Day01

### my.h

#ifndef \_MY\_H  
#define \_MY\_H  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <unistd.h>  
#include <fcntl.h>  
#include <string.h>  
#include <errno.h>  
#include <dirent.h>  
#include <pthread.h>  
#include <semaphore.h>  
#include <signal.h>  
#include <sys/shm.h>// share memory  
#include <sys/msg.h>// message queue  
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <arpa/inet.h>  
  
#endif

# 一. 网络体系结构

网络的分层结构及每层使用的协议集合

## 1. OSI七层模型

OSI（Open System Interconnection）七层模型:  
 应用层、表示层、会话层、传输层、网络层、数据链路层、物理层  
   
7.应用层 //FTP、E-mail  
6.表示层 //数据格式定义，数据转换加密  
5.会话层 //建立通信进程的逻辑名字与物理名字之间的联系  
4.传输层 //差错处理、恢复，流量控制，提供可靠的数据传输  
3.网络层 //数据分组，路由选择  
2.数据链路层 //数据组成可发送和接收的帧  
1.物理层 //传输物理信号、接口、信号形式、速率

## 2. TCP/IP四层模型

TCP/IP是一个复杂协议族，有多个协议

TCP/IP四层模型(应用层、传输层、网络层、网络接口层)  
   
 4.应用层  
   
 ftp （File Transfer Protocol） //文件传输协议 传输文件的时候  
 http（Hyper Text Transfer Protocol）//超文本传输协议 浏览网页的时候使用  
 DNS （Domain Name Server） //DNS是域名解析协议  
   
   
 //注册域名 www.net.cn  
 DNS域名解析协议可以将 baidu.com 解析为对应的IP地址 110.242.68.4   
   
 C:\Users\lige>ping www.baidu.com  
 正在 Ping www.baidu.com [110.242.68.4] 具有 32 字节的数据:  
 来自 110.242.68.4 的回复: 字节=32 时间=24ms TTL=51  
 来自 110.242.68.4 的回复: 字节=32 时间=24ms TTL=51  
 来自 110.242.68.4 的回复: 字节=32 时间=23ms TTL=51  
 来自 110.242.68.4 的回复: 字节=32 时间=24ms TTL=51  
  
 TTL=51 //在ping命令执行过程中，数据包的传送经过路由器的个数  
 TTL的初始值是 64， 每经过一个路由TTL的值就-1,  
 所以经过路由个数 64 - 51 = 13  
   
 DHCP（Dynamic Host Configuration Protocol） //动态获取IP地址  
 SMTP（Simple Mail Transfer Protocol）//简单邮件传输协议  
   
 3.传输层  
 传输层主要实现应用程序之间端对端的通信  
 TCP  
 UDP  
   
 2.网络层  
 为数据包选择路由，也就传输的路径  
   
 IP  
 ICMP //在使用ping命令的时候，就会用到ICMP协议  
   
 1.网络接口  
   
 将二进制流转化为数据帧进行发送，数据帧是网络传输的基本单元  
 ARP //将IP地址解析为对应MAC地址  
 MAC地址: 8C-8C-AA-06-E9-A7

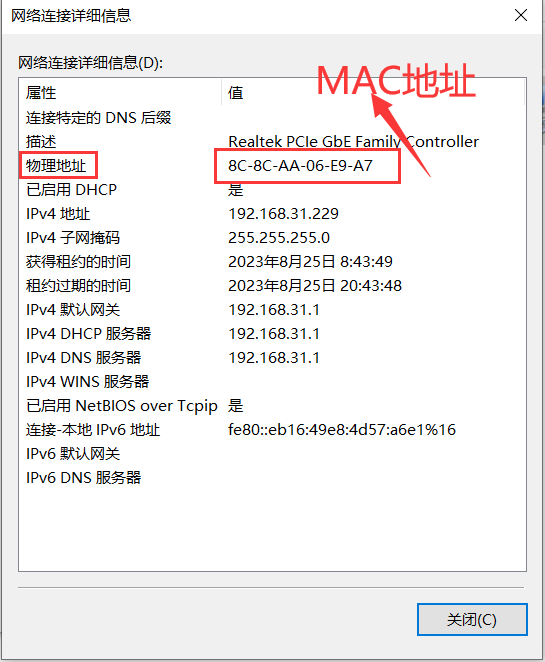


image-20230825111405569

## 3. 二者对比

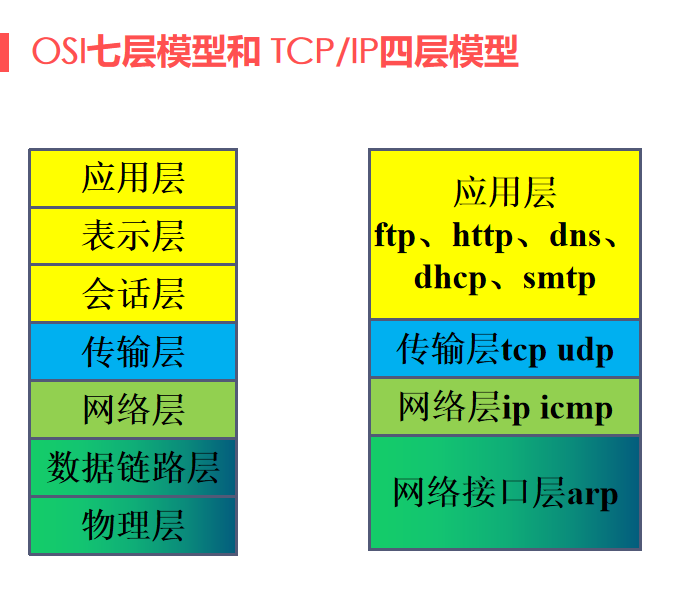


image-20230404134101482

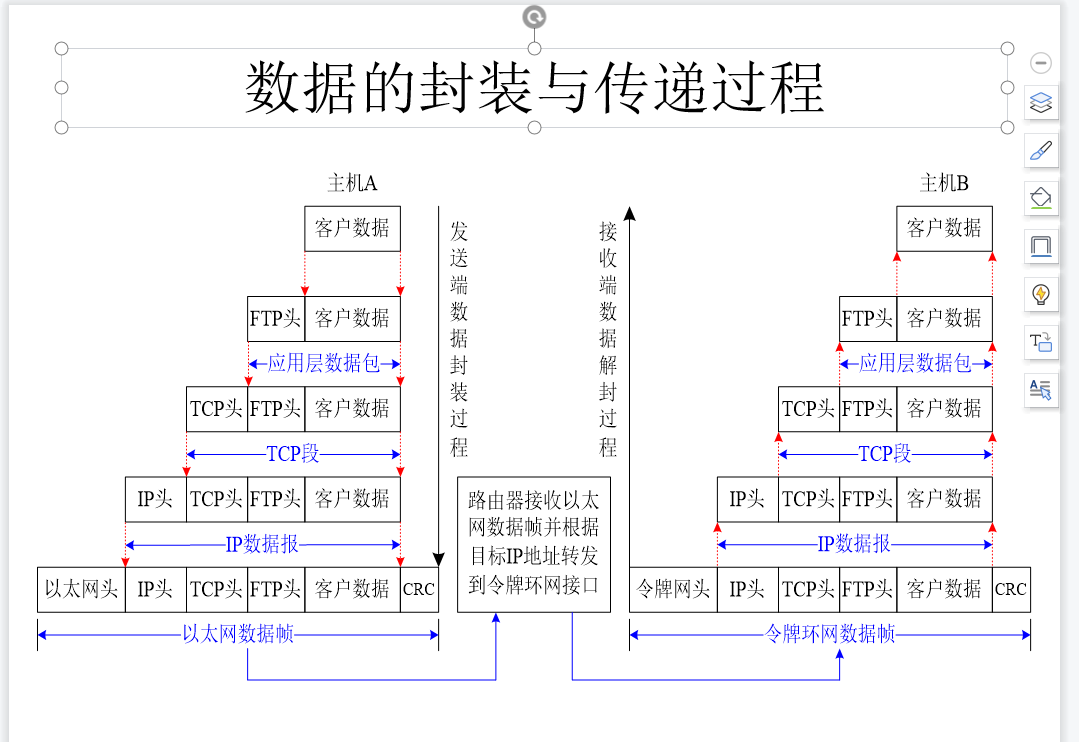


image-20230404134130403

# 二. 网络相关命令

## 0. 设置桥接模式

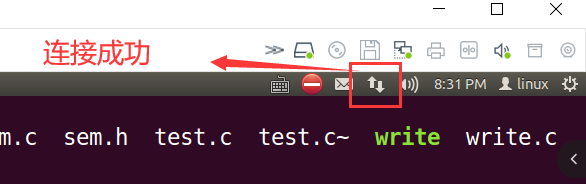
虚拟机设置 ----> 网络适配器



image-20230228102816728

重启网络服务

sudo service network-manager restart



## 1. ifconfig命令

#功能：查看当前Linux系统的IP地址  
sudo ifconfig eth0 192.168.31.25 //临时将eth0这块卡的IP地址设置为192.168.31.25

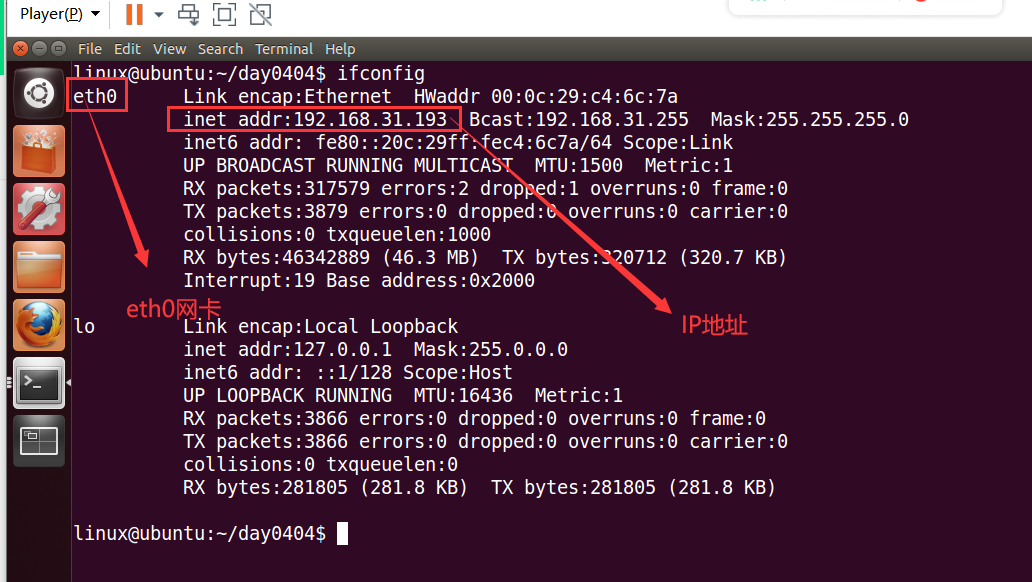


image-20230404135633471

## 2. ping命令

#ping命令功能：可以查看对方主机是否在线  
  
ping 192.168.31.19  
ping baidu.com

## 3. nc命令

netcat 缩写 瑞士军刀   
nc命令可以用来模拟   
   
 TCP 的 客户端和服务器端  
 UDP 的 发送端和接收端  
   
 nc命令模拟UDP的发送端: nc -u 192.168.31.173 4444   
 nc命令模拟UDP的接收端: nc -lu 4444   
   
 nc命令模拟TCP的客户端: nc 192.168.31.173 4444   
 nc命令模拟TCP的服务器端: nc -l 4444

# 三. IP地址

IP地址：主机在网络中的地址，主机要与其他机器通信必须具有的一个IP地址

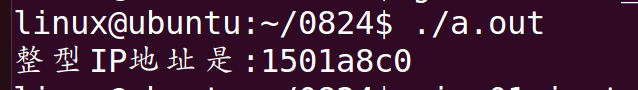
## 1. IP地址表示形式

//windows下查看自己的IP地址 windows+r --->输入cmd ---> ipconfig 命令查询  
(1)点分制 ("192.168.1.21" 字符串) //13字节  
(2)整型 0xC0A80115 //4字节  
 1个十六进制 == 4个二进制位  
 2个十六进制 == 8个二进制位  
 1个字节 == 8bit  
   
 unsigned int

## 2. 点分IP转整型IP

#include <sys/socket.h>  
#include <netinet/in.h>  
  
typedef unsigned int in\_addr\_t;  
in\_addr\_t inet\_addr(const char \*cp);  
  
//调用  
in\_addr\_t addr = inet\_addr("192.168.1.21");  
printf("整型IP地址是%X\n",addr);//%X以十六进制的格式输出  
  
(1)功能:  
 将点分制的ip地址转成整型ip地址  
(2)参数:  
 const char \*cp //保存的是 "192.168.1 21" 字符串的首地址，被转换的点分IP地址  
(3)返回值:  
 返回值就是整型的IP地址

#include "my.h"  
  
int main(int argc, const char \*argv[])  
{  
 //将点分形式的IP地址转换为整型IP地址  
 in\_addr\_t addr = inet\_addr("192.168.1.21");  
 printf("整型IP地址是:%x\n",addr);//%x以十六进制格式输出  
 return 0;  
}



## 3. 整型IP转点分IP

#include <sys/socket.h>  
#include <netinet/in.h>  
#include <arpa/inet.h>  
char \*inet\_ntoa(struct in\_addr in);  
(1)功能:  
 将整型的IP地址转换成点分格式的IP地址)  
(2)参数:  
  
 typedef unsigned int in\_addr\_t;   
 struct in\_addr in //整型的IP地址   
 struct in\_addr {  
 in\_addr\_t s\_addr;  
 };  
   
(3)返回值:  
 char\* 点分形式IP地址字符串的首地址 "192.168.1.21"

#include "my.h"  
  
int main(int argc, const char \*argv[])  
{  
 //将点分形式的IP地址转换为整型IP地址  
 in\_addr\_t addr = inet\_addr("192.168.1.21");  
 printf("整型IP地址是:%x\n",addr);//%x以十六进制格式输出  
 //将整型的IP地址转换为点分形式的IP地址  
 struct in\_addr in;  
 in.s\_addr = addr;  
 printf("点分形式IP地址是%s\n",inet\_ntoa(in));  
 return 0;  
}

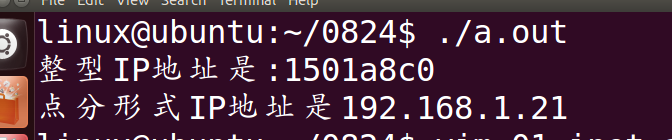


image-20230825141404729

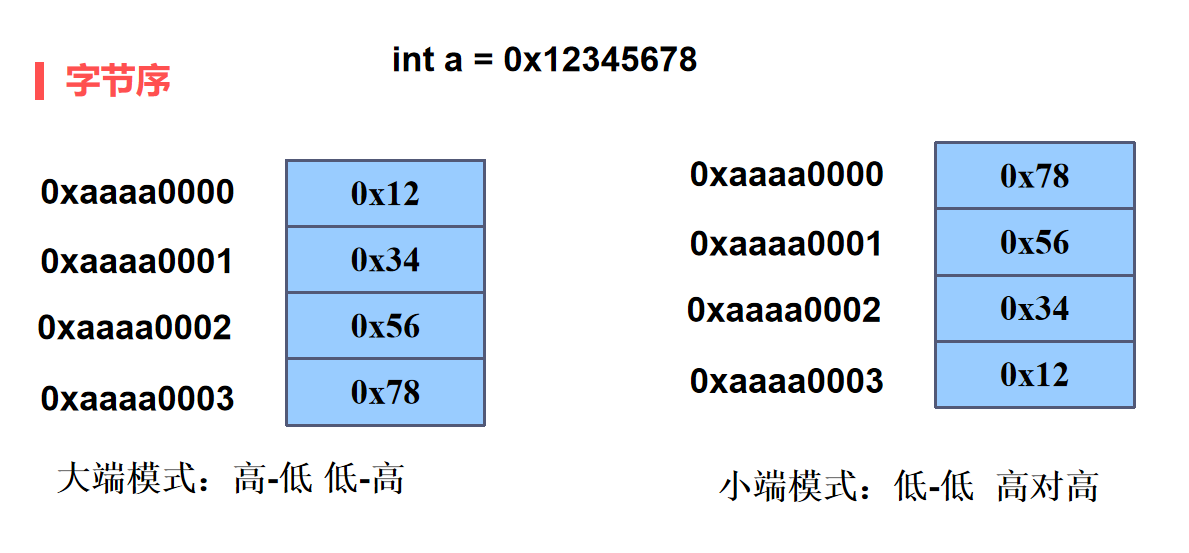
# 四. 端口号

端口号 unsigned short 占2个字节 0 --- 65535  
端口号代表某一类资源 我们写的应用程序尽量用 > 1023的端口号  
   
端口号：是用来识别应用程序的，如果A程序绑定55555端口号，那么网络里面来了一包数据的端口号是55555，那就是发给A程序的数据包  
  
端口号是一种资源，用unsigned short 来保存，(0-65535)有些应用的端口号是固定的  
   
ftp 21 File Transfer Protocol   
http 80 HyperText Transfer Protocol)  
DNS 53 Domain Name System  
DHCP 67 Dynamic Host Configuration Protocol

# 五. 字节序(重点)

## 1. 大端和小端模式

大端模式：高位字节存储在低位地址中，低位字节存储在高位地址中 高对低，低对高  
小端模式：高位字节存储在高位地址中，低位字节存储在低位地址中 高对高，低对低  
   
高位字节和低位字节  
 76 54 32 10  
0x12 34 56 78  
   
0x12 //高位字节  
0x78 //低位字节  
   
高位地址和低位地址  
int a[5];  
&a[0] //低位地址  
&a[4] //高位地址  
   
0xaaaa0000 //低位地址  
0xaaaa0001 //高位地址



#### 笔试题

笔试题：请你编写一个程序，测试当前系统的字节序，是大端模式还是小端模式

#include "my.h"  
  
