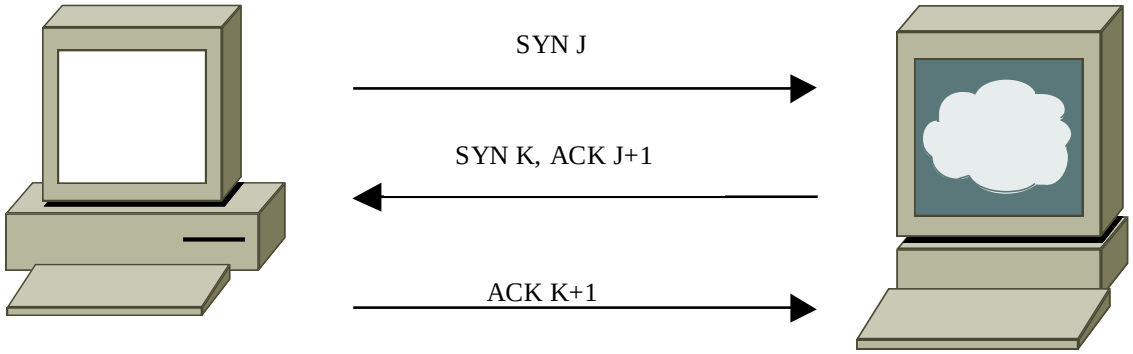
传输层

网络层

网络接口层

TCP

- TCP（传输控制协议）：为应用程序提供可靠的通信连接，建立一次连接需三次握手，适合于一次传输大批数据的情况。



UDP

- UDP（用户数据报协议）：是一种无连接协议，不需要像TCP那样通过三次握手来建立一个连接。同时，一个UDP应用可同时作为客户方或服务器方。
- UDP比TCP能更好地解决实时性的问题，包括网络视频会议系统在内的众多的客户/服务器模式的网络应用都使用UDP协议。

协议的选择

- **对数据可靠性的要求**：高可靠性要求的应用需选择TCP协议，如验证、密码字段的传送都是不允许出错的，而对数据的可靠性要求不那么高的应用可选择UDP传送
- **应用的实时性**：TCP协议在传送过程中要使用三次握手、重传确认等手段来保证数据传输的可靠性。使用TCP协议会有较大的时延，因此不适合对实时性要求较高的应用，如VOIP、视频监控等。相反，UDP协议则在这些应用中能发挥很好的作用。
- **网络的可靠性**：由于TCP协议的提出主要是解决网络的可靠性问题，它通过各种机制来减少错误发生的概率。因此，在网络状况不是很好的情况下需选用TCP协议（如在广域网等情况），在网络状况很好的情况下（如局域网等）应选择UDP协议来减少网络负荷。

9.2 网络编程基础

- 套接字socket：
  - 套接字是一种进程间通信的方法，不同于以往介绍的进程间通信方法的是，它并不局限于同一台计算机的资源，例如共享内容或者消息队列。
  - 套接字接口（socket interface）由伯克利版本UNIX引入，可以认为是对管道概念的扩展，一台机器上的进程可以使用套接字与另一台机器上的进程通信。因此socket适用于网络中不同机器间的通信。
  - 同一台机器的进程间也可以用套接字通信。
  - 微软的windows系统也实现了套接字接口，因此windows程序可以通过网络和Linux/UNIX计算机进行通信，实现客户/服务器系统。

socket工作过程-服务器（1）

- 首先，服务器应用程序通过socket系统调用创建一个套接字，它是系统分配给该服务器进程的类似文件描述符的资源，不能与其他进程共享。
- 其次，服务器进程使用bind系统调用给套接字命名。
  - 本地套接字的名字是linux文件系统的文件名，一般将其放在/tmp或者/usr/tmp目录下。
  - 网络套接字的名字是与客户相连接的特定网络有关的服务标识符。此标识符允许linux将进入的针对特定端口号的连接转到正确的服务器进程。

socket工作过程-服务器（2）

- 接下来，服务器进程开始等待客户连接到这个命名套接字，调用listen创建一个等待队列，以便存放来自客户的连接。
- 最后，服务器通过accept系统调用来接受客户的连接。此时，会产生一个 与原有的命名套接字不同的新套接字，它仅用于与这个特定的客户通信，而命名套接字则被保留下来继续处理来自其他客户的连接。

socket工作过程-客户端

- 调用socket创建一个未命名套接字，将服务器的命名套接字作为一个地址来调用connect与服务器建立连接。
- 一旦建立了连接，就可以像使用底层文件描述符那样来用套接字进行双向的数据通信。

