# 目录

[目录 1](#_Toc22291998)

[第12章 网络编程 2](#_Toc22291999)

[12.1 网络编程介绍 2](#_Toc22292000)

[12.1.1 OSI参考模型及TCP/IP参考模型 2](#_Toc22292001)

[12.1.2 TCP协议 2](#_Toc22292002)

[12.1.3 UDP协议 2](#_Toc22292003)

[12.1.4 协议的选择 2](#_Toc22292004)

[12.2 网络编程相关概念 2](#_Toc22292005)

[12.2.1 IP地址的表示 2](#_Toc22292006)

[12.2.2 IPv4和IPv6 2](#_Toc22292007)

[12.2.3 端口号的概念 2](#_Toc22292008)

[12.2.4 socket概念 2](#_Toc22292009)

[12.2.5 socket类型 2](#_Toc22292010)

[12.2.6 Linux网络相关命令 2](#_Toc22292011)

[12.3 TCP网络编程 2](#_Toc22292012)

[12.3.1 使用TCP协议的流程图 2](#_Toc22292013)

[12.3.2 相关API函数介绍 2](#_Toc22292014)

[12.3.3 TCP示例 2](#_Toc22292015)

[12.4 UDP网络编程 2](#_Toc22292016)

[12.4.1 UDP框图 2](#_Toc22292017)

[12.4.2 UDP相关函数 2](#_Toc22292018)

[12.4.3 UDP示例 2](#_Toc22292019)

# 网络编程

## 网络编程介绍

### OSI参考模型及TCP/IP参考模型

OSI(Open System Interconnect)开放式系统互联。 一般都叫OSI参考模型 是ISO(国际标准化组织)组织在1985年研究的网络互联模型。

其模型如图12.1和图12.2所示：

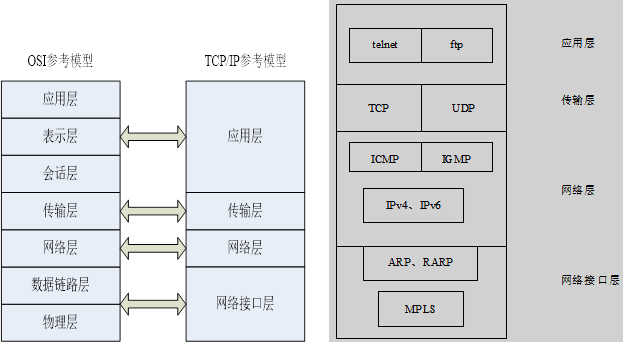


图12.1 图12.2

TCP/IP协议族的每一层的作用：

网络接口层：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。要注意的是数据帧是独立的网络信息传输单元。

网络层：负责将数据帧封装成IP数据报，并运行必要的路由算法。

传输层：负责端对端之间的通信会话连接和建立。传输协议的选择根据数据传输方式而定。

应用层：负责应用程序的网络访问，这里通过端口号来识别各个不同的进程。

TCP/IP协议族的每一层协议的相关注解如下，各层关系如图12.3和图12.4所示。

ARP：（地址转换协议）用于获得同一物理网络中的硬件主机地址。

MPLS：（多协议标签交换）很有发展前景的下一代网络协议。

IP：（网际互联协议）负责在主机和网络之间寻址和路由数据包。

ICMP：（网络控制消息协议）用于发送报告有关数据包的传送错误的协议。

IGMP：（网络组管理协议）被IP主机用来向本地多路广播路由器报告主机组成员的协议。

TCP：（传输控制协议）为应用程序提供可靠的通信连接。适合于一次传输大批数据的情况。并适用于要求得到相应的应用程序。

UDP：（用户数据包协议）提供了无连接通信，且不对传送包进行可靠的保证。适合于一次传输少量数据。

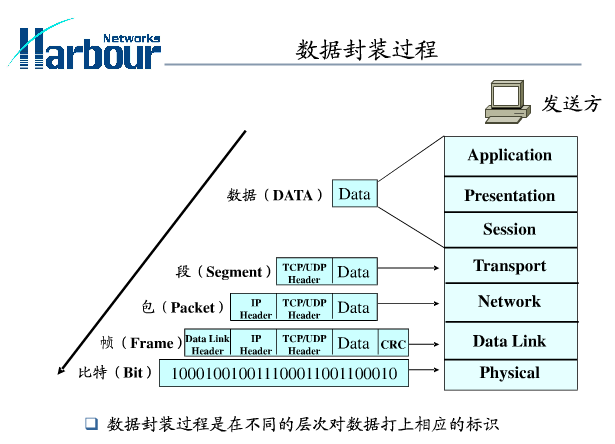


图12.3

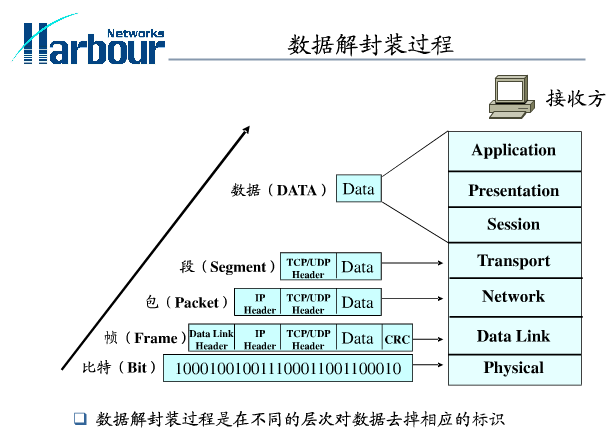


图12.4

### TCP协议

#### 概述

TCP是TCP/IP体系中面向连接的运输层协议，它提供全双工和可靠交付的服务。它采用许多机制来确保端到端结点之间的可靠数据传输，如采用序列号、确认重传、滑动窗口等。

首先，TCP要为所发送的每一个报文段加上序列号，保证每一个报文段能被接收方接收，并只被正确的接收一次。

其次，TCP采用具有重传功能的积极确认技术作为可靠数据流传输服务的基础。这里“确认”是指接收端在正确收到报文段之后向发送端回送一个确认（ACK）信息。发送方将每个已发送的报文段备份在自己的缓冲区里，而且在收到相应的确认之前是不会丢弃所保存的报文段的。“积极”是指发送发在每一个报文段发送完毕的同时启动一个定时器，加入定时器的定时期满而关于报文段的确认信息还没有达到，则发送发认为该报文段已经丢失并主动重发。为了避免由于网络延时引起迟到的确认和重复的确认，TCP规定在确认信息中捎带一个报文段的序号，使接收方能正确的将报文段与确认联系起来。