//请你编写程序,测试当前主机字节序是大端模式还是小端模式  
int main(int argc, const char \*argv[])  
{  
 int a = 0x12345678;  
 char\* p = (char\*)&a;  
 printf("%p %p %p %p\n",p+0, p+1, p+2,p+3);  
 printf("%x %x %x %x\n",\*(p+0), \*(p+1), \*(p+2),\*(p+3));  
 return 0;  
}

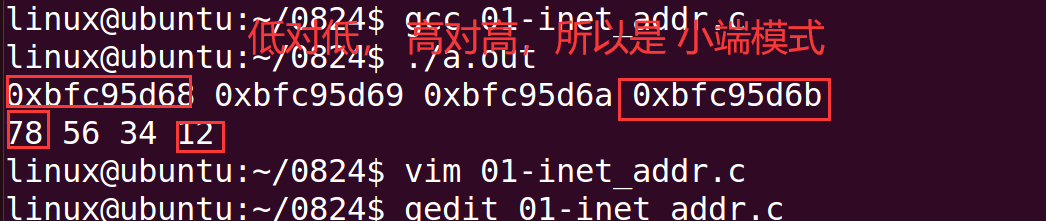


image-20230825145015241

#include "my.h"  
  
//利用共用体  
union data  
{  
 int a;  
 char b;  
};  
  
//请你编写程序,测试当前主机字节序是大端模式还是小端模式  
int main(int argc, const char \*argv[])  
{  
 union data s;  
 s.a = 0x12345678;  
 //利用共用体，取出成员变量a第一个字节存储的数据  
 printf("s.b is %x\n",s.b);//s.b is 78,说明地位地址,存储的是地位字节,所以是小端  
 return 0;  
}

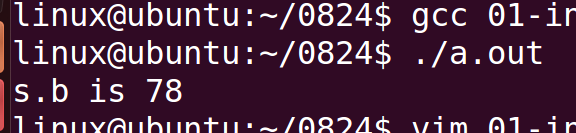


image-20230825145648887

## 2. 主机和网络字节序（重点）

主机字节和网络字节序，本质上就是探索是小端存储还是大端存储  
   
主机字节序：通常采用的是小端模式，高位字节存储在高位地址中，低位字节存储在低位地址中 高对高，低对低  
网络字节序：采用的是大端模式，高位字节存储在低位地址中，低位字节存储在高位地址中 高对低，低对高

### 2.1 主机字节序转网络字节序

本质上就小端模式转换为大端模式  
h host//主机  
n network //网络  
s short   
l long   
   
htons()函数 //将主机字节序的短整型转换为网络字节序  
htonl()函数 //将主机字节序的长整型转换为网络字节序

### 2.2 网络字节序转主机字节序

ntohs()函数 //将网络字节序的短整型转换为主机字节序  
ntohl()函数 //将网络字节序的长整型转换为主机字节序

#include "my.h"  
  
  
//请你编写程序,测试当前主机字节序是大端模式还是小端模式  
int main(int argc, const char \*argv[])  
{  
 in\_addr\_t addr = 0x12345678;//默认当前主机存储addr是小端模式  
 printf("addr is %x\n",addr);  
  
 in\_addr\_t bigAddr = htonl(addr);//将小端转换为大端,返回值是大端模式的存储  
 printf("addr is %x\n",bigAddr);  
  
 return 0;  
}

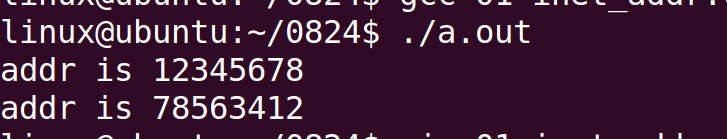


image-20230825153714188

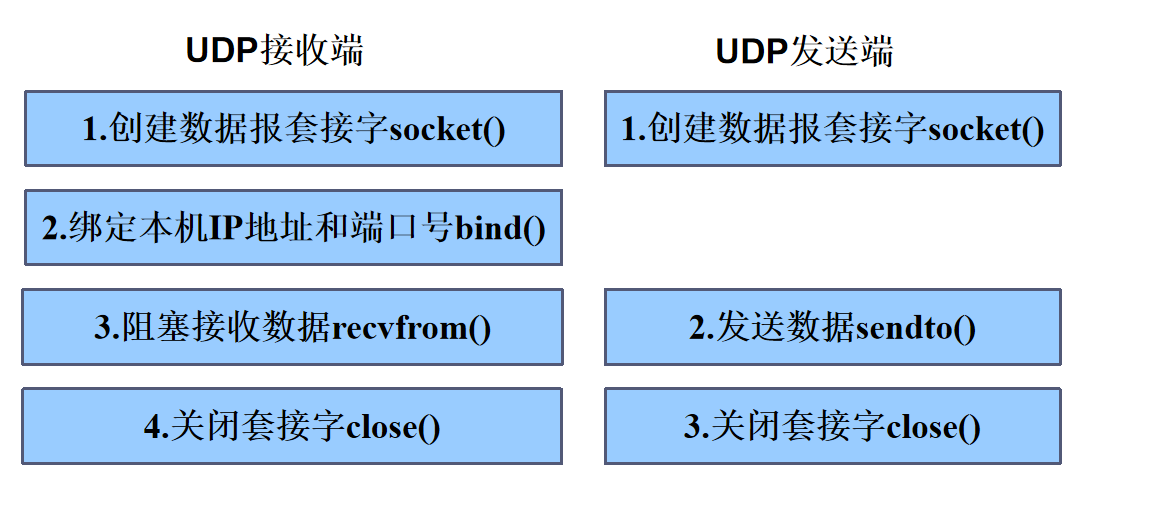
# 六. UDP通信

## 1. UDP概述

(1)什么是UDP？  
 UDP是一个面向无连接的不可靠的传输协议  
(2)UDP通信协议位于哪一层？  
 传输层  
 UDP //User Datagram Protocol 用户数据报协议  
(3)UDP通信的优点和缺点  
 优点： 传输速度快，效率高，实时性好  
 缺点： 不可靠，传输数据的时候容易丢失数据  
(4)应用场合  
 传输大文件、网络视频实时会议等

## 2. UDP两端通信(重点)

接收端:  
  
(1) 创建一个socket数据报套接字socket()  
(2) 绑定自己主机的IP地址和端口号bind()  
(3) 接收数据recvfrom()   
(4) 关闭套接字close()   
   
发送端:  
  
(1)创建一个数据报套接字socket()  
(2)发送数据sendto()  
(3)关闭套接字close()



### 2.1 接收端

(1) 创建一个socket数据报套接字socket()  
(2) 绑定自己主机的IP地址和端口号bind()  
(3) 接收数据recvfrom()  
(4) 关闭套接字close()

#### (1)创建一个数据报套接字socket()

#include <sys/types.h>   
#include <sys/socket.h>  
int socket(int domain, int type, int protocol);  
  
//调用:  
int sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);  
  
(1)功能：  
 创建一个socket套接字  
(2)参数说明：  
 int domain //网络的通信协议 AF\_INET IPv4  
 int type //套接字的类型   
 SOCK\_DGRAM 数据报套接字 UDP通信使用  
 int protocol //0 代表自动默认匹配相关协议  
  
(3)返回值:  
 成功：非负套接字描述符 ,socket套接字是一个特殊的文件描述符  
 失败: -1

#### (2) 绑定自己主机的IP地址和端口号bind()

int bind(int sockfd, struct sockaddr \*addr, int addrlen);  
网络地址结构 struct sockaddr （协议、本机地址、本机端口）  
   
(1)功能：绑定自己本机的IP地址和端口号  
  
 struct sockaddr\_in myaddr = { 0 };//用来保存自己的IP地址和端口号   
 myaddr.sin\_family = AF\_INET;//family协议选择IPv4  
 myaddr.sin\_port = htons(4444);//端口号   
 myaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");  
 //myaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);//使用INADDR\_ANY 可以自动获取本机网卡IP地址  
 //myaddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); //"127.0.0.1"IP地址是主机环回地址  
 bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myaddr));  
 &myaddr//类型 struct sockaddr\_in \*,函数参数需要的是struct sockaddr\*,所以强制类型转换一下  
  
   
(2)参数说明:  
 int sockfd //上一步，socket函数的返回值   
 struct sockaddr \*addr //用来保存自己的IP地址和端口号的结构体的首地址  
 int addrlen //网络地址结构长度  
   
(3)返回值:  
 成功: 0   
 失败: -1  
  
   
struct sockaddr  
{  
 short sa\_family; //AF\_INET (tcp/ip通信)  
 char sa\_data[14]; //字符数组有14个字节   
 1-2字节 端口号  
 3-6 ip地址 7-14 预留   
};  
   
因为 struct sockaddr 存储数据不方便，所以，linux又定义了下面这个结构体struct sockaddr\_in 方便赋值  
struct sockaddr\_in  
{  
 short sin\_family; //AF\_INET //2  
 short sin\_port; //端口号 //2  
 struct in\_addr sin\_addr; // 4 struct in\_addr 4 s\_addr //IP地址  
 unsigned char sin\_zero[8]; // 预留  
 };  
struct in\_addr {  
 in\_addr\_t s\_addr;  
 };

#### (3) 接收数据recvfrom()

ssize\_t recvfrom(int sockfd, void \*buf, size\_t len, int flags,  
 struct sockaddr\*src\_addr, socklen\_t \*addrlen);  
  
(1)功能: 阻塞 接收数据  
   
 //调用   
 //不想保存发送方的IP地址和端口号 (不想知道发送消息的那个人的IP地址和端口号,不想知道他是谁)  
 char buf[100] = { 0 };//用来保存接收到的数据  
 recvfrom(sockfd, buf, sizeof(buf), 0, NULL, NULL);  
   
  
 //想要保存发送方的IP地址和端口号,也就是想要知道到底是谁给我发送的数据  
 struct sockaddr\_in youaddr = { 0 };//用来保存对方的IP地址和端口号，即发送方  
 int len = sizeof(youaddr);  
 recvfrom(sockfd, buf, sizeof(buf), 0, (struct sockaddr\*)&youaddr, &len);  
  
(2)参数说明:  
   
 int sockfd //第一步socket函数的返回值  
 void \*buf //接收到的数据存放的位置  
 size\_t len //接收数据的最大字节数  
 int flags //0 阻塞接收  
 struct sockaddr \*src\_addr //用来保存对方的IP地址和端口号   
 socklen\_t \*addrlen //对方iP地址和端口号的长度  
   
(3)返回值:  
  
 成功: 实际接收字节数   
 失败: -1

#### (4)关闭套接字close()

close(sockfd);

#### my.h

#ifndef \_MY\_H  
#define \_MY\_H  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <unistd.h>  
#include <fcntl.h>  
#include <string.h>  
#include <errno.h>  
#include <dirent.h>  
#include <pthread.h>  
#include <semaphore.h>  
#include <signal.h>  
#include <sys/shm.h>// share memory  
#include <sys/msg.h>// message queue  
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <arpa/inet.h>  
  
#endif

#### recv.c接收端

#include "my.h"  
  
int main(int argc, const char \*argv[])  
{  
 char buf[100] = { 0 };//用来保存接收到的数据  
 //1.创建一个数据报套接字 SOCK\_DGRAM数据报套接字  
 //第3个参数0,自动匹配默认的相关协议  
 int sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed");  
 exit(-1);  
 }  
 printf("sockfd is %d\n",sockfd);//sockfd is 3  
 //2.绑定本机的IP地址和端口号  
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(4444);  
 myaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");//"127.0.0.1"  
 int ret = bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myaddr));  
 if(ret == -1)  
 {  
 perror("bind failed");  
 exit(-1);  
 }  
 printf("bind sucessful!!\n");  
  
 struct sockaddr\_in youaddr = { 0 };//用来保存发送方的IP地址和端口号  
 int len = sizeof(youaddr);  
 while(1)  
 {  
 //3.阻塞接收数据  
 //方法一:第5 6个参数,赋值为NULL,代表不想知道是谁给我发送的消息  
 //recvfrom(sockfd, buf, sizeof(buf), 0, NULL, NULL);  
 //方法二:通过第5 6个参数,知道到底是谁给我发送的消息,也就是要获取发送方的IP地址和端口号  
 recvfrom(sockfd, buf, sizeof(buf), 0, (struct sockaddr\*)&youaddr, &len);  
 printf("接收到的数据是来自于%s-%d: %s\n", inet\_ntoa(youaddr.sin\_addr),ntohs(youaddr.sin\_port),buf);  
 //inet\_ntoa将整型IP地址转换为点分形式IP地址  
 //ntohs 将网络字节序的端口号转换为主机字节序的端口号进行打印  
 }  
 //4.关闭套接字  
 close(sockfd);  
  
 return 0;  
}

#### send.c发送端

nc -u 192.168.31.179 4444 //参数192.168.31.179 4444 是接收端绑定的IP地址和端口号，代表的是给  
 192.168.31.179 4444 这个人发送数据

### 2.2 发送端

(1)创建一个数据报套接字socket()  
(2)发送数据sendto()  
(3)关闭套接字close()

#### (1)创建一个数据报套接字socket()

和接收端第一步相同

#### (2)发送数据sendto()

ssize\_t sendto(int sockfd, const void \*buf, size\_t len, int flags,  
 const struct sockaddr \*dest\_addr, socklen\_t addrlen);个  
(1)功能: 发送数据  
   
 //调用  
 char buf[100] = "hello";  
 //发送方，在发送数据之前，一定是提前已经知道了接收方的IP地址和端口号  
 struct sockaddr\_in toaddr = { 0 }; //用来提前保存上接收方的IP地址和端口号   
 //发送方是当前的他,toaddr里面存的是对方的手机号即 她的手机号  
 toaddr.sin\_family = AF\_INET;  
 toaddr.sin\_port = htons(4444);  
 //toaddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");  
 toaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");  
  
 //在调用sendto函数的时候，数据发送给谁，取决于 toaddr里面装的是谁的IP地址和端口号  
 sendto(sockfd, buf, strlen(buf)+1, 0, (struct sockaddr\*)&toaddr, sizeof(toaddr));  
   
(2)函数说明:  
  
 int sockfd //socket函数的返回值   
 const void \*buf //发送数据存放的位置   
 size\_t len //实际发送的字节数   
 int flags // 通常为0   
 const struct sockaddr \*dest\_addr //接收方的IP地址和端口号   
 socklen\_t addrlen //网络地址结构长度  
  
(3)返回值:  
  
 成功: 实际发送的字节数   
 失败: -1

#### (3)关闭套接字close()

close(sockfd);

#### recv.c

#include "my.h"  
  
int main(int argc, const char \*argv[])  
{  
 char buf[100] = { 0 };//用来保存接收到的数据  
 //1.创建一个数据报套接字 SOCK\_DGRAM数据报套接字  
 //第3个参数0,自动匹配默认的相关协议  
 int sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed");  
 exit(-1);  
 }  
 printf("sockfd is %d\n",sockfd);//sockfd is 3  
 //2.绑定本机的IP地址和端口号  
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(4444);  
 myaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");//"127.0.0.1"  
 int ret = bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myaddr));  
 if(ret == -1)  
 {  
 perror("bind failed");  
 exit(-1);  
 }  
 printf("bind sucessful!!\n");  
  
 struct sockaddr\_in youaddr = { 0 };//用来保存发送方的IP地址和端口号  
 int len = sizeof(youaddr);  
 while(1)  
 {  
 //3.阻塞接收数据  
 //方法一:第5 6个参数,赋值为NULL,代表不想知道是谁给我发送的消息  
 //recvfrom(sockfd, buf, sizeof(buf), 0, NULL, NULL);  
 //方法二:通过第5 6个参数,知道到底是谁给我发送的消息,也就是要获取发送方的IP地址和端口号  
 recvfrom(sockfd, buf, sizeof(buf), 0, (struct sockaddr\*)&youaddr, &len);  
 printf("接收到的数据是来自于%s-%d: %s\n", inet\_ntoa(youaddr.sin\_addr),ntohs(youaddr.sin\_port),buf);  
 //inet\_ntoa将整型IP地址转换为点分形式IP地址  
 //ntohs 将网络字节序的端口号转换为主机字节序的端口号进行打印  
 }  
 //4.关闭套接字  
 close(sockfd);  
  
 return 0;  
}

#### send.c

#include "my.h"  
  
int main(int argc, const char \*argv[])  
{  
 char buf[100] = "hello";  
 //1.创建一个数据报套接字  
 int sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed");  
 exit(-1);  
 }  
 //2.给接收方发送数据  
 //(1)在发送之前已经提前知道了接收方的IP地址和端口号  
 struct sockaddr\_in toaddr = { 0 };//toaddr保存的是谁的IP地址和端口号就是给谁发送数据  
 toaddr.sin\_family = AF\_INET;  
 toaddr.sin\_port = htons(4444);  
 toaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");  
   
 sendto(sockfd, buf, strlen(buf)+1, 0, (struct sockaddr\*)&toaddr, sizeof(toaddr));  
  
 //3.关闭套接字  
 close(sockfd);  
 return 0;  
}

### #练习1：

gcc -o send send.c  
gcc -o recv recv.c   
   
//作业：发送端和接收端IP地址和端口号都改成命令行传递参数  
./recv 192.168.31.173 4444 //做为接收方，绑定的IP地址是192.168.31.173 端口号4444  
./send 192.168.31.173 4444 //做为发送方，给IP地址是192.168.31.173 端口号4444的接收方发送数据