套接字属性

- 套接字的特性由三个属性决定：
  - 域（domain）
  - 类型（type）
  - 协议（protocol）

socket域

- socket域：指定套接字通信中使用的网络介质，包括地址格式。主要包括：
  - AF_INET：即互联网络，基于IP协议，并且每个服务对应一个端口号，套接字地址由IP地址+端口号决定；
  - AF_INET6：适用于IPV6网络
  - AF_UNIX：基于本地机器，底层协议使用文件输入/输出，地址为绝对路径的文件名。

套接字类型

- 因特网协议提供了两种通信协议，流(stream)和数据报(datagram)。
  - SOCK_STREAM： 流套接字可以提供有序、可靠的、面向连接的通讯流。发送的数据可以确保不会丢失、复制或乱序到达。它使用了TCP协议。TCP保证了数据传输的正确性和顺序性。
  - SOCK_DGRAM：数据报套接字定义了一种无连接的服务，数据通过相互独立的报文进行传输，是无序的，并且不保证可靠，无差错。使用数据报协议UDP协议。
- AF_UNIX域的套接字没有通信方面的问题，因为其基于文件系统，提供了一个可靠的双向通信路径。

协议类型

- 通常为0，表示按给定的域和套接字类型选择默认协议

socket接口函数

- 创建套接字：socket系统调用创建一个套接字，并返回一个描述符，该描述符可以用来访问这个套接字。创建的套接字是一条通信链路的一个端点。

创建套接字：

#include <sys/socket.h>

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

返回值：成功返回描述符，出错返回-1

套接字地址

- 每个套接字域都有自己的地址格式。
  - AF_UNIX：地址格式由sockaddr_un来描述

```
struct sockaddr_un
{
    sa_family_t sun_family;      //AF_UNIX
    char sun_path[];             //pathname
}
```

套接字地址

- 在AF_INET域中，套接字地址由sockaddr_in来指定，该结构在头文件netinet/in.h中。

```
struct in_addr{
    in_addr_ts_addr    /*ipv4地址*/(unsigned int)
};
struct sockaddr_in {
    short int   sin_family;                 /* Internet地址族 */
    unsigned short int   sin_port;          /* 端口号 */
    struct in_addr    sin_addr;             /* Internet地址 */
    unsigned char    sin_zero[8];           /* 添0
                                （和struct sockaddr一样大小）*/
};
```

通用地址格式

```
struct sockaddr {
        unsigned short     sa_family;   /* address族, AF_xxx */
        char     sa_data[14];              /* 14 bytes的协议地址 */
};
```

- sa_data 包含了一些远程电脑的地址、端口和套接字的数目，它里面的数据是杂溶在一起的。
- sa_family一般来说，IPV4使用AF_INET
- 这两个数据类型是等效的，可以相互转换，通常使用sockaddr_in更为方便。

地址格式转换

- Linux提供将点分格式的地址与长整型数之间的转换函数。inet_addr()能够把一个用数字和点表示的IP地址的字符串转换成一个无符号整数。

#include <arp/inet.h>
const char *inet_ntop(int domain, const void *restrict addr,
char *restrict str, socklen_t size);
成功返回地址字符串指针，出错返回NULL
功能：将“整数” —>“点分十进制”
int inet_pton(int domain, const char *restrict str, void *restrict addr);
成功返回1，无效格式返回0，出错返回-1
这个函数转换字符串到网络地址,转换后存在addr中(in_addr结构体)