最后，采用可变长的滑动窗口协议进行流量控制，以防止由于发送端与接收端之间的不匹配而引起的数据丢失。这里所采用的滑动窗口协议与数据链路层的滑动窗口协议在工作原理上完全相同，唯一的区别在于滑动窗口协议用于传输层是为了在端对端节点之间实现流量控制，而用于数据链路层是为了在相邻节点之间实现流量控制。TCP采用可变长的滑动窗口，使得发送端与接收端可根据自己的CPU和数据缓存资源对数据发送和接收能力来进行动态调整，从而灵活性更强，也更合理。

#### 三次握手协议

在利用TCP实现源主机和目的主机通信时，目的主机必须同意，否则TCP连接无法建立。为了确保TCP连接的成功建立，TCP采用了一种称为三次握手的方式，三次握手方式使得“序号/确认号”系统能够正常工作，从而使它们的序号达成同步。如果三次握手成功，则连接建立成功，可以开始传送数据信息。

其三次握手分别为：

1）源主机A的TCP向主机B发送连接请求报文段，其首部中的SYN（同步）标志位应置为1，表示想跟目标主机B建立连接，进行通信，并发送一个同步序列号X（例：SEQ=100）进行同步，表明在后面传送数据时的第一个数据字节的序号为X+1（即101）。

2）目标主机B的TCP收到连接请求报文段后，如同意，则发回确认。再确认报中应将ACK位和SYN位置为1.确认号为X+1，同时也为自己选择一个序号Y。

3）源主机A的TCP收到目标主机B的确认后要想目标主机B给出确认。其ACK置为1，确认号为Y+1，而自己的序号为X+1。TCP的标准规定，SYN置1的报文段要消耗掉一个序号。

运行客户进程的源主机A的TCP通知上层应用进程，连接已经建立。当源主机A向目标主机B发送第一个数据报文段时，其序号仍为X+1，因为前一个确认报文段并不消耗序号。

当运行服务进程的目标主机B的TCP收到源主机A的确认后，也通知其上层应用进程，连接已经建立。至此建立了一个全双工的连接，如图12.5所示。

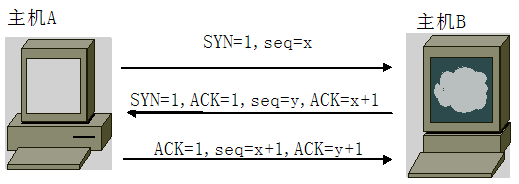


图12.5

三次握手：为应用程序提供可靠的通信连接。适合于一次传输大批数据的情况。并适用于要求得到响应的应用程序。

TCP数据报头如图12.6所示

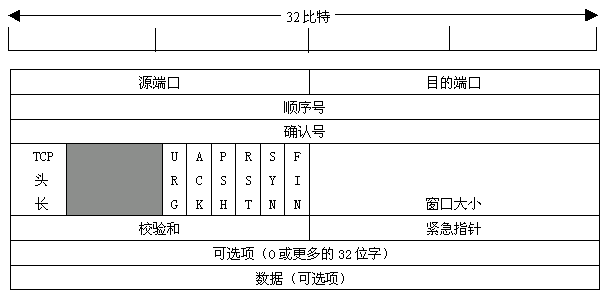


图12.6 TCP头信息

源端口、目的端口：16位长。标识出远端和本地的端口号。

序号：32位长。标识发送的数据报的顺序。

确认号：32位长。希望收到的下一个数据报的序列号。

TCP头长：4位长。表明TCP头中包含多少个32位字。

6位未用。

ACK：ACK位置1表明确认号是合法的。如果ACK为0，那么数据报不包含确认信息，确认字段被省略。

PSH：表示是带有PUSH标志的数据。接收方因此请求数据报一到便可送往应用程序而不必等到缓冲区装满时才发送。

RST：用于复位由于主机崩溃或其他原因而出现的错误的连接。还可以用于拒绝非法的数据报或拒绝连接请求。

SYN：用于建立连接。

FIN：用于释放连接。

窗口大小：16位长。窗口大小字段表示在确认了字节之后还可以发送多少个字节。

校验和：16位长。是为了确保高可靠性而设置的。它校验头部、数据和伪TCP头部之和。

可选项：0个或多个32位字。包括最大TCP载荷，窗口比例、选择重复数据报等选项。

### UDP协议

#### 概述

UDP即用户数据报协议，它是一种无连接协议，因此不需要像TCP那样通过三次握手来建立一个连接。同时，一个UDP应用可同时作为应用的客户或服务器方。由于UDP协议并不需要建立一个明确的连接，因此建立UDP应用要比建立TCP应用简单得多。

它比TCP协议更为高效，也能更好地解决实时性的问题。如今，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。

#### Udp数据包头格式如图8.7所示。

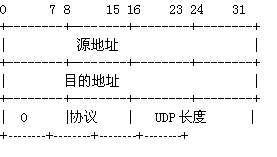


图8.7

### 协议的选择

TCP和UDP的区别

1. **对数据可靠性的要求**

对数据要求高可靠性的应用需选择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。

1. **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。

1. **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），但是若在网络状况很好的情况下（如局域网等）就不需要再采用TCP协议，而建议选择UDP协议来减少网络负荷。

## 网络编程相关概念

### IP地址的表示

在同一个网络，这些网络设备的IP地址互不相同。

通常用户在表达IP地址时采用的是点分十进制表示的数值（或者是为冒号分开的十进制Ipv6地址），而在通常使用的socket编程中使用的则是二进制值，这就需要将这两个数值进行转换。

ipv4地址：32bit, 4字节，通常采用点分十进制记法。

比如：点分记法192.168.1.24

原数据 11000000 10101000 00000001 00011000

例如对于：10000000 00001011 00000011 00011111

点分十进制表示为：128.11.3.31

ip地址的分类如图8.8所示：

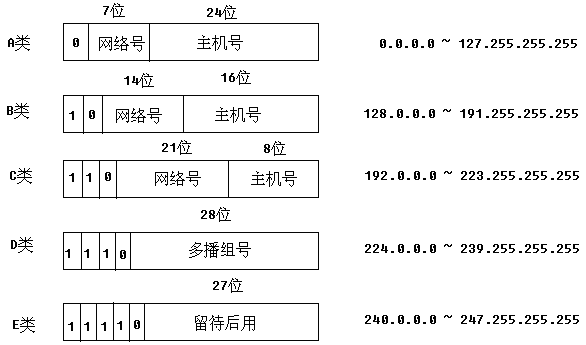


图8.8

#### 私有地址

在IP地址3种主要类型里，各保留了3个区域作为私有地址（用于局域网IP），其地址范围如下

A类地址：10.0.0.0～10.255.255.255

B类地址：172.16.0.0～172.31.255.255

C类地址：192.168.0.0～192.168.255.255

#### 回环地址

A类网络地址127是一个保留地址，用于网络软件测试以及本地机进程间通信，叫做回环地址（loopback address）。无论什么程序，一旦使用回送地址发送数据，协议软件立即返回之，不进行任何网络传输。含网络号127的分组不能出现在任何网络上。