#### my.h

#ifndef \_MY\_H  
#define \_MY\_H  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <unistd.h>  
#include <fcntl.h>  
#include <string.h>  
#include <errno.h>  
#include <dirent.h>  
#include <pthread.h>  
#include <semaphore.h>  
#include <signal.h>  
#include <sys/shm.h>// share memory  
#include <sys/msg.h>// message queue  
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <arpa/inet.h>  
  
#endif

#### send.c

#include "my.h"  
  
int main(int argc, const char \*argv[])  
{  
 if(argc != 3)  
 {  
 printf("忘记传递参数!! ./send 192.168.31.179 4444\n");  
 exit(-1);  
 }  
 char buf[100] = "hello";  
 //1.创建一个数据报套接字  
 int sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed");  
 exit(-1);  
 }  
 //2.给接收方发送数据  
 //(1)在发送之前已经提前知道了接收方的IP地址和端口号  
 struct sockaddr\_in toaddr = { 0 };//toaddr保存的是谁的IP地址和端口号就是给谁发送数据  
 toaddr.sin\_family = AF\_INET;  
 toaddr.sin\_port = htons(atoi(argv[2]));  
 toaddr.sin\_addr.s\_addr = inet\_addr(argv[1]);  
   
 while(1)  
 {  
 printf("请输入要发送的话:\n");  
 scanf("%s",buf);  
 sendto(sockfd, buf, strlen(buf)+1, 0, (struct sockaddr\*)&toaddr, sizeof(toaddr));  
 }  
  
 //3.关闭套接字  
 close(sockfd);  
 return 0;  
}

#### recv.c

#include "my.h"  
  
int main(int argc, const char \*argv[])  
{  
 if(argc != 3)  
 {  
 printf("忘记传递参数!! ./recv 192.168.31.179 4444\n");  
 exit(-1);  
 }  
 char buf[100] = { 0 };//用来保存接收到的数据  
 //1.创建一个数据报套接字 SOCK\_DGRAM数据报套接字  
 //第3个参数0,自动匹配默认的相关协议  
 int sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed");  
 exit(-1);  
 }  
 printf("sockfd is %d\n",sockfd);//sockfd is 3  
 //2.绑定本机的IP地址和端口号  
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(atoi(argv[2]));  
 myaddr.sin\_addr.s\_addr = inet\_addr(argv[1]);//"127.0.0.1"  
 int ret = bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myaddr));  
 if(ret == -1)  
 {  
 perror("bind failed");  
 exit(-1);  
 }  
 printf("bind sucessful!!\n");  
  
 struct sockaddr\_in youaddr = { 0 };//用来保存发送方的IP地址和端口号  
 int len = sizeof(youaddr);  
 while(1)  
 {  
 //3.阻塞接收数据  
 //方法一:第5 6个参数,赋值为NULL,代表不想知道是谁给我发送的消息  
 //recvfrom(sockfd, buf, sizeof(buf), 0, NULL, NULL);  
 //方法二:通过第5 6个参数,知道到底是谁给我发送的消息,也就是要获取发送方的IP地址和端口号  
 recvfrom(sockfd, buf, sizeof(buf), 0, (struct sockaddr\*)&youaddr, &len);  
 printf("接收到的数据是来自于%s-%d: %s\n", inet\_ntoa(youaddr.sin\_addr),ntohs(youaddr.sin\_port),buf);  
 //inet\_ntoa将整型IP地址转换为点分形式IP地址  
 //ntohs 将网络字节序的端口号转换为主机字节序的端口号进行打印  
 }  
 //4.关闭套接字  
 close(sockfd);  
  
 return 0;  
}

#### INADDR\_ANY获取IP，命令行传端口号

myaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);//自动获取得到IP地址

# @作业

类似于消息队列的聊天程序  
   
进程A: ./chat 1 2   
进程B: ./chat 2 1   
gcc -o chat chat.c -lpthread  
 0 1 2 3  
A同学电脑(李任杰:192.168.31.7): ./chat 33333 192.168.31.22 44444   
 //A同学绑定的端口号是33333，本机IP地址通过INADDR\_ANY获取192.168.31.7 做为接收方绑定的IP地址和端口号  
 //192.168.31.22 44444 B同学绑定的IP地址和端口,给这个IP地址和端口号发数据  
   
B同学电脑(庄宇杰:192.168.31.22): ./chat 44444 192.168.31.7 33333   
 //B同学绑定的端口号是44444，本机IP地址通过INADDR\_ANY获取192.168.31.22  
 //192.168.31.7 33333 A同学绑定的IP地址和端口号，给这个IP地址和端口号发数据  
  
自己和自己测试  
 ./chat 33333 127.0.0.1 44444   
 ./chat 44444 127.0.0.1 33333  
   
void\* send(void \*p)  
{  
 //子线程循环输入发送  
 while(1)  
 {  
 scanf();  
 sendto();  
 }  
}  
int main()  
{  
 pthread\_create();  
 //主进程接收消息  
 while(1)  
 {  
 recvfrom();//接收  
 }  
}

### 方案一: 利用全局变量

#include "my.h"  
  
//全局变量解决  
int sockfd;  
struct sockaddr\_in toaddr;  
  
//发送数据线程  
void\* sendThread(void\* p)  
{  
 char buf[100] = { 0 };  
 while(1)  
 {  
 scanf("%s", buf);  
 sendto(sockfd, buf, strlen(buf)+1, 0, (struct sockaddr\*)&toaddr, sizeof(toaddr));  
 }  
}  
  
int main(int argc, const char \*argv[])  
{  
 if(argc != 4)  
 {  
 perror("忘记传递参数 ./chat 4444 192.168.31.56 3333\n");  
 exit(-1);  
 }  
 pthread\_t id;  
 char buf[100] = { 0 };//保存接收到的数据  
 //1.创建一个数据报套接字  
 sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed");  
 exit(-1);  
 }  
  
 toaddr.sin\_family = AF\_INET;  
 toaddr.sin\_port = htons(atoi(argv[3]));  
 toaddr.sin\_addr.s\_addr = inet\_addr(argv[2]);  
  
 //2.绑定自己本机的IP地址和端口号  
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(atoi(argv[1]));  
 myaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  
 int ret = bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myaddr));  
 if(ret == -1)  
 {  
 perror("bind failed");  
 exit(-1);  
 }  
 //创建一个线程,专门负责发送  
 pthread\_create(&id, NULL, sendThread, NULL);  
  
 //主进程负责循环接收数据  
 struct sockaddr\_in youaddr = { 0 };  
 int len = sizeof(youaddr);  
 while(1)  
 {  
 recvfrom(sockfd, buf, sizeof(buf), 0, (struct sockaddr\*)&youaddr, &len);  
 printf("from%s-%d: %s\n",inet\_ntoa(youaddr.sin\_addr), ntohs(youaddr.sin\_port),buf);  
 }  
 return 0;  
}

### 方法二: 利用线程传递参数

#include "my.h"  
  
//利用线程传递参数  
typedef struct   
{  
 int sockfd;  
 struct sockaddr\_in toaddr;  
}data\_t;  
  
//发送数据线程  
void\* sendThread(void\* p)//void\* p = &data  
{  
 data\_t\* q = p;  
 char buf[100] = { 0 };  
 while(1)  
 {  
 scanf("%s", buf);  
 sendto(q->sockfd, buf, strlen(buf)+1, 0, (struct sockaddr\*)&q->toaddr, sizeof(q->toaddr));  
 }  
}  
  
int main(int argc, const char \*argv[])  
{  
 if(argc != 4)  
 {  
 perror("忘记传递参数 ./chat 4444 192.168.31.56 3333\n");  
 exit(-1);  
 }  
 pthread\_t id;  
 char buf[100] = { 0 };//保存接收到的数据  
 //1.创建一个数据报套接字  
 int sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed");  
 exit(-1);  
 }  
 data\_t data = { 0 };//用来保存给线程传递sockfd 和 IP地址 端口号  
  
 data.sockfd = sockfd;  
 data.toaddr.sin\_family = AF\_INET;  
 data.toaddr.sin\_port = htons(atoi(argv[3]));  
 data.toaddr.sin\_addr.s\_addr = inet\_addr(argv[2]);  
  
 //2.绑定自己本机的IP地址和端口号  
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(atoi(argv[1]));  
 myaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  
 int ret = bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myaddr));  
 if(ret == -1)  
 {  
 perror("bind failed");  
 exit(-1);  
 }  
 //创建一个线程,专门负责发送  
 pthread\_create(&id, NULL, sendThread, &data);  
  
 //主进程负责循环接收数据  
 struct sockaddr\_in youaddr = { 0 };  
 int len = sizeof(youaddr);  
 while(1)  
 {  
 recvfrom(sockfd, buf, sizeof(buf), 0, (struct sockaddr\*)&youaddr, &len);  
 printf("from%s-%d: %s\n",inet\_ntoa(youaddr.sin\_addr), ntohs(youaddr.sin\_port),buf);  
 }  
 return 0;  
}

# #复习

1. OSI七层模型 和 TCP/IP四层模型  
   
 应用层   
 表示层 应用层  
 会话层  
 传输层 传输层  
 网络层 网络层  
 数据链路层  
 物理层 网络接口层  
  
2. TCP/IP是一个复杂的协议族，有多个协议  
   
 应用层  
   
 ftp //文件传输协议  
 http //超文本传输协议 浏览网页  
 dns //域名解析协议  
 dhcp //动态获取IP地址  
 smtp //简单邮件传输协议  
   
 传输层  
 功能:实现 端对端的通信  
   
 端口号： unsigned short 占2个字节，端口号用来区别应用程序的   
 取值范围 0-65535 , 0-1023端口号不能用，系统占用  
 tcp 和 udp  
   
 网络层  
 功能：为数据包选择传输路径 (选择路由)  
 ip  
 icmp //ping命令使用的就是icmp协议  
   
 网络接口层  
 功能：将二进制流转换为数据帧发送，数据帧是网络传输数据的基本单元  
 arp //将IP地址解析对应的MAC地址  
  
3. 网络编程称为socket编程 socket套接字  
 套接字是一个特殊的文件描述符  
   
4. IP地址的表现形式   
  
 点分形式: "192.168.1.21"   
 整型形式: 0xC0A80115  
 两种IP地址形式的相互转换  
   
 点分 ----> 整型  
 in\_addr\_t addr = inet\_addr("192.168.1.21");  
  
 整型 ----> 点分  
 struct in\_addr in;   
 in.s\_addr = addr;   
 char\* p = inet\_ntoa(in);  
 printf("点分: %s\n",p);  
  
5. 字节序(重点)  
   
 主机字节序:通常是小端模式，高位字节存储在高位地址中，低位字节存储在低位地址中  
 网络字节序:采用的是大端模式，高位字节存储在低位地址中，低位字节存储在高位地址中  
   
 字节序转换   
 主机 -----> 网络 (小端-->大端)  
 htons()  
 htonl()  
 网络 -----> 主机 (大端-->小端)  
 ntohs()  
 ntohl()  
  
6. ping命令：查看对方主机是否在线 ping 192.168.1.21  
   
 Linux查看IP地址命令： ifconfig  
 sudo ifconfig eth0 192.168.31.19 //临时将当前eth0这个网卡的IP地址设置为192.168.31.19  
   
 Window查看IP地址命令: ipconfig   
   
7. UDP通信  
   
 UDP位于传输层，是一个面向无连接的不可靠的传输协议，使用的套接字类型 数据报套接字SOCK\_DGRAM  
 UDP优点：传输速度快，效率高，实时性好  
 UDP缺点：不可靠，容易丢失数据  
   
 接收端  
 //1.创建一个数据报套接字socket函数  
 int sockfd = socket(AF\_INET, SOCK\_DGREAM, 0);  
 //2.绑定本机的IP地址和端口号  
 struct sockaddr //这个结构体类型不好赋值  
 struct sockaddr\_in //这个结构体方便赋值  
   
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(55555);  
 //myaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");  
 myaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  
 int ret = bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myadddr));  
  
 //3. 阻塞接收数据recvfrom()  
 char buf[100] = { 0 };  
 //不想知道是谁给我发送的消息  
 recvfrom(sockfd, buf, sizeof(buf), 0, NULL, NULL);  
 //想要知道是谁给我发送的消息  
 struct sockaddr\_in youaddr = { 0 };  
 int len = sizeof(youaddr);  
 recvfrom(sockfd, buf, sizeof(buf), 0, (struct sockaddr\*)&youaddr, &len);  
  
 //4. 关闭套接字  
 close(sockfd);  
   
  
 发送端  
 //1.创建一个数据报套接字socket函数  
 int sockfd = socket(AF\_INET, SOCK\_DGREAM, 0);   
 //2. 发送数据：在发送之前已经提前知道了接收方的IP地址和端口号  
 struct sockaddr\_in toaddr = { 0 };  
 toaddr.sin\_family = AF\_INET;  
 toaddr.sin\_port = htons(55555);  
 toaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");  
 //sendto函数发送数据，给谁发送，取决于toaddr变量里面保存的IP地址和端口号，存的是谁，就是给谁发送  
 sendto(sockfd, buf, strlen(buf)+1, 0, (struct sockaddr\*)&toaddr, sizeof(toaddr));  
 //3.关闭套接字  
 close(sockfd);

# -----------------------------------

# Day02

# 一. TCP通信

## 1. TCP概述

(1)什么是TCP？？  
   
 UDP是一个面向无连接的不可靠的传输协议  
   
 TCP是一个面向有连接的可靠的传输协议  
 TCP优点：传输数据安全，可靠，不容已丢失数据  
 TCP缺点：传输速度慢  
   
(2)TCP协议位于哪一层？  
   
 传输层  
   
(3)为什么TCP是可靠的？  
   
 1. 三次握手、四次挥手  
 2. 重传确认  
   
(4)TCP缺点？  
   
 传输效率低，传输速度慢

## 2. TCP通信



TCP通信分为服务器端和客户端  
   
1. 搭建TCP服务器端  
   
(1)创建一个流式套接字 SOCK\_STREAM socket()  
(2)绑定自己本机的IP地址和端口号 bind()  
(3)设置监听 listen()  
(4)阻塞等待连接 accept()  
(5)接收数据 recv()  
(6)发送数据 send()   
(7)关闭套接字 close()  
   
2. 搭建TCP客户端  
(1) 创建一个流式套接字 socket()   
(2) 连接服务器 connect()   
(3) 发送数据 send()  
(4) 接收互数据 recv()  
(5) 关闭套接字 close()

### 2.1 TCP服务器端

(1)创建一个流式套接字 SOCK\_STREAM socket()  
(2)绑定自己本机的IP地址和端口号 bind()  
(3)设置监听 listen()  
(4)阻塞等待连接 aceept()  
(5)接收数据 recv()  
(6)发送数据 send()  
(7)关闭套接字 close()

#### (1)创建一个流式套接字 socket()

//调用socket函数  
int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);  
if(sockfd == -1)  
{  
 perror("socket failed");  
 exit(-1);  
}  
注意点在第二个参数套接字类型：  
TCP通信套接字类型是 流式套接字 SOCK\_STREAM   
UDP通信套接字类型是 数据报套接字 SOCK\_DGRAM

#### (2)绑定自己本机的IP地址和端口号 bind()

调用同UDP通信的bind函数调用  
struct sockaddr\_in myaddr = { 0 };//用来保存自己的IP地址和端口号   
myaddr.sin\_family = AF\_INET;//family协议选择IPv4  
myaddr.sin\_port = htons(5555);//端口号   
myaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");  
//myaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  
//myaddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); //"127.0.0.1"IP地址是主机环回地址  
bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myaddr));

#### (3)设置监听 listen()

#include <sys/types.h> /\* See NOTES \*/  
#include <sys/socket.h>  
   
int listen(int sockfd, int backlog);  
  
//调用  
listen(sockfd, 5);  
   
(1)功能: 允许同时连接的最大个数   
   
(2)参数说明:  
 int sockfd //socket函数的返回值  
 int backlog //允许同时连接的最大的个数  
  
(3)返回值:  
 成功: 0  
 失败: -1

#### (4) 阻塞等待连接accept()

#include <sys/types.h> /\* See NOTES \*/  
#include <sys/socket.h>  
  
int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);  
   
//调用  
//对第2,3个参数的理解，类似于 UDP里面的youaddr变量  
//不想知道连接服务器的那个客户端的IP地址和端口号,直接赋值为NULL  
int newsockfd = accept(sockfd, NULL, NULL);  
   
//想要知道连接服务器的那个客户端的IP地址和端口号，可以定义一个结构体变量，参数上的地址传递，得到IP地址和端口号  
struct sockaddr\_in clientaddr = { 0 };  
int len = sizeof(clientaddr);  
int newsockfd = accept(sockfd, (struct sockaddr\*)&client\_addr, &len);  
   
(1)功能: 阻塞等到客户端连接  
 一但客户端连接成功，就解除阻塞，并且返回一个新的套接字描述符  
   
(2)参数说明:  
   
 int sockfd //socket函数的返回值   
 struct sockaddr \*addr //保存的是连接的那个客户端的IP地址和端口号   
 socklen\_t \*addrlen //客户端的IP地址和端口号的长度  
   
(3)返回值://##非常非常重要  
   
 成功：  
 返回一个非负的整数，是一个新的套接字描述符 newsockfd,  
 在TCP通信中只要有了newsockfd的值就可以与客户端通信，不需要知道客户端的IP地址和端口号  
  
 失败： -1

#### (5)阻塞接收数据 recv()

#include <sys/types.h>  
#include <sys/socket.h>  
  
ssize\_t recv(int sockfd, void \*buf, size\_t len, int flags);  
  
//调用  
char buf[100] = { 0 };//接收数据存放的位置  
recv(newsockfd, buf, sizeof(buf), 0); //注意此处用的新的newsockfd,accept返回值  
//第三个参数，最多能够接收的字节数  
  