套接字与地址绑定

- 要想通过socket调用创建的套接字可以被其他进程使用，必须将该套接字命名，对于**AF_UNIX**套接字才会关联到一个文件系统的路径名上；对于AF_INET套接字就是关联到一个IP端口号，这个过程称为绑定。

```
#include  <sys/socket.h>
int bind(int sockfd, const struct sockaddr  *addr,  socklen_t len);
成功返回0，出错返回-1
```

创建套接字队列

- 为了能够在套接字上接受进入的连接，服务器程序必须创建一个队列来保存未处理的请求，使用listen来完成这一工作。
  - 当服务器正忙于处理一个客户请求时，后续的客户连接放入队列等待处理。函数执行成功返回0，失败返回-1.
  - 套接字队列中，等待处理的进入连接的个数最多不能超过backlog这个数字，多出的连接请求将被拒绝，导致客户连接失败。

```
#include <sys/socket.h>
int  listen(int sockfd, int backlog);
成功返回0，出错返回-1
//Linux系统对队列中可以容纳的未处理的连接的数量做出限制，由backlog设定。backlog常用值为5
```

接受连接

- 一旦服务器程序创建了命名套接字之后，它就可以通过accept来等待客户建立连接。

#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *restrict addr, socklen_t *restrict len);
成功返回一个新的套接字文件描述符，出错返回-1
注意：accept系统调用只有当有客户程序试图连接到由sockfd参数指定的套接字上时才返回，accept函数将创建一个新的套接字来与该客户进行通信，并且返回新套接字的描述符。新套接字的类型和服务器监听套接字类型相同。

- 如果套接字队列中没有未处理的连接，accept将阻塞直到有客户建立连接为止。

请求连接

- 客户程序通过在一个未命名套接字和服务器监听套接字之间建立连接的方法连接到服务器。通过connect调用来完成这一工作。

客户端：

#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *addr, socklen_t len)

成功返回0，出错返回-1

说明：如果连接不能立刻建立，connect调用将阻塞一段不确定的时间，称为超时时间，一旦超过这个时间连接将放弃。

关闭套接字

- 通过调用close来终止服务器和客户上的套接字连接，同文件的关闭。
- 应该在连接的两端都关闭套接字。

数据传输-发送和接收

发送：

```
#include <sys/socket.h>
ssize_t  send(int  sockfd, const void *buf, size_t nbytes, int flags);
成功返回发送字节数，出错返回-1
ssize_t  sendto(int sockfd, const void *buf, size_t nbytes, int flags,
const  struct sockaddr *destaddr, socklen_t destlen);
成功返回发送的字节数，出错返回-1
```

接收：

```
#include <sys/socket.h>

ssize_t  recv(int sockfd, void *buf, size_t  nbytes, int flags);
ssize_t recvfrom(int sockfd, void *restrict buf, size_t len, int flags,
struct sockaddr *restrict addr,  socklen_t  *restrict addrlen);
返回消息的字节数，无消息返回0，出错返回-1
```