#### 广播地址

TCP/IP规定，主机号全为"1"的网络地址用于广播之用，叫做广播地址。所谓广播，指同时向同一子网所有主机发送报文。

#### 网络地址

使用浏览器等软件登录一些网站时，本质上是连接到对应公司的服务器，这是由于使用浏览器时开启了DNS功能，可以将输入的网址（域名）转换为相应的IP地址。

在WINDOWS可以使用以下步骤查看一些网站的IP：

1. 在 “运行” 中打开 cmd 或 搜索 “命令提示符” ，最终打开一个终端
2. 输入命令 ping 网址
3. 比如：ping [www.baidu.com](http://www.baidu.com)

查看到相应IP ：182.61.200.7

1. 即目前www.baidu.com对应的IP是182.61.200.7，可以通过在浏览器中输入该IP访问对应网站

网络地址：把IP地址转成二进制和子网掩码进行位与运算得到地址，不同IP运算出来结果相同则表示它们在同一个网络中。

#### 子网掩码

子网掩码的作用就是和IP地址与运算后得出网络地址，子网掩码也是32bit，并且是一串1后跟随一串0组成，其中1表示在IP地址中的网络号对应的位数，而0表示在IP地址中主机对应的位数。

1）标准子网掩码

A类网络（1 - 126） 缺省子网掩码：255.0.0.0

255.0.0.0 换算成二进制为 11111111.00000000.00000000.00000000

可以清楚地看出前8位是网络地址，后24位是主机地址，也就是说，如果用的是标准子网掩码，看第一段地址即可看出是不是同一网络的。如 21.0.0.0.1和21.240.230.1，第一段为21属于A类，如果用的是默认的子网掩码，那这两个地址就是一个网段的。

B类网络（128 - 191） 缺省子网掩码：255.255.0.0

C类网络（192 - 223） 缺省子网掩码：255.255.255.0

B类、C类分析同上。

### IPv4和IPv6

IP（网际协议）是TCP/IP协议族中网络层中的协议，IPv4和IPv6是其版本名。

IPv4提供了最多43亿左右（32位地址）的IP地址（对于平民，实际提供了约5亿IP地址），显然其IP地址数量在未来可能是不够用的。所以IPv6被提出且开始实行，IPv6使用128位的IP地址，（保守估计最终效果是一平方米的空间最多支持1000+个网络设备）IPv6提出是为了一劳永逸解决未来IP地址可能短缺的问题，并对IPv4存在的一些进行修复。

### 端口号的概念

TCP/IP协议中的端口，端口号的范围从0~65535，一类是由**互联网名称与数字地址分配机构（**ICANN）负责分配给一些常用的应用程序固定使用的“周知的端口”，其值一般为0~1023.例如http的端口号是80，ftp为21，ssh为22，telnet为23等。还有一类是用户自己定义的，通常是大于1024的整型值。

端口号是用于识别一个网络设备上的网络进程的。

网络通信的本质是不同网络设备上的进程间的通信，要想实现，则至少需要一方获取到另一方的IP和端口号。

### socket概念

Linux中的网络编程是通过socket接口来进行的。socket是一种特殊的I/O接口，它也是一种文件描述符。它是一种常用的进程之间通信机制，通过它不仅能实现本地机器上的进程之间的通信，而且通过网络能够在不同机器上的进程之间进行通信。

每一个socket都用一个半相关描述{协议、本地地址、本地端口}来表示；一个完整的套接字则用一个相关描述{协议、本地地址、本地端口、远程地址、远程端口}来表示。socket也有一个类似于打开文件的系统调用，该系统调用返回一个整型的socket描述符，随后的连接建立、数据传输等操作都是通过这个socket描述符来实现的。

### socket类型

1. 流式socket（SOCK\_STREAM） 🡪用于TCP通信

流式套接字提供可靠的、面向连接的通信流；它使用TCP协议，从而保证了数据传输的正确性和顺序性。

1. 数据报socket（SOCK\_DGRAM） 🡪用于UDP通信

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，并且不保证是可靠、无差错的。它使用数据报协议UDP。

1. 原始socket （SOCK\_RAW） 🡪用于新的网络协议实现的测试等

原始套接字允许对底层协议如IP或ICMP进行直接访问，它功能强大但使用较为不便，主要用于一些协议的开发。

### Linux网络相关命令

先将虚拟机网络设置为桥接模式



此时虚拟机和宿主机视为在同一网络下的两个不同的网络设备，有各自的IP地址

#### ifconfig

查看网络状态，包括IP

|  |
| --- |
| dingwei@dingwei-virtual-machine:~/桌面$ ifconfig  ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  inet 192.168.1.29 netmask 255.255.0.0 broadcast 192.168.255.255  inet6 fe80::e6f:3406:114d:912f prefixlen 64 scopeid 0x20<link>  ether 00:0c:29:c6:5a:4c txqueuelen 1000 (以太网)  RX packets 815446 bytes 78149541 (78.1 MB)  RX errors 0 dropped 0 overruns 0 frame 0  TX packets 25334 bytes 2120274 (2.1 MB)  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  inet 127.0.0.1 netmask 255.0.0.0  inet6 ::1 prefixlen 128 scopeid 0x10<host>  loop txqueuelen 1000 (本地环回)  RX packets 3731 bytes 451662 (451.6 KB)  RX errors 0 dropped 0 overruns 0 frame 0  TX packets 3731 bytes 451662 (451.6 KB)  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0 |