(1)功能: 接收数据  
(2)参数说明:   
   
 int sockfd //服务器端，在接收数据的时候，用的是新的套接字描述符 accept函数的返回值 newsockfd  
 void \*buf //接收数据存放的位置  
 size\_t len //接收数据的最大长度  
 int flags //0 阻塞接收  
   
(3)返回值:  
 成功: 实际接收到的字节数  
 失败: -1;

#### (6)发送数据 send()

(6)发送数据  
   
#include <sys/types.h>  
#include <sys/socket.h>  
  
ssize\_t send(int sockfd, const void \*buf, size\_t len, int flags);  
  
//调用   
char buf[100] = "hello world!";  
send(newsockfd, buf, strlen(buf)+1, 0);//注意此处用的新的newsockfd,accept返回值  
//第三个参数代表实际发送的字节数  
(1)功能: 发送数据   
(2)参数说明:  
   
 int sockfd //服务器端，在发送数据的时候，用的是新的套接字描述符 accept函数的返回值 newsockfd  
 const void \*buf //发送数据存放的位置  
 size\_t len //实际发送的字节数   
 int flags //通常 0   
(3)返回值:  
 成功: 实际发送的字节数  
 失败: -1

#### (7) 关闭套接字close()

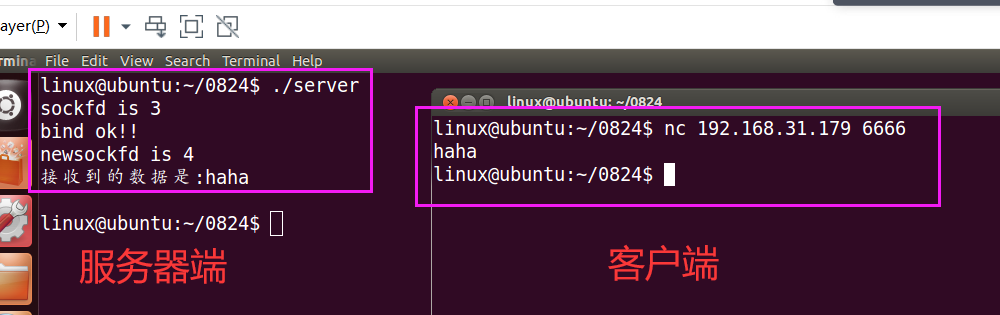
close(sockfd);  
close(newsockfd);

#### server.c不想知道连接服务器的客户端是谁

#include "my.h"  
  
int main(int argc, const char \*argv[])  
{  
 char buf[100] = { 0 };//用来保存接收到的数据  
 //1.创建一个流式套接字  
 int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed!!");  
 exit(-1);  
 }  
 printf("sockfd is %d\n",sockfd);  
 //2.绑定自己本机的IP地址和端口号  
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(6666);  
// myaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");  
 myaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  
 int ret = bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myaddr));  
 if(ret == -1)  
 {  
 perror("bind failed");  
 exit(-1);  
 }  
 printf("bind ok!!\n");  
 //3.设置监听 允许同时最大的连接个数  
 listen(sockfd, 5);  
 //4.阻塞等待连接  
 //将accept的第2 3个参数赋值为NULL，代表不想知道是哪个客户端连接服务器  
 int newsockfd = accept(sockfd, NULL, NULL);  
 printf("newsockfd is %d\n",newsockfd);  
 //5.阻塞接收数据  
 recv(newsockfd, buf, sizeof(buf), 0);  
 printf("接收到的数据是:%s\n",buf);  
 //6.关闭套接字  
 close(sockfd);  
 close(newsockfd);  
 return 0;  
}

#### client.c

nc 192.168.31.179 6666 //192.168.31.179 6666 代表客户端要连接的那个服务器的IP地址和端口号



#### server.c 想要知道连接服务器的客户端是谁

#include "my.h"  
  
int main(int argc, const char \*argv[])  
{  
 char buf[100] = { 0 };//用来保存接收到的数据  
 //1.创建一个流式套接字  
 int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed!!");  
 exit(-1);  
 }  
 printf("sockfd is %d\n",sockfd);  
 //2.绑定自己本机的IP地址和端口号  
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(6666);  
// myaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");  
 myaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  
 int ret = bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myaddr));  
 if(ret == -1)  
 {  
 perror("bind failed");  
 exit(-1);  
 }  
 printf("bind ok!!\n");  
 //3.设置监听 允许同时最大的连接个数  
 listen(sockfd, 5);  
 //4.阻塞等待连接  
 //想要获取连接服务器的这个客户端的IP地址和端口号  
 //accept函数解除阻塞后,连接服务器的这个客户端的IP地址和端口号会保存到youaddr中  
 struct sockaddr\_in youaddr = { 0 };  
 int len = sizeof(youaddr);  
 int newsockfd = accept(sockfd, (struct sockaddr\*)&youaddr, &len);  
 printf("newsockfd is %d\n",newsockfd);  
 //5.阻塞接收数据  
 recv(newsockfd, buf, sizeof(buf), 0);  
 printf("接收到的数据来自于客户端%s-%d: %s\n",inet\_ntoa(youaddr.sin\_addr),ntohs(youaddr.sin\_port),buf);  
 //6.关闭套接字  
 close(sockfd);  
 close(newsockfd);  
 return 0;  
}



### 2.2 TCP客户端

(1) 创建一个流式套接字 socket()   
(2) 连接服务器 connect()   
(3) 发送数据 send()  
(4) 接收数据 recv()  
(5) 关闭套接字 close()

#### (1) 创建一个流式套接字socket()

同服务器端一样  
int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

#### (2) 连接服务器connect()

#include <sys/types.h> /\* See NOTES \*/  
#include <sys/socket.h>  
   
(1)功能: 连接到服务器  
   
(2) 参数说明:  
   
int connect(int sockfd, const struct sockaddr \*addr,socklen\_t addrlen);  
  
 int sockfd //socket函数的返回值   
 const struct sockaddr \*addr //提前知道的服务器的IP地址和端口号  
 socklen\_t addrlen //IP地址和端口号的长度  
  
 //调用:  
  
 //在连接服务器之前，客户端已经知道了服务器的IP地址和端口号  
 //所以我们定义一个toaddr,提前赋值服务器的IP地址和端口号  
 struct sockaddr\_in toaddr = { 0 };  
 toaddr.sin\_family = AF\_INET;  
 toaddr.sin\_port = htons(6666);   
 toaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");  
 //connect函数，通过toaddr里面保存的IP地址和端口号，去找到服务器，并连接  
 //toaddr变量中保存的是哪个服务器的IP地址和端口号，连接的就是哪个服务器  
 connect(sockfd, (struct sockaddr\*)&toaddr, sizeof(toaddr));  
   
(3)返回值:  
 成功: 返回 0   
 失败: 返回 -1

#### (3) 发送数据send()

char buf[100] = "hello";  
send(sockfd, buf, strlen(buf)+1, 0); //第三个参数，实际发送的字节数

#### (4) 接收数据recv()

char buf[100] = { 0 };  
recv(sockfd, buf, sizeof(buf), 0); //第三个参数，最多能够接收的字节数

#### (5) 关闭套接字close()

close(sockfd);

#### server.c

#include "my.h"  
  
int main(int argc, const char \*argv[])  
{  
 char buf[100] = { 0 };//用来保存接收到的数据  
 //1.创建一个流式套接字  
 int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed!!");  
 exit(-1);  
 }  
 printf("sockfd is %d\n",sockfd);  
 //2.绑定自己本机的IP地址和端口号  
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(6666);  
// myaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");  
 myaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  
 int ret = bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myaddr));  
 if(ret == -1)  
 {  
 perror("bind failed");  
 exit(-1);  
 }  
 printf("bind ok!!\n");  
 //3.设置监听 允许同时最大的连接个数  
 listen(sockfd, 5);  
 //4.阻塞等待连接  
 //想要获取连接服务器的这个客户端的IP地址和端口号  
 //accept函数解除阻塞后,连接服务器的这个客户端的IP地址和端口号会保存到youaddr中  
 struct sockaddr\_in youaddr = { 0 };  
 int len = sizeof(youaddr);  
 int newsockfd = accept(sockfd, (struct sockaddr\*)&youaddr, &len);  
 printf("newsockfd is %d\n",newsockfd);  
 //5.阻塞接收数据  
 recv(newsockfd, buf, sizeof(buf), 0);  
 printf("接收到的数据来自于客户端%s-%d: %s\n",inet\_ntoa(youaddr.sin\_addr),ntohs(youaddr.sin\_port),buf);  
 //6.关闭套接字  
 close(sockfd);  
 close(newsockfd);  
 return 0;  
}

#### client.c

#include "my.h"  
  
int main(int argc, const char \*argv[])  
{  
 char buf[100] = { 0 };  
 //1.创建一个式套接字  
 int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed");  
 exit(-1);  
 }  
 //2.连接服务器  
 //(1)在连接服务器之前,客户端已经提前知道了服务器的IP地址和端口号  
 //toaddr里面赋值的是哪个服务器的IP地址和端口号,连接的就是哪个服务器  
 struct sockaddr\_in toaddr = { 0 };  
 toaddr.sin\_family = AF\_INET;  
 toaddr.sin\_port = htons(6666);  
 toaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");  
 int ret = connect(sockfd, (struct sockaddr\*)&toaddr, sizeof(toaddr));  
 if(ret == -1)  
 {  
 perror("connect failed");  
 exit(-1);  
 }  
 //3.输入数据,并发送给服务器  
 printf("请您输入要发送给服务器的话:\n");  
 scanf("%s",buf);  
 send(sockfd, buf, strlen(buf)+1, 0);  
  
 //4.关闭套接字  
 close(sockfd);  
 return 0;  
}

### #练习1:

服务器端: 实现服务器端循环接收客户端发送过来的数据并打印，打印之后并将接收到的数据再回传给客户端  
  
客户端: 从参数（命令行传参）中提取ip地址和端口号(可以连接任意服务器),可以循环从键盘输入数据，发数据给服务器同时还可以接收服务器回传回来的数据，并打印  
  
客户端 ./client 192.168.31.168 8888 //连接的服务器绑定的IP地址是192.168.31.168 端口号8888  
gcc -o client client.c -lpthread

#### client.c

#include "my.h"  
  
//线程负责一直接收客户端回传回来的数据,并打印  
void\* recvThread(void\* p)  
{  
 int sockfd = \*((int\*)p);  
 char buf[100] = { 0 };//用来保存接收到的数据  
 while(1)  
 {  
 recv(sockfd, buf, sizeof(buf), 0);  
 printf("服务器回传回来的数据是%s\n",buf);//打印一定要加'\n'  
 }  
}  
  
int main(int argc, const char \*argv[])  
{  
 if(argc != 3)  
 {  
 printf("忘记传递参数了 ./client 192.168.31.179 6666\n");  
 exit(-1);  
 }  
  
 pthread\_t id;  
 char buf[100] = { 0 };  
 //1.创建一个式套接字  
 int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed");  
 exit(-1);  
 }  
 //2.连接服务器  
 //(1)在连接服务器之前,客户端已经提前知道了服务器的IP地址和端口号  
 //toaddr里面赋值的是哪个服务器的IP地址和端口号,连接的就是哪个服务器  
 struct sockaddr\_in toaddr = { 0 };  
 toaddr.sin\_family = AF\_INET;  
 toaddr.sin\_port = htons(atoi(argv[2]));  
 toaddr.sin\_addr.s\_addr = inet\_addr(argv[1]);  
 int ret = connect(sockfd, (struct sockaddr\*)&toaddr, sizeof(toaddr));  
 if(ret == -1)  
 {  
 perror("connect failed");  
 exit(-1);  
 }  
 //创建一个线程,专门负责接收服务器回传回来的数据  
 pthread\_create(&id, NULL, recvThread, &sockfd);  
 //客户端的主进程只负责发送数据  
 while(1)  
 {  
 //3.输入数据,并发送给服务器  
 scanf("%s",buf);  
 send(sockfd, buf, strlen(buf)+1, 0);  
 }  
  
 //4.关闭套接字  
 close(sockfd);  
 return 0;  
}

#### server.c

#include "my.h"  
  
int main(int argc, const char \*argv[])  
{  
 char buf[100] = { 0 };//用来保存接收到的数据  
 //1.创建一个流式套接字  
 int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed!!");  
 exit(-1);  
 }  
 printf("sockfd is %d\n",sockfd);  
 //2.绑定自己本机的IP地址和端口号  
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(1234);  
// myaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");  
 myaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  
 int ret = bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myaddr));  
 if(ret == -1)  
 {  
 perror("bind failed");  
 exit(-1);  
 }  
 printf("bind ok!!\n");  
 //3.设置监听 允许同时最大的连接个数  
 listen(sockfd, 5);  
 //4.阻塞等待连接  
 //想要获取连接服务器的这个客户端的IP地址和端口号  
 //accept函数解除阻塞后,连接服务器的这个客户端的IP地址和端口号会保存到youaddr中  
 struct sockaddr\_in youaddr = { 0 };  
 int len = sizeof(youaddr);  
 int newsockfd = accept(sockfd, (struct sockaddr\*)&youaddr, &len);  
 printf("newsockfd is %d\n",newsockfd);  
 while(1)  
 {  
 //5.阻塞接收数据  
 recv(newsockfd, buf, sizeof(buf), 0);  
 printf("接收到的数据来自于客户端%s-%d: %s\n",inet\_ntoa(youaddr.sin\_addr),ntohs(youaddr.sin\_port),buf);  
 //将接收到的数据回传客户端  
 send(newsockfd, buf, strlen(buf)+1, 0);  
 }  
 //6.关闭套接字  
 close(sockfd);  
 close(newsockfd);  
 return 0;  
}

### 抛出问题：上面的服务器有什么缺点??

练习1服务器的缺点？？  
服务器端只能支持1个客户端连接服务器  
我们要解决 服务器支持多个客户端连接，与之交互

# 二. 循环服务器

while(1)  
{  
 int newsockfd = accept(sockfd, NULL, NULL);//阻塞函数  
 printf("newsockfd is %d\n",newsockfd);  
 recv(newsockfd buf, sizeof(buf), 0);//阻塞函数  
 printf("接接收到的数据是%s\n",buf);  
}  
终端：客户端1， ./client 192.168.31.185 1234 //连接成功，accept解除阻塞，服务器阻塞在recv  
 客户端2. ./client 192.168.31.185 1234 //连接失败

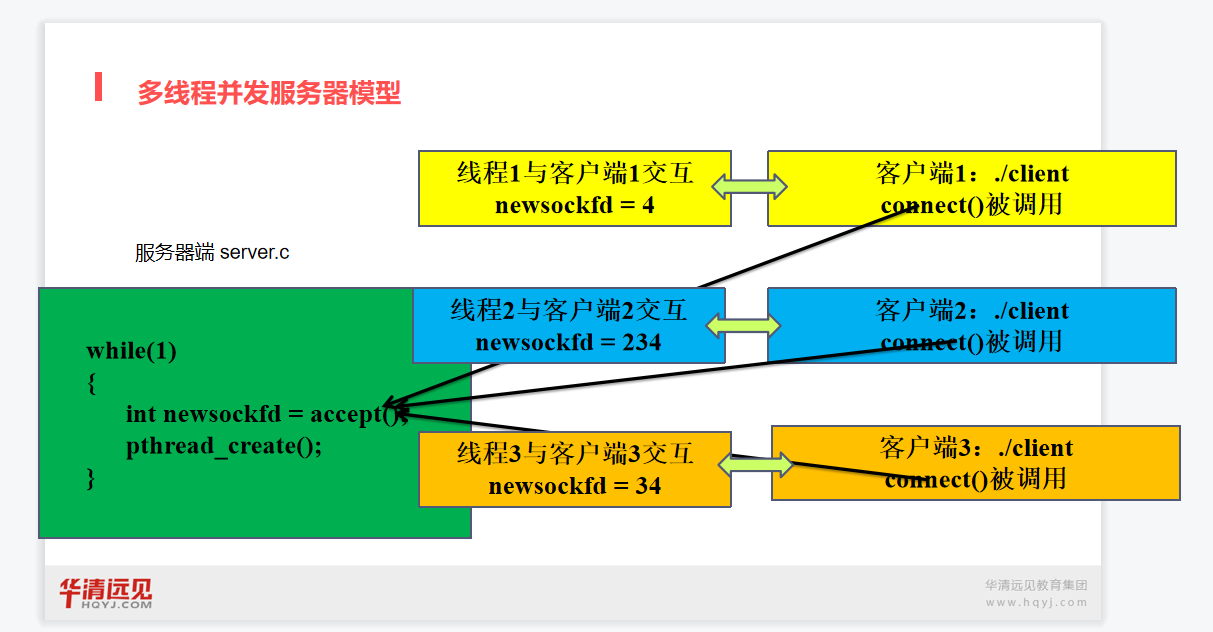
# 三. 多线程并发服务器

多线程并发服务器  
 //线程专门负责与客户端进行交互  
 void\* doClientThread(void\* p)  
 {  
 while(1)  
 {  
 recv(newsockfd buf, sizeof(buf), 0);//阻塞函数  
 printf("接接收到的数据是%s\n",buf);  
 }  
 }  
 int main()  
 {  
 while(1)  
 {  
 int newsockfd = accept(sockfd, NULL, NULL);//阻塞等待被连接函数  
 printf("newsockfd is %d\n",newsockfd);  
 pthread\_create(&id, NULL, doClientThread, NULL);//创建一个线程  
 }  
 //accept函数只要解除阻塞一次，就说明有一个客户端连接服务器成功, 打印输出newsockfd，然后执行pthread\_create创建出了一个子线程与客户端交互  
 }