套接字选项

- 你可以用许多选项来控制套接字连接的行为，通过函数setsockopt函数来控制连接选项，选项数目众多，需要时查阅资料。

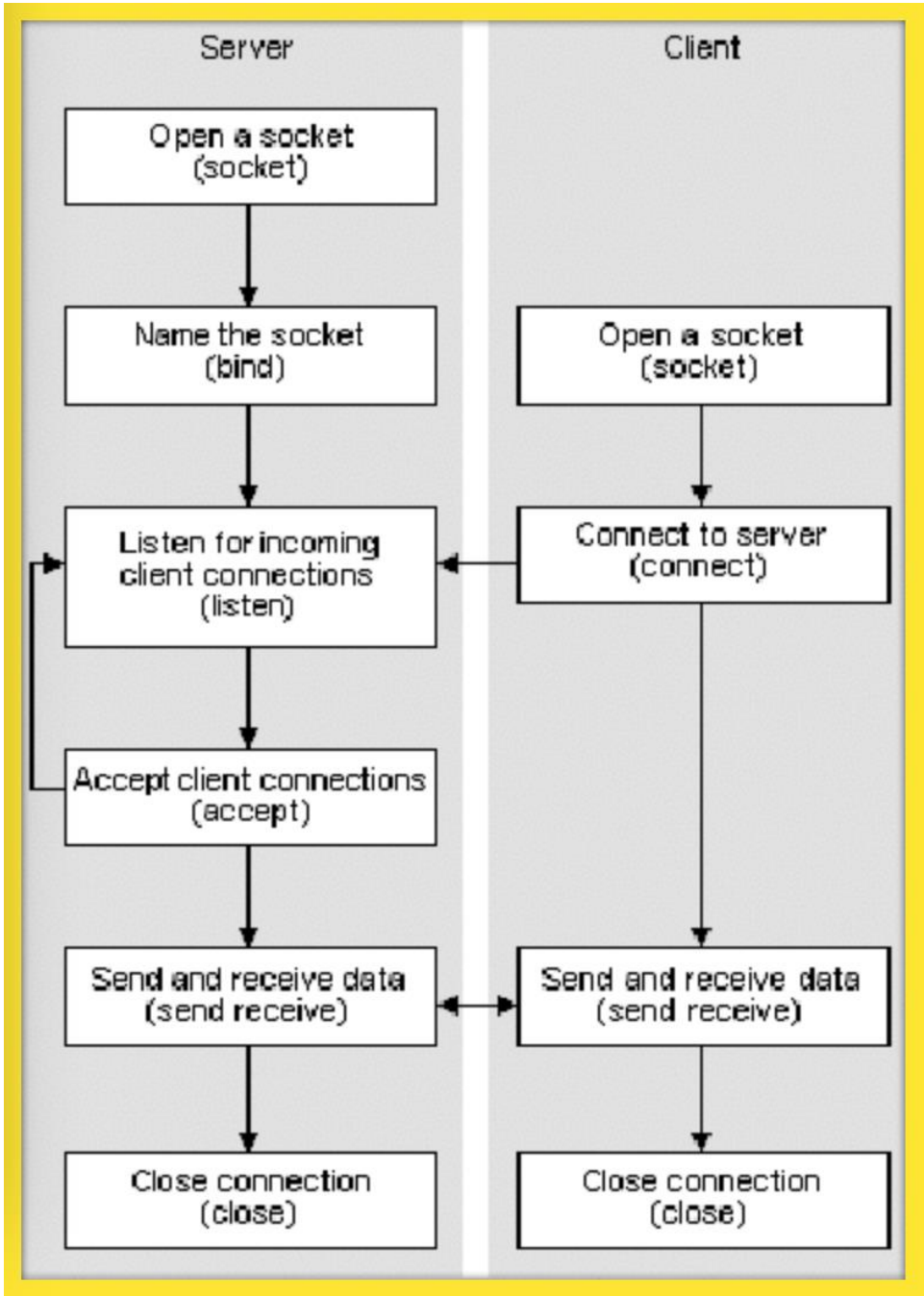
```
#include <sys/socket.h>
int setsockopt(int socket, int level, int option_name,
               const void *option_value, size_t option_len);
```

设置成功返回0，失败返回-1

主机字节序和网络字节序

- 因为每一个机器内部对变量的字节存储顺序不同(有的系统是高位在前，低位在后；而有的系统是低位在前，高位在后)，而网络传输的数据是一定要统一顺序的。所以对于内部字节表示顺序和网络字节顺序不同的机器，一定要对数据进行转换。
  - htons()——“Host to Network Short” 主机字节顺序转换为网络字节顺序（对无符号短型进行操作2bytes)
  - htonl()——“Host to Network Long” 主机字节顺序转换为网络字节顺序（对无符号长型进行操作4bytes)
  - ntohs()——“Network to Host Short” 网络字节顺序转换为主机字节顺序（对无符号短型进行操作2bytes)
  - ntohl()——“Network to Host Long ” 网络字节顺序转换为主机字节顺序（对无符号长型进行操作4bytes)