#### ping

用法： ping IP地址/域名

功能：查看与目标的网络连接状态

|  |
| --- |
| dingwei@dingwei-virtual-machine:~/桌面$ ping 192.168.1.24  PING 192.168.1.24 (192.168.1.24) 56(84) bytes of data.  64 bytes from 192.168.1.24: icmp\_seq=1 ttl=64 time=1.21 ms  64 bytes from 192.168.1.24: icmp\_seq=2 ttl=64 time=0.381 ms  若出现形如以上的内容，则说明与目标之间网络连接正常 |

|  |
| --- |
| dingwei@dingwei-virtual-machine:~/桌面$ ping 192.168.4.99  PING 192.168.4.99 (192.168.4.99) 56(84) bytes of data.  From 192.168.1.29 icmp\_seq=1 Destination Host Unreachable  From 192.168.1.29 icmp\_seq=2 Destination Host Unreachable  若出现形如以上的内容，则说明与目标之间网络连接异常 |

在与目标电脑处于同一网络，但使用ping命令却提示网络连接异常，需要考虑关闭防火墙（宿主机、虚拟机）

#### 防火墙相关命令

sudo ufw status 查看防火墙状态

|  |
| --- |
| dingwei@dingwei-virtual-machine:~/桌面$ sudo ufw status  [sudo] dingwei 的密码：  状态：不活动（这里表示防火墙关闭状态） |

sudo ufw enable 开启防火墙

sudo ufw disable 关闭防火墙

## TCP网络编程

### 使用TCP协议的流程图

TCP通信编程一般采用C/S模型（客户端/服务器）

TCP通信的基本步骤如图8.12所示：

服务端：socket---bind---listen---while(1){---accept---recv---send---close---}---close

客户端：socket----------------------------------connect---send---recv-----------------close

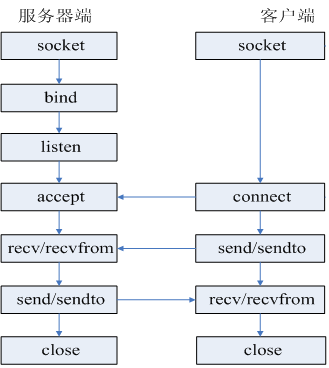


图8.12

### 相关API函数介绍

应用程序需要包含的头文件

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#### socket()函数

|  |
| --- |
| 原型：int socket(int domain,int type,int protocol);  参数：int domain 网络层协议，可以填  AF\_INET 表示IPv4  AF\_INET6 表示IPv6  int type 传输层协议，可以填  SOCK\_STREAM 表示TCP  SOCK\_DGRAM 表示UDP  int protocol 用来指定 socket 所使用的传输协议编号，通常此参考不用管它，设为 0  返回值：成功，套接字文件描述符  失败，-1  功能：基于参数规定的协议，建立一个新的socket通信端口 |

示例：服务器建立一个socket通信端口

|  |
| --- |
| //1. 创建一个套接字  int serfd;  serfd = socket(AF\_INET,SOCK\_STREAM,0);  if(serfd == -1)  {  perror(“socket fail”);  exit(-1);  } |

#### bind()函数

|  |
| --- |
| 原型：int bind(int sockfd,struct sockaddr \* my\_addr,int addrlen);  参数：int sockfd 套接字文件描述符  struct sockaddr \* my\_addr 指向服务器地址信息的指针  int addrlen 服务器地址信息长度，sizeof(struct sockaddr)  返回值：成功，返回0  失败，返回-1  功能：将服务器的地址信息与套接字进行绑定  备注：  原参数中数据类型struct sockaddr是一种通用地址信息结构  struct sockaddr  {  unsigned short int sa\_family;  char sa\_data[14];  };  但是实际使用时一般基于网络层协议，选择与之大小相同但结构的不同专用的地址信息结构  对于IPv4选择以下结构  struct sockaddr\_in  {  unsigned short int sin\_family;  uint16\_t sin\_port;  struct in\_addr sin\_addr;  unsigned char sin\_zero[8];  };  struct in\_addr  {  uint32\_t s\_addr;  };  成员分析：  sin\_family 网络层协议对应的宏，AF\_INET (表示IPv4)  sin\_port 端口号  sin\_addr.s\_addr IP地址  sin\_zero[] 无意义，不需要赋值 |

示例：将服务器的地址信息和套接字绑定

|  |
| --- |
| //2. 填写服务器的地址信息  #define SERVER\_IP “192.168.1.24”  #define PORT\_NUM 11111  typedef struct sockaddr SA;  typedef struct sockaddr\_in SIN;  SIN seraddr;  bzero(&seraddr,sizeof(SIN));  seraddr. sin\_family = AF\_INET ; // AF\_INET-- IPv4  seraddr. sin\_port = htons (PORT\_NUM) ;  seraddr. sin\_addr.s\_addr = inet\_addr(SERVER\_IP) ; //IP地址  //3. 将服务器地址信息与套接字进行绑定  int ret;  ret = bind(serfd,(SA\*)&seraddr,sizeof(SA));  if(ret == -1)  {  perror(“bind fail”);  exit(-1);  } |

补充：

函数inet\_addr()可以实现获取参数中字符串所对应的IP地址，并并返回转换后的32位数据

htons()/htonl()/ntohs()/ntohl() 这些是字节序转换函数，由于主机中采用小端模式，而网络传输时采用大端模式，所以需要使用字节序转换函数。这些函数名中h表示主机，n表示网络，s表示16位数据，l表示32位数据。

#### listen()函数

|  |
| --- |
| 原型：int listen(int s,int backlog);  参数：int s 套接字文件描述符  int backlog 监听队列长度（等待连接的客户端的个数）  返回值：成功，返回0  失败，返回-1  功能：将套接字设置为监听模式，设置监听队列 |

示例：设置服务器的监听队列长度为10

|  |
| --- |
| //4. 设置监听队列  int ret;  ret = listen(serfd , 10);  if(ret == -1)  {  perror(“listen fail”);  exit(-1);  } |

#### accept()函数

|  |
| --- |
| 原型：int accept(int s,struct sockaddr \* addr, socklen\_t \* addrlen);  参数：int s 服务器套接字文件描述符  struct sockaddr \* addr 指向客户端地址信息的指针  int \* addrlen 指向客户端地址信息长度的指针  返回值：成功，返回本次连接的客户端套接字文件描述符  失败，返回-1  功能：等待一个客户端与之连接  注意：每次使用，等待连接一个客户端 |

示例：服务器等待一个客户端的连接

|  |
| --- |
| //5. 等待与客户端连接  int clifd;  SIN cliaddr;  socklen\_t addrlen;  clifd = accept(serfd,(SA\*)& cliaddr,&addrlen);  if(clifd == -1)  {  perror(“accept fail”);  exit(-1);  } |