## 1.多线程并发服务器思想

只要有一个客户端连接服务器成功， accept函数就解除阻塞一次，就创建一个子线程，与该客户端进行交互

## 2. 模型图

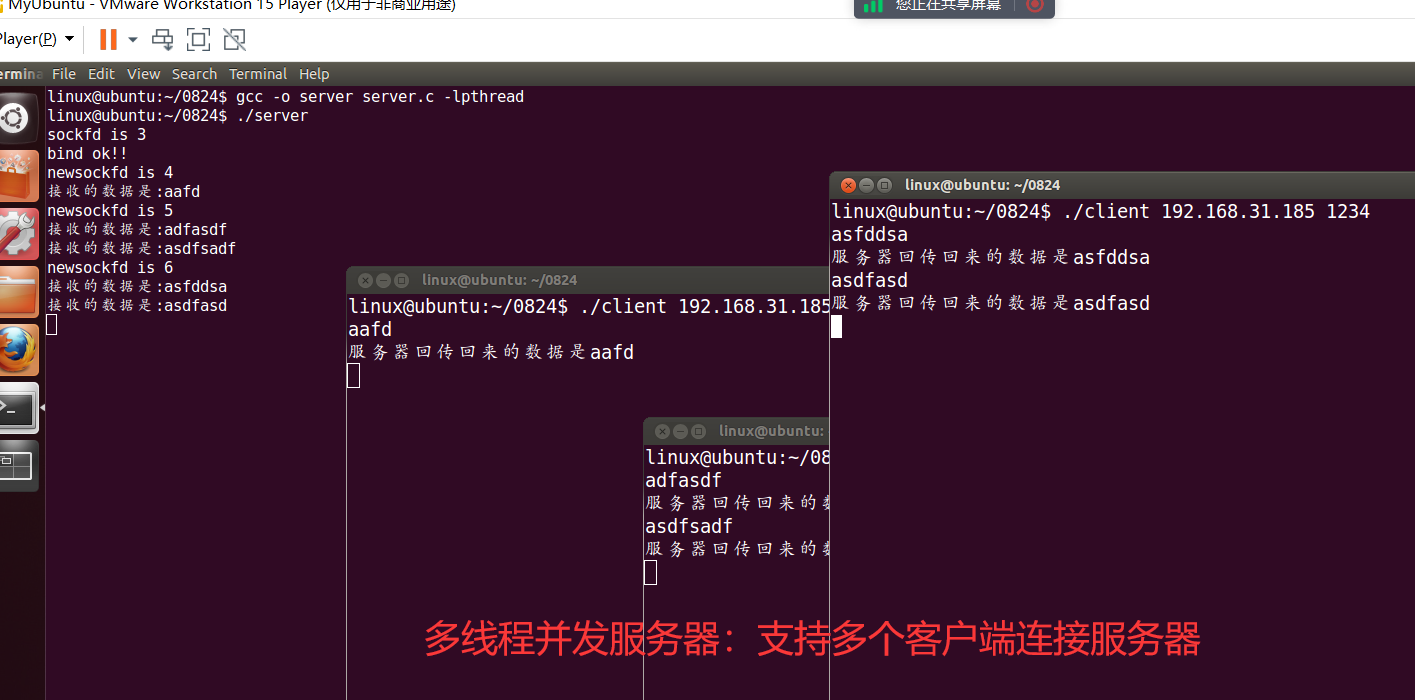


#### server.c多线程并发服务器

#include "my.h"  
  
void\* doClientThread(void\* p)  
{  
 int newsockfd = \*((int\*)p);  
 char buf[100] = { 0 };//用来保存接收到的数据  
 while(1)  
 {  
 int ret = recv(newsockfd, buf, sizeof(buf), 0);  
 if(ret > 0)//说明接收到了数据,让recv解除阻塞  
 {  
 printf("接收的数据是:%s\n",buf);  
 send(newsockfd, buf, strlen(buf)+1, 0);//回传给客户端  
 }  
 else//说明是客户端断开连接,让recv解除阻塞  
 {  
 printf("客户端断开连接,对应的新的套接字描述符%d\n",newsockfd);  
 close(newsockfd);  
 pthread\_exit(NULL);//结束当前所在线程  
 }  
 }  
}  
  
  
int main(int argc, const char \*argv[])  
{  
 //1.创建一个流式套接字  
 int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed!!");  
 exit(-1);  
 }  
 printf("sockfd is %d\n",sockfd);  
 //2.绑定自己本机的IP地址和端口号  
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(1234);  
 // myaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");  
 myaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  
 int ret = bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myaddr));  
 if(ret == -1)  
 {  
 perror("bind failed");  
 exit(-1);  
 }  
 printf("bind ok!!\n");  
 //3.设置监听 允许同时最大的连接个数  
 listen(sockfd, 5);  
   
 //多线程并发服务器  
 //主程序只负责阻塞等待客户端连接  
 pthread\_t id;  
 while(1)  
 {  
 int newsockfd = accept(sockfd, NULL, NULL);  
 printf("newsockfd is %d\n",newsockfd);  
 pthread\_create(&id, NULL, doClientThread, &newsockfd);  
 }  
  
 return 0;  
}

#### client.c

练习1客户端代码



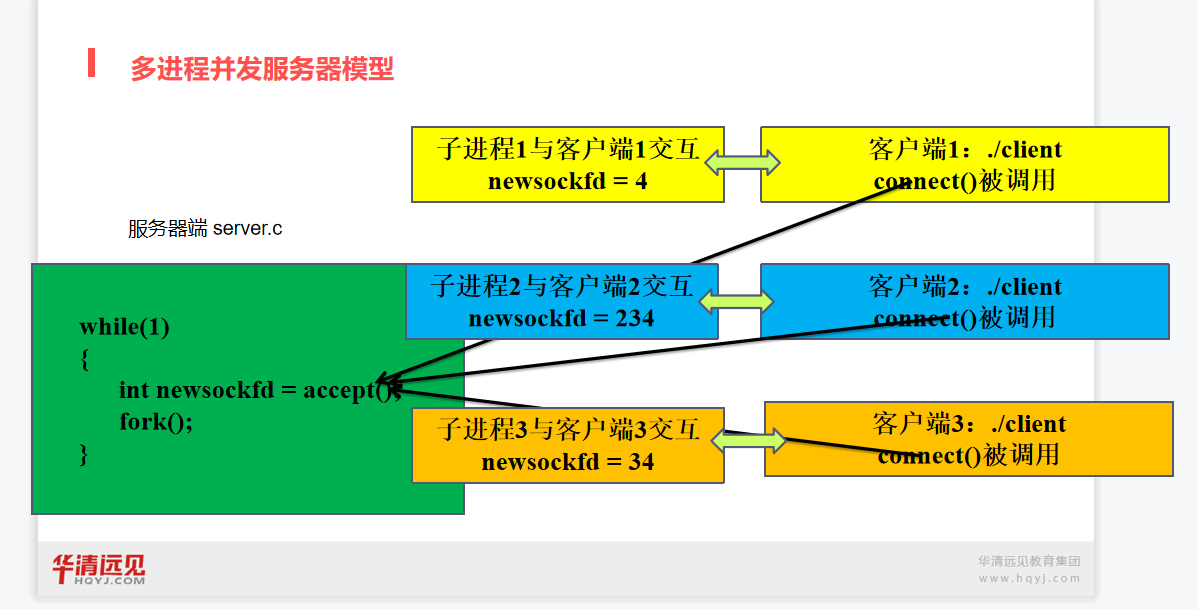
# 四. 多进程并发服务器

多进程并发服务器  
   
 void doClient(int newsockfd)  
 {  
 char buf[100] = { 0 };  
 while(1)  
 {  
 recv(newsockfd buf, sizeof(buf), 0);//阻塞函数  
 printf("接接收到的数据是%s\n",buf);  
 send(newsockfd, buf, strlen(buf)+1, 0);//回传  
 }  
 }  
 int main()  
 {  
 while(1)  
 {  
 int newsockfd = accept(sockfd, NULL, NULL);//阻塞等待被连接函数  
 printf("newsockfd is %d\n",newsockfd);  
 pid\_t ret = fork();//创建一个子进程  
 if(ret == -1)  
 {  
 perror("fork failed");  
 exit(-1);  
 }  
 else if(ret > 0)//父进程就是当前./server 就阻塞等待连接  
 {  
 ;   
 }  
 else if(ret == 0)//说明当前是子进程  
 {  
 doClient(newsockfd);  
 }  
 }  
 //accept函数只要解除阻塞一次，就说明有一个客户端连接服务器成功, 打印输出newsockfd，然后执行fork函数创建出了一个子进程与客户端交互  
 }

## 1.多进程并发服务器思想

只要有一个客户端连接服务器成功， accept函数就解除阻塞一次，就创建一个子进程，与该客户端进行交互

## 2. 模型图



#### server.c多进程并发服务器

#include "my.h"  
  
//与客户端交互的函数  
void doClient(int newsockfd)  
{  
 char buf[100] = { 0 };//用来保存接收到的数据  
 while(1)  
 {  
 int ret = recv(newsockfd, buf, sizeof(buf), 0);  
 if(ret > 0)//说明接收到了数据,让recv解除阻塞  
 {  
 printf("接收的数据是:%s\n",buf);  
 send(newsockfd, buf, strlen(buf)+1, 0);//回传给客户端  
 }  
 else//说明是客户端断开连接,让recv解除阻塞  
 {  
 printf("客户端断开连接,对应的新的套接字描述符%d\n",newsockfd);  
 close(newsockfd);  
 exit(0);//结束当前的子进程  
 }  
 }  
}  
  
  
int main(int argc, const char \*argv[])  
{  
 //1.创建一个流式套接字  
 int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed!!");  
 exit(-1);  
 }  
 printf("sockfd is %d\n",sockfd);  
 //2.绑定自己本机的IP地址和端口号  
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(1234);  
 // myaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");  
 myaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  
 int ret = bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myaddr));  
 if(ret == -1)  
 {  
 perror("bind failed");  
 exit(-1);  
 }  
 printf("bind ok!!\n");  
 //3.设置监听 允许同时最大的连接个数  
 listen(sockfd, 5);  
   
 //多进程并发服务器  
 //主程序只负责阻塞等待客户端连接  
 while(1)  
 {  
 int newsockfd = accept(sockfd, NULL, NULL);  
 printf("newsockfd is %d\n",newsockfd);  
 pid\_t ret = fork();  
 if(ret == -1)  
 {  
 perror("fork failed");  
 exit(-1);  
 }  
 else if(ret == 0)//说明是子进程,与客户端交互  
 {  
 doClient(newsockfd);//与客户端交互的函数  
 }  
 }  
  
 return 0;  
}

#### client.c

练习1客户端代码

# 五. select I/O多路复用

## 1. I/O多路复用优势

多线程并发服务器和多进程并发服务器，当客户端连接数量大的时候，多线程并发服务器和多进程并发服务器依然耗费资源较多  
用select函数实现I/O多路复用，也可以实现并发服务器的效果，节省资源  
   
并发服务器：运行服务器支持多个客户端连接

## 2. 原理

多路复用原理:  
 只有一个阻塞程序，可以接收accept, recv, 把这些需要阻塞的函数归到一起，用一个函数来阻塞,select 函数，能实现总的阻塞(fd1, fd2, fd3, fd4, fd5, fd6), 当某一个条件成立时，就解除阻塞  
  
  
用select函数实现I/O多路复用，select是一个总的阻塞函数，将读阻塞相关的原因归结到一起  
accept和recv都是读阻塞相关的，读阻塞相关的文件描述符是一个集合，当select解除阻塞后，集合只会剩下一个文件描述符，剩下的这个文件描述符就是导致select解除阻塞的具体原因，取出剩下的这个文件描述符的值，分析是 新客户端连接服务器还是已经连接的客户端给服务器发送消息，如果是剩下的这个文件描述符i == sockfd，说明新的客户端连接服务器，其他的值，已经连接的客户端给服务器发送消息

## 3. select函数

select 函数，能实现总的阻塞(fd1, fd2, fd3, fd4, fd5, fd6), 当某一个条件成立时，就解除阻塞  
如果select函数解除阻塞？？思考，是什么原因导致的select解除阻塞   
   
(1) 新的客户端连接服务器导致select解除阻塞  
(2) 已经连接服务器成功的客户端，给服务器发送数据导致select解除阻塞  
  
int select(int maxfd, fd\_set \*read\_fds, fd\_set \*write\_fds, fd\_set \*except\_fds, struct timeval \*timeout);  
  
功能: 阻塞等待 一堆 文件描述符，只要有一个数据过来，就解除阻塞  
参数:  
 maxfd 最大文件描述符 + 1  
 read\_fds 可能引起阻塞的读相关的文件描述符集(accept recv recvfrom read)  
 write\_fds 可能引起阻塞的写相关的文件描述符集(send, sendto write)  
 except\_fds 可能引起阻塞的其他的文件描述符集  
 timeout NULL 表示一直阻塞，  
 其他值：可以设置阻塞时间  
 select执行完：read\_fds中保存引起阻塞解除的文件描述符  
  
read\_fds  
//read 代表的是读相关的文件描述符   
// fd 代表的是文件描述   
// s 代表的一个集合，多个文件描述符  
如何设置read\_fds, write\_fds, except\_fds  
   
fd\_set rdfs; //先定义变量rdfs，这个是一个读文件描述符集  
int ret = select(maxfd + 1, &fdfs, NULL, NULL, NULL);  
  
FD\_ZERO(&rdfs); //此宏能清空rdfs 列表  
FD\_SET(fd, &rdfs); //将文件描述符fd，加入到rdfs中  
FD\_SET(fd1, &rdfs); //将文件描述符fd1，加入到rdfs中  
FD\_CLR(fd1, &rdfs); //将fd1从rdfs中删除  
FD\_ISSET(fd1, &rdfs)) //判断fd1这个文件描述符，是否在rdfs集合存在

### server.c

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include <sys/types.h>  
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <arpa/inet.h>  
  
#define MAXLINE 100  
typedef struct sockaddr SA;  
  
int main(int argc, char \*\*argv)  
{   
 int sockfd, newsockfd, maxfd, i, nbyte;  
 struct sockaddr\_in myaddr;  
 char buf[MAXLINE];  
 fd\_set global\_rdfs, current\_rdfs;  
 //集合global\_rdfs global 总的 全局的  
 //集合current\_rdfs current 当前的   
 //创建socket  
 if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)  
 {  
 perror("fail to socket");  
 exit(-1);  
 }  
 printf("sockfd is %d\n",sockfd);  
 //将myaddr结构体被bzero函数，初始化为空 struct sockaddr\_in myaddr = { 0 };  
 bzero(&myaddr, sizeof(myaddr)); // char buf[100] = "hello "; bzero(buf,sizeof(buf)); memset(buf,0,sizeof(buf));  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY); //htonl(INADDR\_ANY);  
 myaddr.sin\_port = htons(55555); /\* port number \*/  
 //绑定IP和端口号   
 //SA ---> struct sockaddr  
 if(bind(sockfd, (SA \*)&myaddr, sizeof(myaddr)) < 0)  
 {  
 perror("fail to bind");  
 exit(-1);  
 }  
 //监听  
 listen(sockfd, 5);  
 FD\_ZERO(&global\_rdfs); //清空  
 FD\_SET(sockfd, &global\_rdfs); //将sockfd 添加到 global\_rdfs 中  
 maxfd = sockfd; //最大的文件描述符也就是3  
 while(1)  
 {  
 current\_rdfs = global\_rdfs; //global 实际保存 current\_rdfs //临时  
 if (select(maxfd+1, &current\_rdfs, NULL, NULL, 0) < 0) //阻塞等待  
 {  
 perror("fail to select");  
 exit(-1);  
 }  
 else //解除阻塞了，一但select解除阻塞，那么current\_rdfs这个集合里面，只会剩下一个文件描述  
 //剩下的这个文件描述符，就是导致解除阻塞原因的描述符  
 {  
 for (i=0; i<=maxfd; i++) //for循环遍历0-maxfd每一个文件描述符，判断是否在集合上current\_rdfs中出现(注意select解除阻塞后，current\_rdfs中只有一个文件描述符)  
 {  
 if (FD\_ISSET(i, &current\_rdfs)) //判断i这个文件描述符，是否在current\_rdfs集合存在  
 {  
 if (i == sockfd) // new connection is coming //有新客户连接请求  
 {  
 newsockfd = accept(sockfd, NULL, NULL);//new   
 printf("newsockfd is %d\n",newsockfd);  
 FD\_SET(newsockfd, &global\_rdfs);//将newsockfd添加到global\_rdfs中  
 maxfd = maxfd > newsockfd ? maxfd : newsockfd;  
   