TCP 服务器 客户机流程



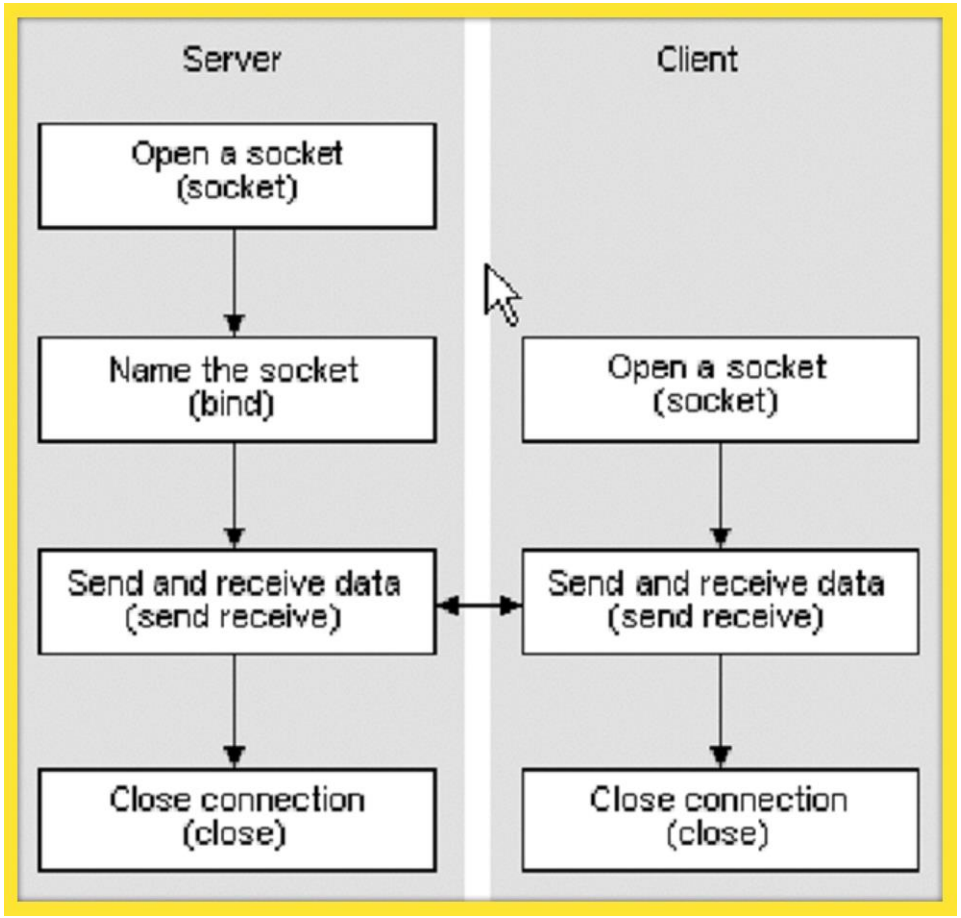
服务器程序流程：

①程序初始化
②填写本机地址信息
③绑定并监听一个固定的端口
④收到Client的连接后建立一个socket连接
⑤产生一个新的进程与Client进行通信和信息处理
⑥子通信结束后中断与Client的连接

客户端程序流程：

①程序初始化
②填写服务器地址信息
③连接服务器
④与服务器通信和信息处理
⑤通信结束后断开连接

UDP服务器-客户机流程



服务器程序流程：

①程序初始化
②填写本机地址信息
③绑定一个固定的端口
④收到Client的数据报后进行处理与通信
⑤通信结束后断开连接

客户端程序流程：

①程序初始化
②填写服务器地址信息
③连接服务器
④与服务器通信和信息处理
⑤通信结束后断开连接

TCP与UDP的区别

1基于连接与无连接

2对系统资源的要求（TCP较多，UDP少）

3UDP程序结构较简单

4流模式与数据报模式

5TCP保证数据正确性，UDP可能丢包

6TCP保证数据顺序，UDP不保证

具体编程时的区别

1socket()的参数不同

2UDP Server不需要调用listen和accept

3UDP收发数据用sendto/recvfrom函数

4TCP：地址信息在connect/accept时确定

5UDP：在sendto/recvfrom函数中每次均 需指定地址信息

6UDP：shutdown函数无效

部分满足以下几点要求时，应该采用UDP 面向数据报方式

1网络数据大多为短消息

2拥有大量Client

3对数据安全性无特殊要求

4网络负担非常重，但对响应速度要求高