#### recv()函数

|  |
| --- |
| 原型：int recv(int s,void \*buf,int len,unsigned int flags);  参数：int s 通信对方的套接字文件描述符  void \*buf 指向接收内容的指针  int len 期望接收数据的长度（单位：Byte）  unsigned int flags 一般填0  返回值：成功，返回实际接收数据的长度（单位：Byte）  失败，返回-1  功能：等待接收网络通信中的数据 |

示例：服务器等待接收来自客户端发送的内容

|  |
| --- |
| //6. 数据传输  char r\_buf[40];  int recv\_len;  recv\_len = recv(clifd , r\_buf , sizeof(r\_buf) , 0);  if(recv\_len == -1)  {  perror(“recv fail”);  exit(-1);  } |

#### send()函数

|  |
| --- |
| 原型：int send(int s,const void \* msg,int len,unsigned int flags);  参数：int s 通信对方的套接字文件描述符  const void \* msg 指向待发送内容的指针  int len 待发送内容的长度  unsigned int flags 一般填0  返回值：成功，返回实际发送的数据长度  失败，返回-1  功能：在网络通信中发送内容 |

示例：服务器向客户端发送数据

|  |
| --- |
| //6. 数据传输  char t\_buf[40];  int send\_len;  //准备待发送的数据  send\_len = send(clifd , t\_buf , sizeof(t\_buf) , 0);  if(send\_len == -1)  {  perror(“send fail”);  exit(-1);  } |

#### connect()函数

|  |
| --- |
| 原型：int connect (int sockfd,struct sockaddr \* serv\_addr,int addrlen);  参数：int sockfd 套接字文件描述符  struct sockaddr \* serv\_addr 指向服务器地址信息的指针  int addrlen 服务器地址信息的长度  返回值：成功，返回0  失败，返回-1  功能：客户端主动连接服务器  备注：由于connect()不具有阻塞功能，所以必须要确保服务器在客户端执行connect()前先执行accept()等待连接，否则将连接失败。 |

示例：客户端连接服务器

|  |
| --- |
| //1. 创建一个套接字  int serfd;  serfd = socket(AF\_INET,SOCK\_STREAM,0);  if(serfd == -1)  {  perror(“socket fail”);  exit(-1);  }  //2. 编写服务器地址信息  #define SERVER\_IP “192.168.1.24”  #define PORT\_NUM 11111  typedef struct sockaddr SA;  typedef struct sockaddr\_in SIN;  SIN seraddr;  bzero(&seraddr,sizeof(SIN));  seraddr. sin\_family = AF\_INET ; // AF\_INET-- IPv4  seraddr. sin\_port = htons (PORT\_NUM) ;  seraddr. sin\_addr.s\_addr = inet\_addr(SERVER\_IP) ; //IP地址  //3. 连接服务器  int ret ;  ret = connect(serfd , (SA\*)&seraddr , sizeof(SA));  if(ret == -1)  {  perror(“connect fail”);  exit(-1);  } |

练习：编写TCP的客户端和服务器基本框架，并验证在同一台网络设备上，客户端和服务器的连接。

### TCP示例

说明：服务器终端后马上重启有时候会出现bind: Address already in use 错误，解决方法在bind前面添加以下代码，设置地址可以重复使用。

**int reuse=1;**

**ret = setsockopt(serfd, SOL\_SOCKET, SO\_REUSEADDR, &reuse, sizeof(reuse))**

|  |
| --- |
| 原型：int setsockopt(int s,int level,int optname,const void \* optval,,socklen\_t optlen);  参数：int s 套接字文件描述符  int level 一般填SOL\_SOCKET  int optname欲设置的选项  SO\_REUSEADDR 表示bind()可以重复使用本机的地址  const void \* optval 欲设置的值  socklen\_t optlen optval指向数据的长度  返回值：成功，返回0  失败，返回-1  功能：基于参数设置socket |

#### 基本框架示例

服务器程序：

|  |
| --- |
| dingwei@dingwei-virtual-machine:~/桌面/network/TCP/ver1/SERVER$ cat server.c  #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <unistd.h>  #include <string.h>  #include <stdio.h>  #include <stdlib.h>  #define SERVER\_IP "192.168.1.29"  #define PORT\_NUM 11111  typedef struct sockaddr SA;  typedef struct sockaddr\_in SIN;  int main(void)  {  char t\_buf[40];  char r\_buf[40];  SIN seraddr,cliaddr;  int serfd,clifd;  int ret;  int recv\_len,send\_len;  socklen\_t addrlen;  int reuse=1;  //1. 创建套接字  serfd = socket(AF\_INET,SOCK\_STREAM,0);  if(serfd == -1)  {  perror("socket fail");  exit(-1);  }  //2. 编写服务器地址信息  bzero(&seraddr,sizeof(SIN));  seraddr.sin\_family = AF\_INET ; // AF\_INET-- IPv4  seraddr.sin\_port = htons (PORT\_NUM) ;  seraddr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP) ; //IP地址    //若运行时出现bind: Address already in use 错误，添加下面一行代码  //ret = setsockopt(serfd, SOL\_SOCKET, SO\_REUSEADDR, &reuse, sizeof(reuse));  //3. 将服务器地址信息与套接字进行绑定  ret = bind(serfd,(SA\*)&seraddr,sizeof(SA));  if(ret == -1)  {  perror("bind fail");  exit(-1);  }  //4. 设置监听队列  ret = listen(serfd , 10);  if(ret == -1)  {  perror("listen fail");  exit(-1);  }  //5. 等待与客户端连接  clifd = accept(serfd,(SA\*)& cliaddr,&addrlen);  if(clifd == -1)  {  perror("accept fail");  exit(-1);  }    printf("connect success\r\n");    //6. 数据传输  while(1)  {  recv\_len = recv(clifd,r\_buf,sizeof(r\_buf),0);  printf("recv:%s\r\n",r\_buf);  scanf("%s",t\_buf);  send\_len = send(clifd,t\_buf,sizeof(t\_buf),0);  printf("send:%s\r\n",t\_buf);  }  //7. 结束通信  close(clifd);  close(serfd);  exit(0);  } |