 }  
 else // client send message //客户端发送来消息，引起的解除阻塞  
 {  
 if((nbyte = recv(i, buf, sizeof(buf), 0)) <= 0)//recv返回值 <=0说明客户端断开连接  
 {  
 close(i); //关闭这个newsockfd  
 FD\_CLR(i, &global\_rdfs);//将这个描述符，从global\_rdfs中删除  
 }  
 else //接收到了数据  
 {   
 printf("recv from client is %s\n",buf);  
 send(i, buf, strlen(buf)+1, 0);//把收到的数据回传给客户端  
 }  
 }  
 break;  
 }  
 } // end for  
 }  
 }  
  
 return 0;  
}

### client.c

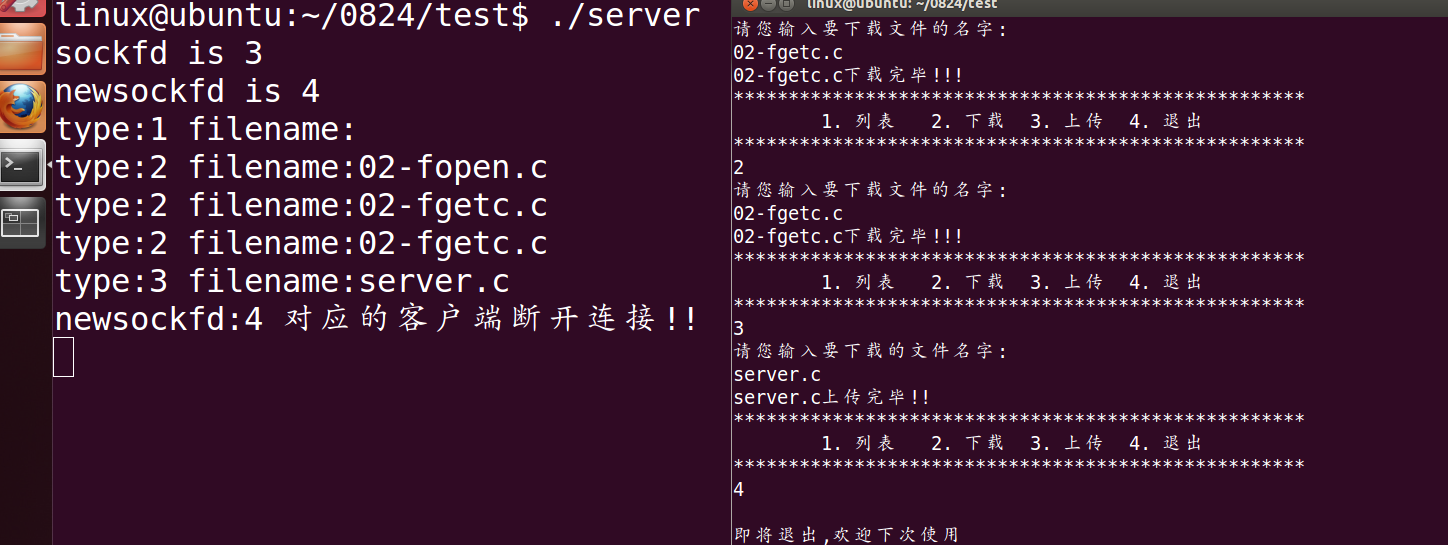
练习1客户端代码

# 六. UDP和TCP总结

TCP和UDP相同点:  
 TCP和UDP都位于传输层  
TCP和UDP不同点：  
 (1)TCP是面向连接的可靠的传输协议，UDP是面向无连接的不可靠的传输协议  
 (2)TCP使用的流式套接字SOCK\_STREAM, UDP使用的是数据报套接字SOCK\_DGRAM  
 (3)TCP 安全可靠，不容易丢失数据，传输速度慢，效率低  
 UDP 不可靠，容易丢失数据，传输速度快，效率高，实时性好

# @作业:

作业(ftp 文件上传下载) 上传文件、下载文件  
tcp 协议实现  
服务器: 设置一个目录，此目录专门存放可下载文件  
客户端 列表, 服务器读出所有文件(opendir readdir closedir)，发给客户端  
客户端 下载, 服务器读取hello.c内容，发给客户端(服务器 read 客户端用write)  
客户端 上传, 客户端读取hello.c内容，发给服务器(服务器 write 客户端 read)  
typedef struct   
{  
 int type; //1 列表 2 下载 3 上传 4 退出  
 char filename[50]; //保存文件名  
 char filedata[100]; //保存文件内容  
 int len; //包数据长度，用来保存每次实际读取到的字节数的长度  
}MSG;  
  
MSG s;  
//客户端 发送  
s.type = 1 //获取服务器提供的共享目录请求   
send(sockfd, &s, sizeof(s), 0);//请求发送给服务器  
s.type = 2;//下载文件请求， 要下载文件的名字  
strcpy(s.filename, "fseek.c");  
send(sockfd, &s, sizeof(s), 0);//请求发送给服务器  
s.type = 3;//上传文件请求， 要上传文件的名字  
strcpy(s.filename, "haha.c");  
send(sockfd, &s, sizeof(s), 0);//请求发送给服务器  
  
//服务器端 接收，根据 s.type的值，来区分客户端不同的请求，做出不同的处理



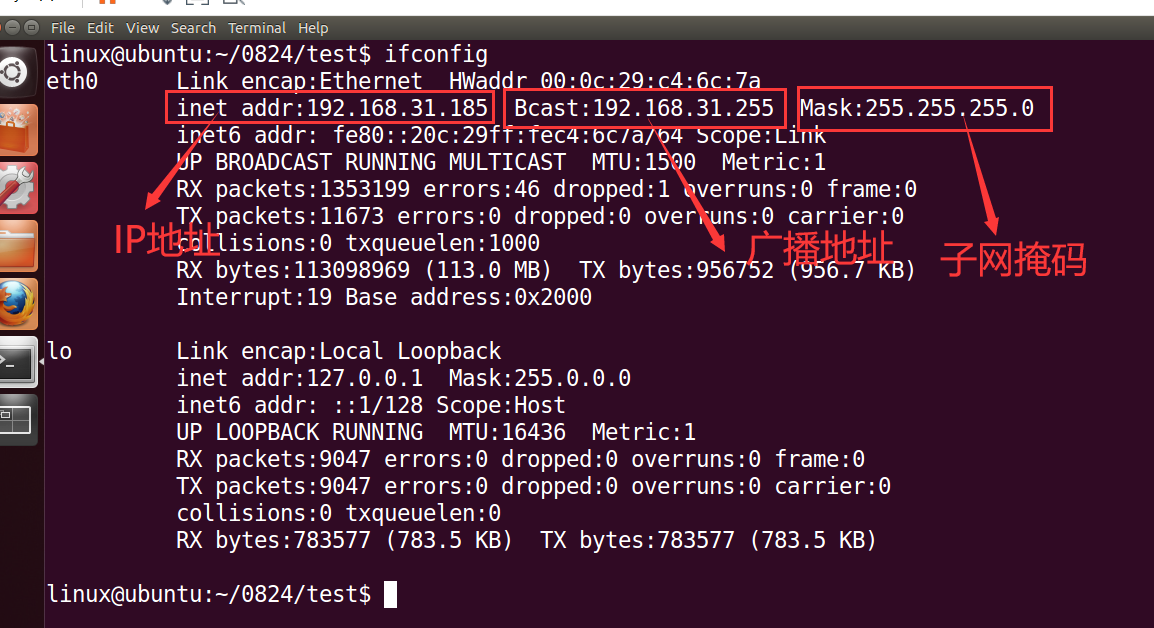
# @复习

1. TCP通信 Transmission Control Protocol //传输控制协议  
 TCP是一个面向连接的可靠的传输协议，位于传输层  
   
 TCP优点：安全可靠，不容易丢失数据  
 TCP缺点：效率低，传输速度慢  
   
 TCP为什么安全可靠？？？  
 (1)三次握手，四次挥手  
 (2)重传确认  
   
2. 搭建TCP通信过程  
 //服务器端  
 1.创建一个流式套接字socket();   
 //套接字类型是SOCK\_STREAM  
 2.绑定自己本机的IP地址和端口号bind();  
 3.设置监听listen();  
 listen(sockfd, 5);  
 4.阻塞等待连接accept();  
 //不想知道连接服务器的客户端是谁  
 int newsockfd = accept(sockfd, NULL, NULL);  
 //想要知道连接服务器的客户端是谁  
 struct sockaddr\_in youaddr = { 0 };  
 int len = sizeof(youaddr);  
 int newsockfd = accept(sockfd, (struct sockaddr\*)&youaddr， &len);  
 5.阻塞接收数据recv();  
 recv(newsockfd, buf, sizeof(buf), 0);  
 6.发送数据send();  
 send(newsockfd, buf, strlen(buf)+1, 0);  
 7.关闭套接字close();  
   
   
 //客户端  
 1.创建一个流式套接字socket();  
 2.连接服务器connect();  
 //客户端在连接服务器之前，已经提前知道了服务器的IP地址和端口号  
 struct sockaddr\_in toaddr = { 0 };  
 toaddr.sin\_family = AF\_INET;  
 toaddr.sin\_port = htons(55555);  
 toaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");  
 //connect函数连接哪个服务器，取决于toaddr里面保存的是哪个服务器的IP地址和端口号  
 connect(sockfd, (struct sockaddr\*)&toaddr, sizeof(toaddr));  
 3.发送数据send();  
 send(sockfd, buf, strlen(buf)+1, 0);  
 4.阻塞接收数据recv();  
 recv(sockfd, buf, sizeof(buf), 0);  
 5.关闭套接字close();  
  
  
3. 并发服务器  
 (1)什么是并发服务器？？  
 服务器端可以支持多个客户端进行连接  
   
 (2)实现并发服务器的方法  
   
 1. 多线程并发服务器  
 2. 多进程并发服务器  
 3. select函数 I/O多路复用  
   
 I/O 多路复用 > 多线程并发服务器 > 多进程并发服务器  
  
4. TCP写出一个多线程并发服务器  
   
 模型  
   
 while(1)  
 {  
 //主程序，一直处于阻塞等待连接状态，只要有客户端连接服务器成功，就立刻创建一个线程  
 //与该客户端交互  
 int newsockfd = accept(sockfd, NULL, NULL);  
 printf("newsockfd is %d\n",newsockfd);  
 pthread\_create(&id, NULL, thread, NULL);//创建一个线程  
 }  
 //面试问题： TCP服务器是怎么找到对应的客户端进行交互？？(TCP服务器是如何区分不同的客户端)  
 每个客户端连接服务器的时候，accept函数的返回值是新的套接字描述符newsockfd,新的  
 套接字描述符的值都不同，通过不同的newsockfd来区分不同的客户端  
 send(newsockfd, buf, strlen(buf)+1, 0);  
 recv(newsockfd, buf, sizeof(buf), 0);  
  
  
5. TCP写一个多进程并发服务器  
   
 while(1)  
 {  
 //主程序，一直处于阻塞等待连接状态，只要有客户端连接服务器成功，就立刻创建一个进程  
 //与该客户端交互  
 int newsockfd = accept(sockfd, NULL, NULL);  
 printf("newsockfd is %d\n",newsockfd);  
 pid\_t ret = fork();//创建一个子进程  
 if(ret == 0)  
 {  
 //子进程负责与客户端交互  
 }  
   
 }  
  
6. select I/O多路复用  
 原理：  
 select是阻塞函数，将所有的阻塞原因归结到一起，实现一个总的阻塞  
 TCP写出的并发服务器，引起阻塞的原因有两个,属于读阻塞相关的文件描述符集  
 (1) accept(sockfd, NULL, NULL); //sockfd   
 (2) recv(newsockfd, buf, sizeof(buf), 0);//newsockfd  
 select函数解除阻塞后，读文件描述符集合只剩下一个文件描述符，这个描述符就是导致select解除阻塞的原因  
 我们需要对select函数解除阻塞原因进行分析  
 如果读阻塞文件描述符集合中剩下的描述符是sockfd，说明是新的客户端连接服务器到select解除阻塞  
 如果读阻塞文件描述符集合中剩下的描述符不是sockfd，说明是以连接的客户端给服务器发送消息select解除阻塞  
   
7. TCP和UDP的相同点和不同点  
 相同点： TCP和UDP都位于传输层  
 不同点：  
 TCP是面向连接的可靠的传输协议  
 UDP是面向无连接的不可靠的传输协议  
   
 TCP使用的流式套接字SOCK\_STREAM  
 UDP使用的是数据报套接字SOCK\_DGRAM  
   
 TCP 安全可靠，传输不容易丢失数据，有三次握手、四次挥手、重传确认，但是效率低、传输速度慢  
 UDP 不可靠，传输数据容易丢失数据，但是UDP，传输效率高， 速度快，实时性好

# ----------------------------------

# Day03

# 一. 广播



BroadCast //广播  
单播 发送端 --- 接收端  
广播 发送端 --- 多个接收端  
  
unsigned int a;//没有负数，只能是正数，没有符号位  
00000000 00000000 00000000 00000000  
31 76543210   
11111111 11111111 11111111 11111111  
   
int a; //有符号，区分正数和负数,最高位代表的是符号位   
 //最高位是0，代表正数  
 //最高位是1, 代表负数  
0000000 00000000 00000000 0000000111 // 7  
1000000 00000000 00000000 0000000111 // -7  
   
unsigned char --->能表示的最大的数是几??? 11111111   
   
   
通过udp广播数据(只有udp可以广播，因为tcp需要连接)   
192.168.31.255  
广播地址：

## 0. 广播的发送端和接收端

发送端  
(1)创建一个数据报套接字SOCK\_DGRAM socket();  
(2)允许发送广播 setsockopt();  
(3)发送广播数据//向广播地址发送数据 sendto();  
(4)关闭套接字 close();  
   
接收端  
(1)创建一个数据报套接字SOCK\_DGRAM socket();  
(2)绑定广播IP地址和端口号 bind();  
(3)阻塞接收数据 recvfrom();  
(4)关闭套接字 close();

## 1. 接收端

(1) 创建UDP套接字   
   
 int sockfd = socket(AF\_INET,SOCK\_DGRAM,0);  
  
(2) 绑定广播地址和端口号  
   
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(8888);  
 myaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.255");  
 bind(sockfd,(struct sockaddr \*)&myaddr,sizeof(myaddr));  
  
(3) 接收数据  
   
 //不想知道送广播者的IP地址和端口号  
 recvfrom(sockfd,buf,sizeof(buf),0,NULL,NULL);   
  
 //想要知道发送广播者的IP地址和端口号  
 struct sockaddr\_in youaddr = { 0 }; //用来保存发送广播者的IP地址和端口号  
 int len = sizeof(youaddr);  
 recvfrom(sockfd,buf,sizeof(buf),0,(struct sockaddr \*)&youaddr,&len);  
  
(4) 关闭套接字  
 close(sockfd);

## 2. 发送端

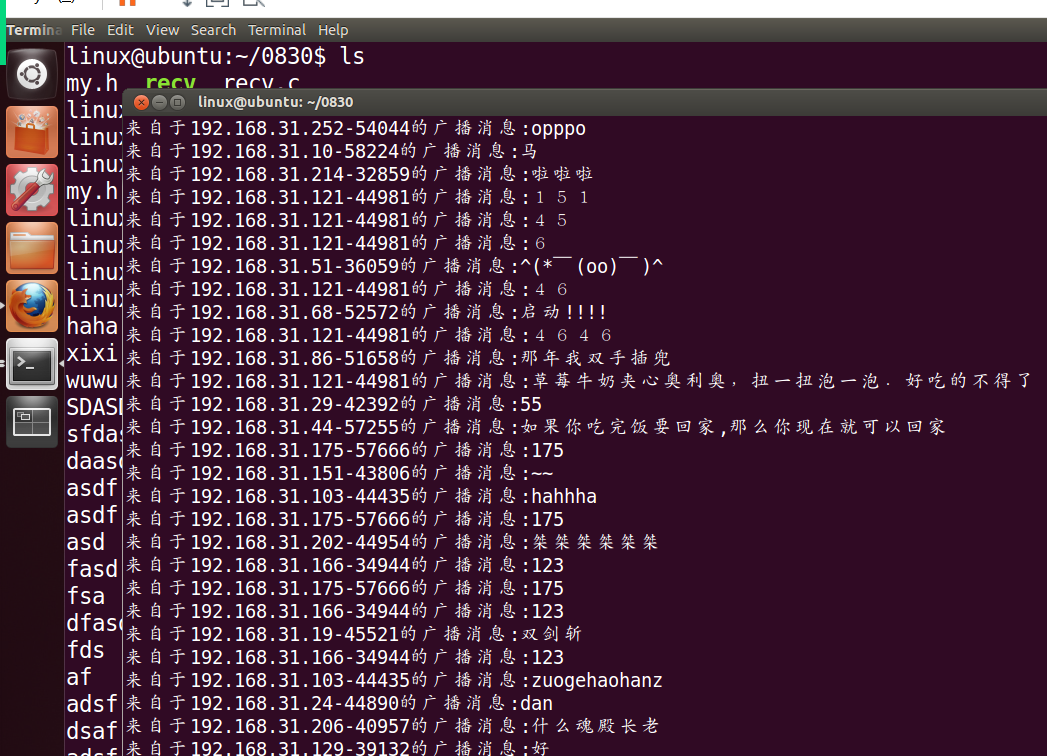
(1) 创建一个UDP数据报套接字   
   
 int sockfd = socket(AF\_INET,SOCK\_DGRAM,0);  
  
(2)设置套接字选项允许发送广播   
   
 int on = 1; //1代表允许发送广播，0代表不允许  
 setsockopt(sockfd,SOL\_SOCKET,SO\_BROADCAST,&on,sizeof(on));  
 //setsockopt   
 set //设置   
 sock ----socket //套接字   
 opt ---- option//选择  
   
(3)向广播地址发送数据  
   
 char buf[100] = "hello world";  
 struct sockaddr\_in toaddr = { 0 };  
 toaddr.sin\_family = AF\_INET;  
 toaddr.sin\_port = htons(8888);  
 toaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.255");  
 sendto(sockfd,buf,sizeof(buf),0,(struct sockaddr \*)&toaddr,sizeof(toaddr));  
   
(4)关闭套接字   
 close(sockfd)

### send.c

#include "my.h"  
  
int main(int argc, const char \*argv[])  
{  
 char buf[100] = { 0 };  
 //1.创建一个数据报套接字  
 int sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);  
 if(sockfd == -1)  
 {  
 perror("sockfd failed");  
 exit(-1);  
 }  
 //2.允许发送广播  
 int on = 1;//1允许发送广播  
 setsockopt(sockfd, SOL\_SOCKET, SO\_BROADCAST, &on, sizeof(on));  
  
 //3.向广播地址循环发送数据  
 struct sockaddr\_in toaddr = { 0 };  
 toaddr.sin\_family = AF\_INET;  
 toaddr.sin\_port = htons(8888);  
 toaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.255");  
 while(1)  
 {  
 scanf("%s", buf);  
 sendto(sockfd, buf, strlen(buf)+1, 0, (struct sockaddr\*)&toaddr, sizeof(toaddr));  
 }  
 //4.关闭套机字  
 close(sockfd);  
 return 0;  
}

### recv.c

#include "my.h"  
  
int sockfd;  
  
void fun(int num)  
{  
 //4.关闭套接字  
 close(sockfd);  
 //结束程序  
 exit(0);  
}  
  
  
int main(int argc, const char \*argv[])  
{  
 char buf[100] = { 0 };   
 //1.创建一个数据报套接字  
 sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed");  
 exit(-1);  
 }  
 //2.绑定广播的IP地址和端口号  
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(8888);  
 myaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.255");  
 int ret = bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myaddr));  
 if(ret == -1)  
 {  
 perror("bind failed");  
 exit(-1);  
 }  
  
 signal(SIGINT, fun);  
  
 //3.阻塞接收数据  
 struct sockaddr\_in youaddr = { 0 };  
 int len = sizeof(youaddr);  
 while(1)  
 {  
 recvfrom(sockfd, buf, sizeof(buf), 0, (struct sockaddr\*)&youaddr, &len);  
 printf("来自于%s-%d的广播消息:%s\n",inet\_ntoa(youaddr.sin\_addr), ntohs(youaddr.sin\_port), buf);  
 }  
  
 return 0;  
}



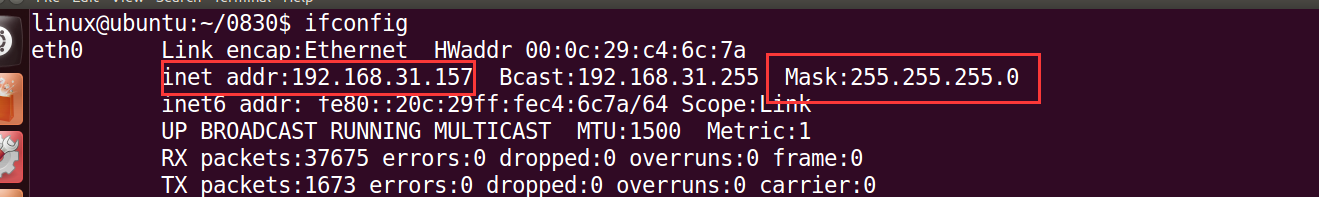
# 二. 组播

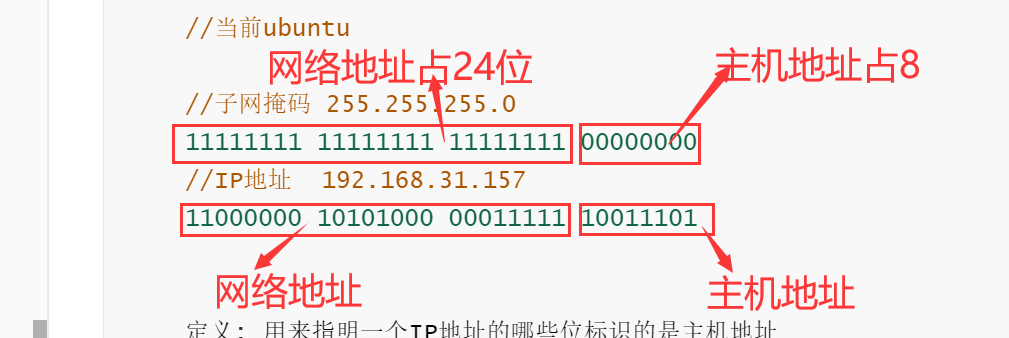
BroadCast //广播  
MultiCast //组播  
组播:可以给1组计算机发送数据  
只要有一台计算机加入到一个组里面，就可以给这个计算机发送数据

## 1. IP地址组成

(1)无符号整型IP地址占几个字节？？   
 unsigned int //4个字节  
 typedef unsigned int in\_addr\_t;  
   
(2)无符号的字符类型 unsigned char 存储的最大整数是多少？？  
   
 11111111 ----> 255  
   
(3)IP地址由网络地址和主机地址两部分组成,如何划分？？  
 划分网络地址和主机地址需要 子网掩码

## 2. 如何划分网络地址和主机地址

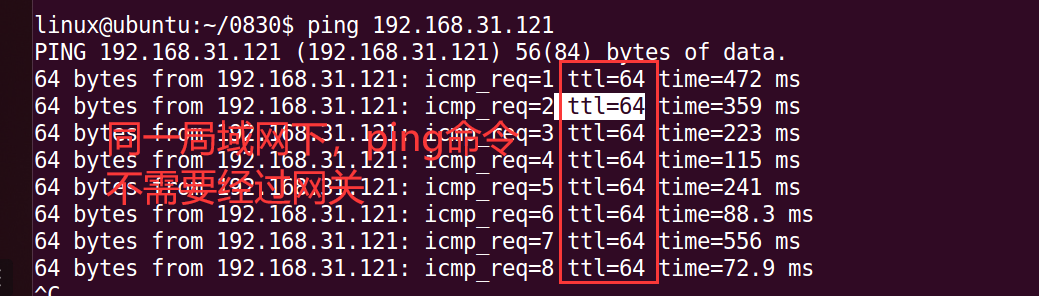


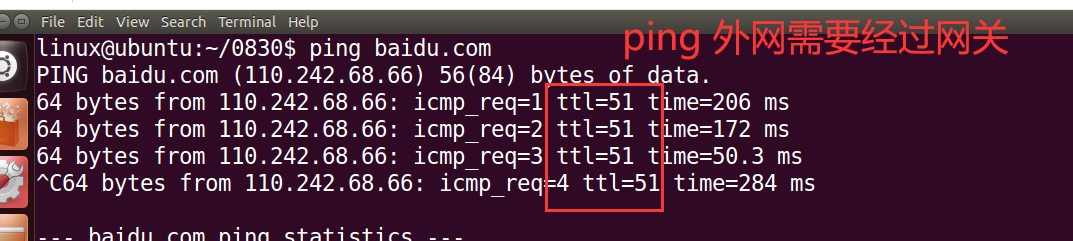


子网掩码:  
 //当前ubuntu  
   
 //子网掩码 255.255.255.0   
 11111111 11111111 11111111 00000000  
 //IP地址 192.168.31.157   
 11000000 10101000 00011111 10011101   
   
 网络地址：192.168.31.0 //网络地址代表的就是当前所处的网段  
 主机地址：157  
   
 定义: 用来指明一个IP地址的哪些位标识的是主机地址  
 作用: 用来将一个大网络分成几个子网  
 子网掩码：必须满足条件 左边全是1，右边全是0  
 子网掩码都是 左全是1 右全是0  
   
 1的部分是网络地址  
 0的部分是主机地址  
   
 (1)请问下面的是子网掩码吗？  
   
 255.255.255.0 //是  
 254.255.255.252 //不是  
 255.255.255.240 //是 网络地址占28位，主机地址占4位  
 //240 11110000   
 255.255.255.252 //是 网络地址占30位，主机地址占2位  
 //252 11111100   
   
 (2) 请问子网掩码255.255.255.0，网络地址占多少位，主机地址占多少位？  
   
 11111111 11111111 11111111 00000000  
 网络地址占24位，主机地址占8位  
  
 (3) 192.168.31.170在255.255.255.0子网掩码下，网络地址和主机地址分别是多少？？  
   
 网络地址 192.168.31.0  
 主机地址 170

## 3. 判断是否在同一网段

如何判断两个IP地址是否在同一个网段??  
   
(1)子网掩码 255.255.255.0  
 //子网掩码下，网络地址占24位，是192.168.1.0  
 192.168.1.1 //网络地址 192.168.1.0  
 192.168.1.19 //网络地址 192.168.1.0  
 请问在该子网掩码下，二者是否属于同一网段?? 属于同一网段  
   
(2)子网掩码 255.255.255.0  
   
 192.168.0.1 //网络地址 192.168.0.0  
 192.168.2.19 //网络地址 192.168.2.0   
 请问在该子网掩码下，二者是否属于同一网段?? 不属于同一网段  
   
(3)子网掩码 255.0.0.0  
   
 10.0.0.5 //网络地址 10.0.0.0  
 10.5.2.1 //网络地址 10.0.0.0  
 请问在该子网掩码下，二者是否属于同一网段?? 属于同一网段  
   
(4)子网掩码 255.255.0.0  
   
 192.168.1.1 //网络地址 192.168.0.0  
 192.168.0.2 //网络地址 192.168.0.0  
 请问在该子网掩码下，二者是否属于同一网段?? 属于同一网段  
   
 当设置完子网掩码之后，如果是和我同一个局域网的主机，通信(不通过网关)  
否则(需要通过网关)  
   
 网关: 当访问另一个网络中的主机时，就要通过网关(通常是路由器)





### 笔试题

0. 三个二进制位，一共有多少中不同的组合??  
 2^3 种组合  
 000 //0  
 001 //1  
 010 //2  
 011 //3  
 100 //4  
 101 //5  
 110 //6  
 111 //7  
   
1. 子网掩码如下，那么此网络中最多有多少台主机??  
   
 255.255.255.0,在此子网掩码下，能够分配多少台主机？？  
 //能够分配多少台主机，要看主机地址占多少位  
 //当前子网掩码下，主机地址占8位  
 //因为8位 共有 2^8 == 256 种组合  
 //表达范围 0 -- 255  
   
 分配的主机个数 = 2^8 - 2 == 254个  
 //-2 是因为 一个是 最小的地址 代表的该网段的地址，不能用  
 // 一个是 最大的地址 代表的广播地址，不能用  
   
   
 255.255.255.128,在此子网掩码下，能够分配多少台主机？？  
 //128 -->10000000  
 //主机地址占7位， 分得的主机个数就是2^7 - 2 = 126  
   
2. 子网掩码如下, 那么此网络中最多有多少台主机??  
 255.255.255.240  
 //240 -->11110000  
 //主机地址占4位， 分得的主机个数就是2^4 - 2 = 14  
   
3. 172.16.100.5 在此子网掩码下 255.255.255.252 网络地址和主机地址分别是多少??  
 255.255.255.252 //此子网掩码下最多能分配2台主机 2^2 - 2  
 76543210  
 //子网掩码 11111111 11111111 11111111 11111100  
 //地址地址 172 16 100 00000101  
 网络地址：172.16.100.4  
 主机地址：1

## 4. IP地址分类

属于哪一类地址，就看以谁为开头  
   
(1)A类地址,0开头 A类型 政府 大型公司  
   
 默认子网掩码 255.0.0.0  
 8位网络地址 24位主机地址   
 最高位是0, 7位有效网络地址   
 24位主机地址(2^24-2个主机)  
 1.0.0.1 --- 126.255.255.254  
 00000001 01111110  
   
(2)B类地址,10开头 B类 中小型公司   
 默认子网掩码 255.255.0.0  
 16位网络地址 16位主机地址   
 最高位是10, 14位有效网络地址   
 16位主机地址(2^16-2 个主机)  
   
 10000000 - 10111111  
 128.0.0.1 - 191.255.255.254   
   
(3)C类地址,110开头 //个人使用C类  
   
 默认子网掩码 255.255.255.0  
 24位网络地址 8位主机地址  
 最高位是110, 21位有效网络地址   
 8位主机地址(2^8-2 个主机)  
 192.0.0.1 --- 223.255.255.254  
 11000000 --- 11011111  
   
 172.2.3.5  
(4)D 类地址 最高位为 1110 专门用来做(多播)组播  
   
 11100000 --- 11101111  
 224.0.0.1 --- 239.255.255.254   
  
   
 127开头的ip地址  
 127.0.0.0到127.255.255.255是保留地址，用做循环测试用的。(自发自收 127.0.0.1)  
   
 总结：判断一个IP地址是哪一类就看 是以谁开头的  
  
 0开头 A类 1.0.0.1 -- 126.255.255.254  
 10开头 B类 128.0.0.1 --- 191.255.255.254  
 110开头 C类 192.0.0.1 --- 223.255.255.254  
 1110开头 D类 组播 224.0.0.1 --- 239.255.255.254

## 5. 组播通信

### 5.0.组播的发送端和接收端

//组播的发送端和接收端  
发送端  
(1)创建一个数据报套接字SOCK\_DGRAM socket();   
(2)发送组播数据//向组播地址发送数据 sendto();  
(3)关闭套接字 close();  
接收端  
(1)创建一个数据报套接字SOCK\_DGRAM socket();  
(2)绑定组播IP地址和端口号 bind();  
(3)将自己的本机的IP地址和组播地址关联在一起 setsockopt();  
(4)阻塞接收数据  
(5)关闭套接字

### 5.1 接收端

(1)创建一个UDP套接字  
 int sockfd = socket(AF\_INET,SOCK\_DGRAM,0);  
  
(2)将本机地址和组播地址关联在一起  
   
 /\*  
 struct ip\_mreq  
 {  
 struct in\_addr imr\_multiaddr; //保存播组的IP地址  
 struct in\_addr imr\_interface; //保存本机的IP地址  
 }  
 \*/  
   
 struct ip\_mreq mreq = { 0 } //里面用来保存组播地址和本机地址  
 mreq.imr\_multiaddr.s\_addr = inet\_addr("224.10.10.10"); //组播地址  
 mreq.imr\_interface.s\_addr = htonl(INADDR\_ANY);//本机地址  
 //mreq.imr\_interface.s\_addr = inet\_addr("192.168.31.170");//本机地址  
 setsockopt(sockfd,IPPROTO\_IP,IP\_ADD\_MEMBERSHIP,&mreq,sizeof(mreq));  
 //执行此函数后，才将本机地址与组播地址关联在一起  
  
(3)绑定组播地址及端口号  
   
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(6666);  
 myaddr.sin\_addr.s\_addr = inet\_addr("224.10.10.10");  
 bind(sockfd,(struct sockaddr \*)&myaddr,sizeof(myaddr));  
  
(4)接收数据   
   
 struct sockaddr\_in youaddr = { 0 }; //用来保存发送方的IP地址和端口号  
 int len = sizeof(youaddr);  
 recvfrom(sockfd,buf,sizeof(buf),0,(struct sockaddr \*)&youaddr,&len);  
   
(5) 关闭套接字   
 close(sockfd);

### 5.2 发送端

(1)创建一个UDP套接字   
   
 int sockfd = socket(AF\_INET,SOCK\_DGRAM,0);  
   
(2)向指定的组播地址发送数据   
   
 struct sockaddr\_in toaddr = { 0 }; //用来保存指定的组播地址  
 char buf[100] = "hello";  
 toaddr.sin\_family = AF\_INET;  
 toaddr.sin\_port = htons(6666);  
 toaddr.sin\_addr.s\_addr = inet\_addr("224.10.10.10"); //注意此处为组播地址  
   
sendto(sockfd,buf,sizeof(buf),0,(struct sockaddr \*)&toaddr,sizeof(toaddr));  
   
(3)关闭套接字   
   
 close(sockfd);

#### recv.c

#include "my.h"  
  
int main(int argc, const char \*argv[])  
{  
 char buf[100] = { 0 };   
 //1.创建一个数据报套接字  
 int sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed");  
 exit(-1);  
 }  
  
 //2.绑定组播的地址和端口号  
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(6666);  
 myaddr.sin\_addr.s\_addr = inet\_addr("224.10.10.10");  
 int ret = bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myaddr));  
 if(ret == -1)  
 {  
 perror("bind failed");  
 exit(-1);  
 }  
 //3.将组播地址和本机地址关联在一起  
 struct ip\_mreq mreq = { 0 };  
 mreq.imr\_multiaddr.s\_addr = inet\_addr("224.10.10.10");  
 mreq.imr\_interface.s\_addr = htonl(INADDR\_ANY);  
 setsockopt(sockfd, IPPROTO\_IP, IP\_ADD\_MEMBERSHIP, &mreq, sizeof(mreq));  
  
 //4.阻塞接收数据  
 struct sockaddr\_in youaddr = { 0 };  
 int len = sizeof(youaddr);  
 while(1)  
 {  
 recvfrom(sockfd, buf, sizeof(buf), 0, (struct sockaddr\*)&youaddr, &len);  
 printf("来自于%s-%d的组播消息:%s\n",inet\_ntoa(youaddr.sin\_addr), ntohs(youaddr.sin\_port), buf);  
 fflush(stdout);  
 }  
 //5.关闭套接字  
 close(sockfd);  
 return 0;  
}

#### send.c

#include "my.h"  
  
int main(int argc, const char \*argv[])  
{  
 char buf[100] = { 0 };  
 //1.创建一个数据报套接字  
 int sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);  
 if(sockfd == -1)  
 {  
 perror("sockfd failed");  
 exit(-1);  
 }  
 //2.向组播地址循环发送数据  
 struct sockaddr\_in toaddr = { 0 };  
 toaddr.sin\_family = AF\_INET;  
 toaddr.sin\_port = htons(6666);  
 toaddr.sin\_addr.s\_addr = inet\_addr("224.10.10.10");  
 while(1)  
 {  
 scanf("%s", buf);  
 sendto(sockfd, buf, strlen(buf)+1, 0, (struct sockaddr\*)&toaddr, sizeof(toaddr));  
 }  
 //3.关闭套机字  
 close(sockfd);  
 return 0;  
}

# @作业：

继续完成day02 文件下载作业

# server.c

#include "my.h"  
  
  
typedef struct   
{  
 int type; //1 列表 2 下载 3 上传 4 退出  
 char filename[50]; //保存文件名  
 char filedata[100]; //保存文件内容  
 int len; //包数据长度，用来保存每次实际读取到的字节数的长度  
}MSG;  
  