客户端程序：

|  |
| --- |
| dingwei@dingwei-virtual-machine:~/桌面/network/TCP/ver1/CLIENT$ cat client.c  #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <unistd.h>  #include <string.h>  #include <stdio.h>  #include <stdlib.h>  #define SERVER\_IP "192.168.1.29"  #define PORT\_NUM 11111  typedef struct sockaddr SA;  typedef struct sockaddr\_in SIN;  int main(void)  {  char t\_buf[40];  char r\_buf[40];  SIN seraddr,cliaddr;  int serfd,clifd;  int ret;  int recv\_len,send\_len;  socklen\_t addrlen;  //1. 创建套接字  serfd = socket(AF\_INET,SOCK\_STREAM,0);  if(serfd == -1)  {  perror("socket fail");  exit(-1);  }  //2. 编写服务器地址信息  bzero(&seraddr,sizeof(SIN));  seraddr.sin\_family = AF\_INET ; // AF\_INET-- IPv4  seraddr.sin\_port = htons (PORT\_NUM) ;  seraddr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP) ; //IP地址  //3. 连接服务器  ret = connect(serfd , (SA\*)&seraddr , sizeof(SA));  if(ret == -1)  {  perror("connect fail");  exit(-1);  }  printf("connect success\r\n");    //4. 数据传输  while(1)  {  scanf("%s",t\_buf);  send\_len = send(serfd , t\_buf , sizeof(t\_buf) , 0);  printf("send:%s\r\n",t\_buf);  recv\_len = recv(serfd , r\_buf , sizeof(r\_buf) , 0);  printf("recv:%s\r\n",r\_buf);  }  //5. 结束通信  close(serfd);  exit(0);  } |

使用程序时，需要根据服务器端实际的IP地址（可用命令ifconfig查看）进行修改，最终客户端中和服务器端中

#define SERVER\_IP "192.168.1.29"

#define PORT\_NUM 11111

这两个宏的值一致。

#### 多进程服务器

若存在多个客户端，如何编写服务器的程序。

服务器在各个客户端之间切换处理事务，此时服务器端的程序，可以采用多进程或多线程。

多进程：每个新创建进程处理一个新连接的客户端

多线程：每个新创建线程处理一个新连接的客户端

由于服务器中的accept()每成功执行一次就可以连接一个新的客户端，之后就应该为新连接的客户端创建一个子进程或子线程处理该客户端的事务。创建子进程后，父进程执行等待客户端连接的任务，每个子进程执行处理一个客户端事务的任务。

socket();

bind();

listen();

while(1)

{

accept();

ret = fork();

if(ret == 0)

{

//子进程：与一个客户端通信

while(1)

{

recv()/send()

}

}

//父进程：继续执行等待连接客户端

}

close();

考虑以下问题：

1. 客户端断开连接后，服务器如何识别客户端断开了连接？ 检测到recv()返回值为0

服务器怎么处理？ 结束与该客户端通信的子进程或子线程

1. 服务器断开连接、尚未执行到accept()，客户端应该如何处理？ 结束进程 或 尝试多次连接

当服务器等待客户端发送数据（比如使用recv()）时（客户端等待服务器发送数据同理），此时若客户端退出运行，接收来自该客户端的recv()函数将不再具有阻塞功能。那么可以根据recv()函数的返回值（表示实际接收到数据长度）来判断客户端是否断开连接，若断开，则recv()不具有阻塞，那么返回值是0，相应地服务器应该结束这个进程。

服务器代码：

|  |
| --- |
| dingwei@dingwei-virtual-machine:~/桌面/network/TCP/ver2/SERVER$ cat server.c  #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <unistd.h>  #include <string.h>  #include <stdio.h>  #include <stdlib.h>  #define SERVER\_IP "192.168.1.41"  #define PORT\_NUM 11111  typedef struct sockaddr SA;  typedef struct sockaddr\_in SIN;  //多进程服务器  int main(void)  {  char t\_buf[40];  char r\_buf[40];  SIN seraddr,cliaddr;  int serfd,clifd;  int ret;  int recv\_len,send\_len;  socklen\_t addrlen;  int reuse=1;  //1. 创建套接字  serfd = socket(AF\_INET,SOCK\_STREAM,0);  if(serfd == -1)  {  perror("socket fail");  exit(-1);  }  //2. 编写服务器地址信息  bzero(&seraddr,sizeof(SIN));  seraddr.sin\_family = AF\_INET ; // AF\_INET-- IPv4  seraddr.sin\_port = htons (PORT\_NUM) ;  seraddr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP) ; //IP地址  /\*ret = setsockopt(serfd, SOL\_SOCKET, SO\_REUSEADDR, &reuse, sizeof(reuse));  if(ret == -1)  {  perror("setsockopt fail");  exit(-1);  }\*/  //3. 将服务器地址信息与套接字进行绑定  ret = bind(serfd,(SA\*)&seraddr,sizeof(SA));  if(ret == -1)  {  perror("bind fail");  exit(-1);  }  //4. 设置监听队列  ret = listen(serfd , 10);  if(ret == -1)  {  perror("listen fail");  exit(-1);  }    while(1)  {  //5. 父进程：等待与新客户端连接  clifd = accept(serfd,(SA\*)&cliaddr,&addrlen);  if(clifd == -1)  {  perror("accept fail");  exit(-1);  }  printf("connect success\r\n");    ret = fork();  if(ret == 0)  {  //新建子进程：处理客户端事务  //获取客户姓名  /\*  char clientname[10]={0};  bzero(clientname,sizeof(clientname));  recv\_len = recv(clifd,clientname,sizeof(clientname),0);  printf("client %s online\r\n",clientname);  \*/  while(1)  {  //6.收发数据  //clifd--当前客户端套接字  bzero(r\_buf,sizeof(r\_buf));  recv\_len = recv(clifd,r\_buf,sizeof(r\_buf),0);  if(recv\_len == 0)  {  //客户端退出  printf("client%d offline\r\n",clifd);  close(clifd);  exit(0);  }  printf("recv from %d:%s\r\n",clifd,r\_buf);  bzero(t\_buf,sizeof(t\_buf));  scanf("%s",t\_buf);  send\_len = send(clifd,t\_buf,sizeof(t\_buf),0);  printf("send to %d:%s\r\n",clifd,t\_buf);  }  }  }  //7. 结束通信  close(clifd);  close(serfd);  exit(0);  } |

客户端代码：

|  |
| --- |
| dingwei@dingwei-virtual-machine:~/桌面/network/TCP/ver2/CLIENT$ cat client.c  #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <unistd.h>  #include <string.h>  #include <stdio.h>  #include <stdlib.h>  #define SERVER\_IP "192.168.1.41"  #define PORT\_NUM 11111  typedef struct sockaddr SA;  typedef struct sockaddr\_in SIN;  int main(void)  {  char t\_buf[40];  char r\_buf[40];  SIN seraddr,cliaddr;  int serfd,clifd;  int ret;  int recv\_len,send\_len;  socklen\_t addrlen;  //1. 创建套接字  serfd = socket(AF\_INET,SOCK\_STREAM,0);  if(serfd == -1)  {  perror("socket fail");  exit(-1);  }  //2. 编写服务器地址信息  bzero(&seraddr,sizeof(SIN));  seraddr.sin\_family = AF\_INET ; // AF\_INET-- IPv4  seraddr.sin\_port = htons (PORT\_NUM) ;  seraddr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP) ; //IP地址  //3. 连接服务器  ret = connect(serfd , (SA\*)&seraddr , sizeof(SA));  if(ret == -1)  {  perror("connect fail");  exit(-1);  }  //先发送客户姓名  /\*  printf("connect success\r\n");  printf("please input your name:\r\n");  bzero(t\_buf,sizeof(t\_buf));  scanf("%s",t\_buf);  send\_len = send(serfd , t\_buf , sizeof(t\_buf) , 0);  \*/  //4. 数据传输  while(1)  {  fflush(stdin);  bzero(t\_buf,sizeof(t\_buf));  scanf("%s",t\_buf);  send\_len = send(serfd,t\_buf,sizeof(t\_buf),0);  printf("send to server:%s\r\n",t\_buf);  if(strcmp(t\_buf,"QUIT")==0)  {  close(serfd);  exit(0);  }  bzero(r\_buf,sizeof(r\_buf));  recv\_len = recv(serfd , r\_buf , sizeof(r\_buf) , 0);  printf("recv from server:%s\r\n",r\_buf);  }  //5. 结束通信  close(serfd);  exit(0);  } |

#### 多线程服务器

主线程执行等待连接新客户端的任务，并且每连接一个新客户端，就创建一个新的子线程。

每个由主线程创建的子线程任务：处理对应新客户端的事务。

int main()

{

socket();

bind();

listen();

while(1)

{

clifd = accept();

pthread\_create(thread\_task);

}

close();

}

thread\_task()

{

//处理对应客户端的事务

while(1)

{

recv()/send()

}

}

使用多线程服务器时，主线程创建子线程后需要传入哪些数据？

至少传入客户端的套接字clifd。

练习：编写多线程服务器的框架

服务器代码：

|  |
| --- |
| dingwei@dingwei-virtual-machine:~/桌面/network/TCP/ver3/SERVER$ cat server.c  #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <unistd.h>  #include <string.h>  #include <stdio.h>  #include <stdlib.h>  #include <pthread.h>  #define SERVER\_IP "192.168.1.41"  #define PORT\_NUM 11111  #define BUFFER\_SIZE 1024  typedef struct sockaddr SA;  typedef struct sockaddr\_in SIN;  void\* client\_task(void \*arg);  //多线程服务器  int main(void)  {  SIN seraddr,cliaddr;  int serfd,clifd;  int ret;  socklen\_t addrlen;  int reuse=1;  pthread\_t tid;  //1. 创建套接字  serfd = socket(AF\_INET,SOCK\_STREAM,0);  if(serfd == -1)  {  perror("socket fail");  exit(-1);  }  //2. 编写服务器地址信息  bzero(&seraddr,sizeof(SIN));  seraddr.sin\_family = AF\_INET ; // AF\_INET-- IPv4  seraddr.sin\_port = htons (PORT\_NUM) ;  seraddr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP) ; //IP地址  /\*ret = setsockopt(serfd, SOL\_SOCKET, SO\_REUSEADDR, &reuse, sizeof(reuse));  if(ret == -1)  {  perror("setsockopt fail");  exit(-1);  }\*/  //3. 将服务器地址信息与套接字进行绑定  ret = bind(serfd,(SA\*)&seraddr,sizeof(SA));  if(ret == -1)  {  perror("bind fail");  exit(-1);  }  //4. 设置监听队列  ret = listen(serfd , 10);  if(ret == -1)  {  perror("listen fail");  exit(-1);  }    while(1)  {  //5. 主线程：等待与新客户端连接  clifd = accept(serfd,(SA\*)&cliaddr,&addrlen);  if(clifd == -1)  {  perror("accept fail");  exit(-1);  }  printf("connect success\r\n");  //6. 创建子线程，并传入新客户端套接字  pthread\_create(&tid,NULL,client\_task,(void \*)clifd);  }  //7. 结束通信  close(clifd);  close(serfd);  exit(0);  }  //线程任务函数  void\* client\_task(void \*arg)  {  int recv\_len,send\_len;  char r\_buf[BUFFER\_SIZE];  int clifd = (int)arg;  printf("client%d online\r\n",clifd);  while(1)  {  //接收数据  //clifd--当前客户端套接字  bzero(r\_buf,sizeof(r\_buf));  recv\_len = recv(clifd,r\_buf,sizeof(r\_buf),0);  if(recv\_len == 0)  {  //客户端退出  printf("client%d offline\r\n",clifd);  close(clifd);  pthread\_exit(NULL);  }  else  {  printf("recv from %d:%s\r\n",clifd,r\_buf);  }  }  } |

客户端代码：

|  |
| --- |
| dingwei@dingwei-virtual-machine:~/桌面/network/TCP/ver3/CLIENT$ cat client.c  #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <unistd.h>  #include <string.h>  #include <stdio.h>  #include <stdlib.h>  #define SERVER\_IP "192.168.1.41"  #define PORT\_NUM 11111  typedef struct sockaddr SA;  typedef struct sockaddr\_in SIN;  int main(void)  {  char t\_buf[40];  char r\_buf[40];  SIN seraddr,cliaddr;  int serfd,clifd;  int ret;  int recv\_len,send\_len;  socklen\_t addrlen;  //1. 创建套接字  serfd = socket(AF\_INET,SOCK\_STREAM,0);  if(serfd == -1)  {  perror("socket fail");  exit(-1);  }  //2. 编写服务器地址信息  bzero(&seraddr,sizeof(SIN));  seraddr.sin\_family = AF\_INET ; // AF\_INET-- IPv4  seraddr.sin\_port = htons (PORT\_NUM) ;  seraddr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP) ; //IP地址  //3. 连接服务器  ret = connect(serfd , (SA\*)&seraddr , sizeof(SA));  if(ret == -1)  {  perror("connect fail");  exit(-1);  }  //先发送客户姓名  /\*  printf("connect success\r\n");  printf("please input your name:\r\n");  bzero(t\_buf,sizeof(t\_buf));  scanf("%s",t\_buf);  send\_len = send(serfd , t\_buf , sizeof(t\_buf) , 0);  \*/  //4. 数据传输  while(1)  {  fflush(stdin);  bzero(t\_buf,sizeof(t\_buf));  scanf("%s",t\_buf);  send\_len = send(serfd,t\_buf,sizeof(t\_buf),0);  printf("send to server:%s\r\n",t\_buf);  if(strcmp(t\_buf,"QUIT")==0)  {  close(serfd);  exit(0);  }  }  //5. 结束通信  close(serfd);  exit(0);  } |

## UDP网络编程

### UDP框图

UDP通信流程图如图12.13所示：

服务端：socket---bind---recvfrom---sendto---close

客户端：socket----------sendto---recvfrom---close

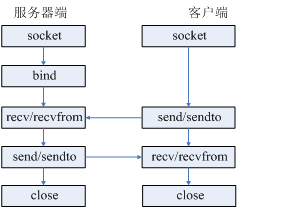


图12.13

UDP框架中没有C/S单独的连接步骤，所以需要客户端先主动发送一次给服务器，在这次数据传输后，服务器才可以获取客户端的信息。之后双方可以自由通信

### UDP相关函数

#### sendto()

|  |
| --- |
| 原型：int sendto ( int s , const void \* msg, int len, unsigned int flags, const struct sockaddr \* to , int tolen ) ;  参数：int s 套接字文件描述符  const void \* msg 指向待发送消息的指针  int len 待发送消息的长度  unsigned int flags 一般填0  const struct sockaddr \* to 指向对方地址信息的指针  int tolen 地址信息长度sizeof(struct sockaddr)  返回值：成功，返回实际发送的数据长度  失败，返回-1  功能：往指定地址的主机发送消息 |

#### recvfrom()

|  |
| --- |
| 原型：int recvfrom(int s,void \*buf,int len,unsigned int flags ,struct sockaddr \*from ,int \*fromlen);  参数：int s 套接字文件描述符  void \*buf 存储接收消息的缓存地址  int len 期望接收的数据长度  unsigned int flags 一般填0  struct sockaddr \*from 用于存储发送方地址信息  int \*fromlen 用于存储发送方地址信息的长度  返回值：成功，返回实际接收数据长度  失败，返回-1  功能：在网络通信中接收一个数据，并获取发送方地址信息 |

在UDP中服务器端使用recvfrom()时无法指定数据的发送方，所以在UDP中若采用多个客户端，需要通过一定算法确定哪个客户端发送数据的。（比如基于recvfrom()的struct sockaddr \*from成员获取客户端的地址进行区分）

### UDP示例

一个服务器和一个客户端通信示例

服务器代码：

|  |
| --- |
| dingwei@dingwei-virtual-machine:~/桌面/network/UDP/ver1/SERVER$ cat server.c  #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <unistd.h>  #include <string.h>  #include <stdio.h>  #include <stdlib.h>  #define SERVER\_IP "192.168.1.41"  #define PORT\_NUM 11111  typedef struct sockaddr SA;  typedef struct sockaddr\_in SIN;  int main(void)  {  char t\_buf[40]={0};  char r\_buf[40]={0};  SIN seraddr,cliaddr;  int serfd,clifd;  int ret;  int recv\_len,send\_len;  socklen\_t addrlen;  int reuse=1;  //1. 创建套接字  serfd = socket(AF\_INET,SOCK\_DGRAM,0);  if(serfd == -1)  {  perror("socket fail");  exit(-1);  }  //2. 编写服务器地址信息  bzero(&seraddr,sizeof(SIN));  seraddr.sin\_family = AF\_INET ; // AF\_INET-- IPv4  seraddr.sin\_port = htons (PORT\_NUM) ;  seraddr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP) ; //IP地址  //3. 将服务器地址信息与套接字进行绑定  ret = bind(serfd,(SA\*)&seraddr,sizeof(SA));  if(ret == -1)  {  perror("bind fail");  exit(-1);  }    do  {  recv\_len = recvfrom(serfd,r\_buf,sizeof(r\_buf),0,\  (SA\*)&cliaddr,&addrlen);  }  while(recv\_len == 0);  printf("connect success\r\n");  //4. 数据传输  while(1)  {  bzero(r\_buf,sizeof(r\_buf));  recv\_len = recvfrom(serfd,r\_buf,sizeof(r\_buf),0,\  (SA\*)&cliaddr,&addrlen);  printf("recv:%s\r\n",r\_buf);  printf("client\_addr is %s\r\n", inet\_ntoa(cliaddr.sin\_addr));  }  //5. 结束通信  close(serfd);  exit(0);  } |

客户端代码

|  |
| --- |
| dingwei@dingwei-virtual-machine:~/桌面/network/TCP/ver3/CLIENT$ cat client.c  #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <unistd.h>  #include <string.h>  #include <stdio.h>  #include <stdlib.h>  #define SERVER\_IP "192.168.1.41"  #define PORT\_NUM 11111  typedef struct sockaddr SA;  typedef struct sockaddr\_in SIN;  int main(void)  {  char t\_buf[40];  char r\_buf[40];  SIN seraddr,cliaddr;  int serfd,clifd;  int ret;  int recv\_len,send\_len;  socklen\_t addrlen;  //1. 创建套接字  serfd = socket(AF\_INET,SOCK\_DGRAM,0);  if(serfd == -1)  {  perror("socket fail");  exit(-1);  }  //2. 编写服务器地址信息  bzero(&seraddr,sizeof(SIN));  seraddr.sin\_family = AF\_INET ; // AF\_INET-- IPv4  seraddr.sin\_port = htons (PORT\_NUM) ;  seraddr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP) ; //IP地址  //3. 连接服务器  ret = connect(serfd , (SA\*)&seraddr , sizeof(SA));  if(ret == -1)  {  perror("connect fail");  exit(-1);  }  //先发送客户姓名  /\*  printf("connect success\r\n");  printf("please input your name:\r\n");  bzero(t\_buf,sizeof(t\_buf));  scanf("%s",t\_buf);  send\_len = send(serfd , t\_buf , sizeof(t\_buf) , 0);  \*/  //4. 数据传输  while(1)  {  fflush(stdin);  bzero(t\_buf,sizeof(t\_buf));  scanf("%s",t\_buf);  send\_len = send(serfd,t\_buf,sizeof(t\_buf),0);  printf("send to server:%s\r\n",t\_buf);  if(strcmp(t\_buf,"QUIT")==0)  {  close(serfd);  exit(0);  }  }  //5. 结束通信  close(serfd);  exit(0);  } |