  
//处理客户端获取列表请求的函数  
void doFileListRequest(int newsockfd)  
{  
 MSG s = { 0 };  
 struct dirent\* ep = NULL;  
 //打开共享文件夹  
 DIR\* dp = opendir("/home/linux/0216/");  
 if(dp == NULL)  
 {  
 perror("opendir failed");  
 exit(-1);  
 }  
 //读取出所有文件的名字  
 //循环发送给客户端  
 while((ep=readdir(dp)) != NULL)  
 {  
 if(ep->d\_name[0] == '.')  
 continue;  
 s.type = 1;  
 strcpy(s.filename, ep->d\_name);//将读取的文件名,撞到结构体中  
 send(newsockfd, &s, sizeof(s), 0);  
 }  
 //循环结束后,说明文件名字已经全部发送完毕  
 //额外在多发送一次,告诉客户端全部发送完毕  
 s.type = 0;//用0表达全部发送完成  
 send(newsockfd, &s, sizeof(s), 0);  
   
  
 closedir(dp);  
  
}  
  
int main(int argc, const char \*argv[])  
{  
 char buf[100] = { 0 };//用来保存接收到的数据  
 //1.创建一个流式套接字  
 int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed!!");  
 exit(-1);  
 }  
 printf("sockfd is %d\n",sockfd);  
 //2.绑定自己本机的IP地址和端口号  
 struct sockaddr\_in myaddr = { 0 };  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(6666);  
// myaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.179");  
 myaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);  
 int ret = bind(sockfd, (struct sockaddr\*)&myaddr, sizeof(myaddr));  
 if(ret == -1)  
 {  
 perror("bind failed");  
 exit(-1);  
 }  
 printf("bind ok!!\n");  
 //3.设置监听 允许同时最大的连接个数  
 listen(sockfd, 5);  
  
 int newsockfd = accept(sockfd, NULL, NULL);  
 printf("newsockfd is %d\n",newsockfd);  
 MSG s = { 0 };//用来保存接收到的数据  
 while(1)  
 {  
 int ret = recv(newsockfd, &s, sizeof(MSG), 0);  
 if(ret > 0)//recv解除阻塞,是因为接收到的数据  
 {  
 printf("type:%d\n",s.type);//调试程序  
 //服务器接收到数据之后,要判断客户端请求的类型  
 switch(s.type)  
 {  
 case 1://列表请求  
 doFileListRequest(newsockfd);  
 break;  
 case 2://下载请求  
 break;  
 case 3://上传请求  
 break;  
  
 }  
 }  
 else //客户端断开连接  
 {  
 close(newsockfd);  
 }  
 }  
  
  
 //6.关闭套接字  
 close(sockfd);  
 return 0;  
}

# client.c

#include "my.h"  
  
typedef struct   
{  
 int type; //1 列表 2 下载 3 上传 4 退出  
 char filename[50]; //保存文件名  
 char filedata[100]; //保存文件内容  
 int len; //包数据长度，用来保存每次实际读取到的字节数的长度  
}MSG;  
  
//给服务器发送获取列表请求  
void getFileList(int sockfd)  
{  
 MSG s = { 0 };//发送给服务器数据的结构体变量  
 //将客户端的获取列表请求发送给服务器  
 //先给结构体赋值,装上请求类型和数据然后发送给服务器  
 s.type = 1;//1代表获列表请求  
 send(sockfd, &s, sizeof(MSG), 0);//发送给服务器  
 //接收客户端的应答  
 while(1)  
 {  
 recv(sockfd, &s, sizeof(MSG), 0);  
 if(s.type == 0)//说明已经全部接收完毕  
 break;  
 printf("%s\n", s.filename);  
 }  
  
}  
  
  
int main(int argc, const char \*argv[])  
{  
 int n;  
 char buf[100] = { 0 };  
 //1.创建一个式套接字  
 int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);  
 if(sockfd == -1)  
 {  
 perror("socket failed");  
 exit(-1);  
 }  
 //2.连接服务器  
 //(1)在连接服务器之前,客户端已经提前知道了服务器的IP地址和端口号  
 //toaddr里面赋值的是哪个服务器的IP地址和端口号,连接的就是哪个服务器  
 struct sockaddr\_in toaddr = { 0 };  
 toaddr.sin\_family = AF\_INET;  
 toaddr.sin\_port = htons(6666);  
 toaddr.sin\_addr.s\_addr = inet\_addr("192.168.31.157");  
 int ret = connect(sockfd, (struct sockaddr\*)&toaddr, sizeof(toaddr));  
 if(ret == -1)  
 {  
 perror("connect failed");  
 exit(-1);  
 }  
 while(1)  
 {  
 printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  
 printf(" 1.列表 2.下载 3.上传 4.退出\n");  
 printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  
 scanf("%d",&n);  
 switch(n)  
 {  
 case 1:  
 getFileList(sockfd);  
 break;  
 case 2:  
  
 break;  
 case 3:  
  
 break;  
 case 4:  
 break;  
 }  
 }   
  
 //4.关闭套接字  
 close(sockfd);  
 return 0;  
}

# @复习:

1. 广播 BroadCast  
 广播地址：当前网段最大的IP地址是广播地址  
   
 广播接收端  
 (1)创建一个数据报套接字SOCK\_DGRAM socket();  
 (2)绑定广播地址和端口号bind(); //192.168.31.255  
 (3)阻塞接收数据 recvfrom();  
 (4)关闭套接字 close();  
   
 广播发送端  
 (1)创建一个数据报套接字SOCK\_DGRAM socket();  
 (2)允许发送广播  
 int on = 1;//允许发送广播  
 setsockopt(sockfd, SOL\_SOCKET, SO\_BROADCAST, &on, sizeof(on));  
 (3)向广播地址发送数据 sentdto();//192.168.31.255  
 (4)关闭套接字  
  
2. 组播 MultiCast  
   
 组播接收端  
 (1)创建一个数据报套接字SOCK\_DGRAM socket();  
 (2)绑定组播地址和端口号bind(); //224.10.10.10  
 (3)将本机地址和组播地址关联在一起  
 struct ip\_mreq mreq = { 0 };  
 mreq.imr\_multiaddr.s\_addr = inet\_addr("224.10.10.10");  
 mreq.imr\_interface.s\_addr = htonl(INADDR\_ANY);  
 setsockopt(sockfd, IPPROTO\_IP, IP\_ADD\_MEMBERSHIP, &mreq, sizeof(mreq));  
 (4)阻塞接收数据 recvfrom();  
 (5)关闭套接字 close();   
   
   
 组播发送端  
 (1)创建一个数据报套接字SOCK\_DGRAM socket();  
 (2)向组播地址发送数据 sentdto();//224.10.10.10  
 (3)关闭套接字   
   
 子网掩码：用来划分子网段  
 子网掩码：左边全是1，右边全是0  
 子网掩码可以用来划分 网络地址和主机地址   
 1的位数，就是网络地址， 0的位数，就是主机地址  
 255.255.255.224 //子网掩码  
 224// 11100000  
 当前子网掩码下可以分配多少个主机地址??   
 2^5 - 2   
   
 IP地址分类  
 A类： 0开头 1.0.0.1 --- 126.255.255.254  
 B类： 10开头 128.0.0.1 ---- 191.255.255.254  
 C类 110开头 192.0.0.1 --- 223.255.255.254  
 D类 1110开头 224.0.0.1 --- 239.255.255.254 //D类用来做组播地址

# --------------------------------

# Day04

# 一. TCP/IP 3卷

卷1 TCP/IP协议的原理 (tcp udp ip arp ....) //理解卷1  
卷2 TCP/IP协议的实现

# 二. wireshark软件

WireShark是网络封包分析工具，可以截取各种网络数据包，并显示数据包详细信息，用于开发和测试中的问题定位

## 1. WireShark使用示例

### 1.1 抓取ping命令过程

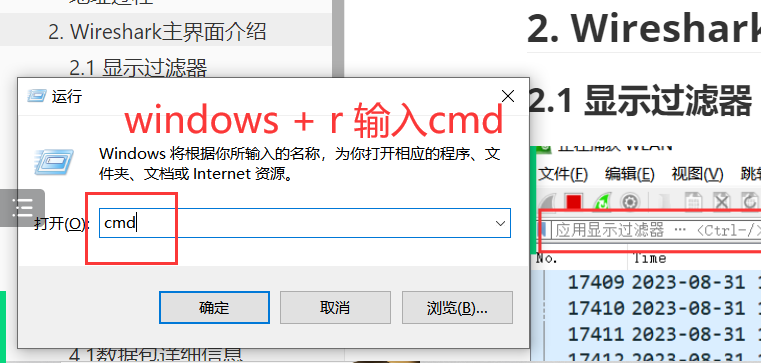


image-20230831102724972

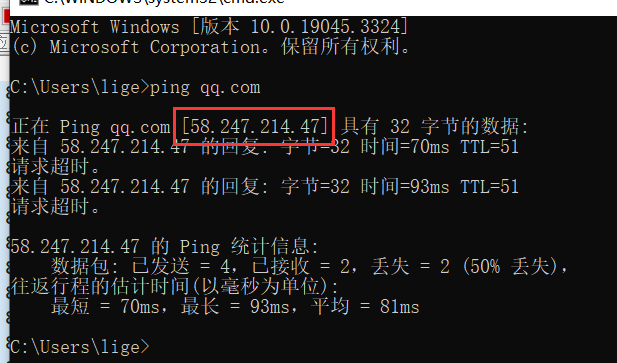
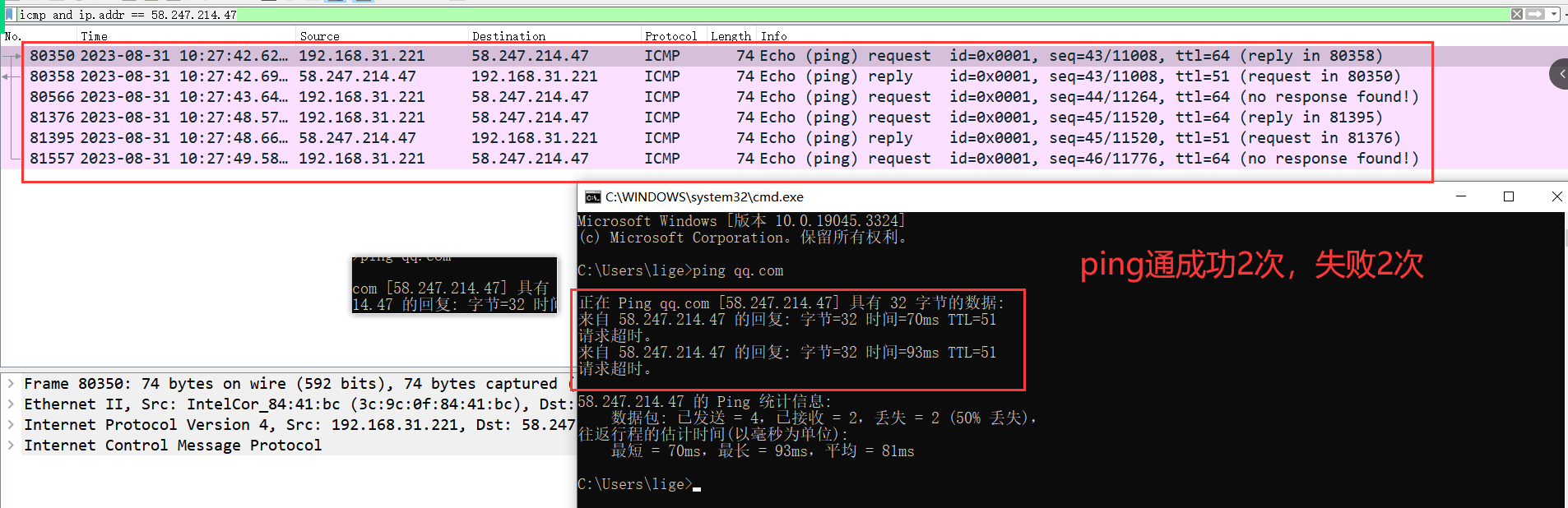
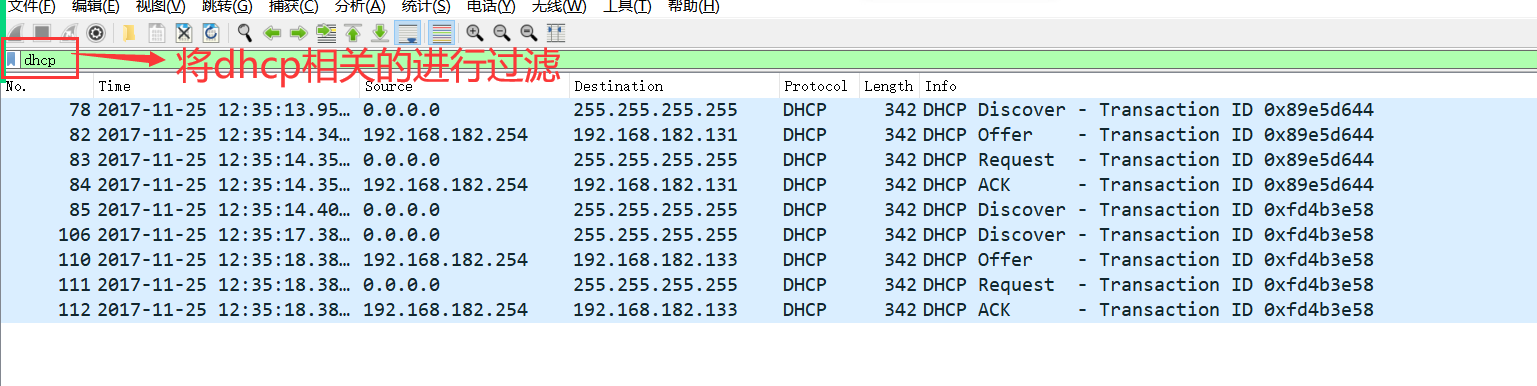


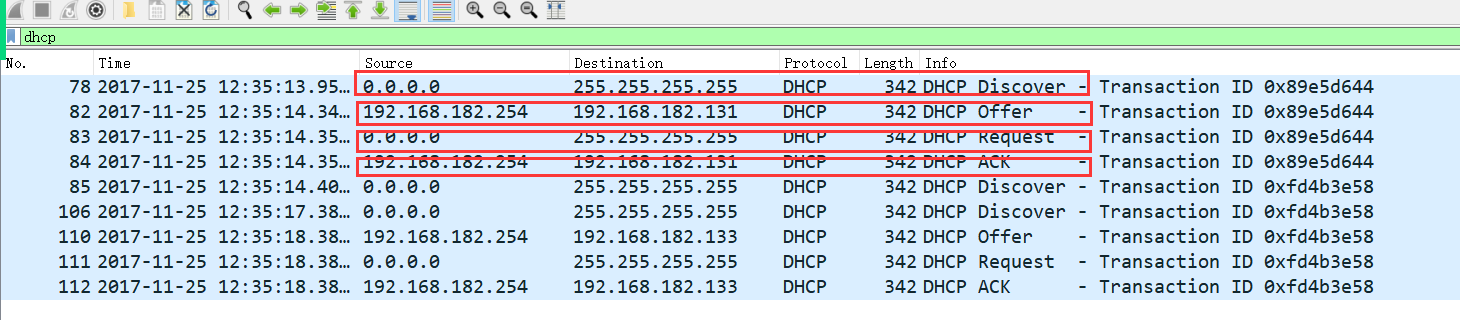
image-20230831102835992

在显示过滤栏，输入  
 icmp and ip.addr == 58.247.214.47 //过滤条件  
 //只过滤查看 使用icmp协议 ip地址是 58.247.214.47的数据包



### 1.2 分析dhcp动态分配IP地址过程





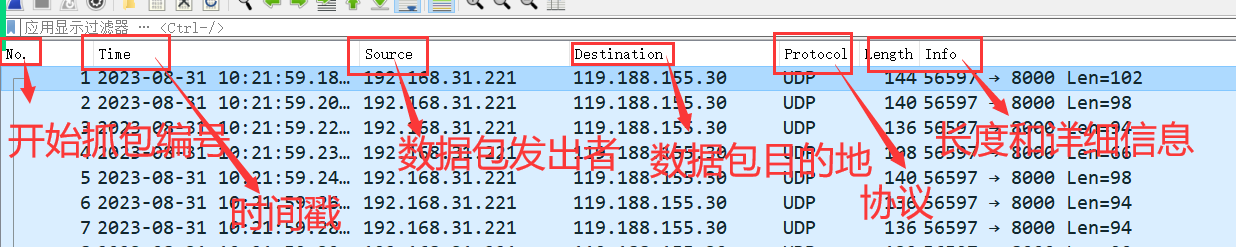
DHCP分配IP地址过程  
 (1)客户端发送一个 广播 DHCP Discover报文，问谁是DHCP服务器  
 (2)多个DHCP服务器收到后，会给客户端回应一个 DHCP Offer报文，并且包含了分配IP地址  
 (3)客户端发送一个 广播 DHCP Request报文，告诉多个DHCP服务器，客户端选择了谁分配的IP地址  
 (4)被选择的那个DHCP服务器给 客户端回应一个 DHCP ACK报文， 确定IP地址租约时间

## 2. Wireshark主界面介绍

### 2.1 显示过滤器



### 2.2 数据包列表



## 3. WireShark过滤器规则

### 3.1 协议过滤

直接输入协议进行过滤  
tcp，只显示TCP协议的数据包列表  
http，只查看HTTP协议的数据包列表  
icmp，只显示ICMP协议的数据包列表  
oicq, 只显示oicq协议的数据包列表

### 3.2 ip过滤

ip.src ==192.168.1.108 显示源地址为192.168.1.108的数据包列表  
ip.dst==192.168.1.108, 显示目标地址为192.168.1.108的数据包列表  
ip.addr == 192.168.1.108 显示源IP地址或目标IP地址为192.168.1.108的数据包列表

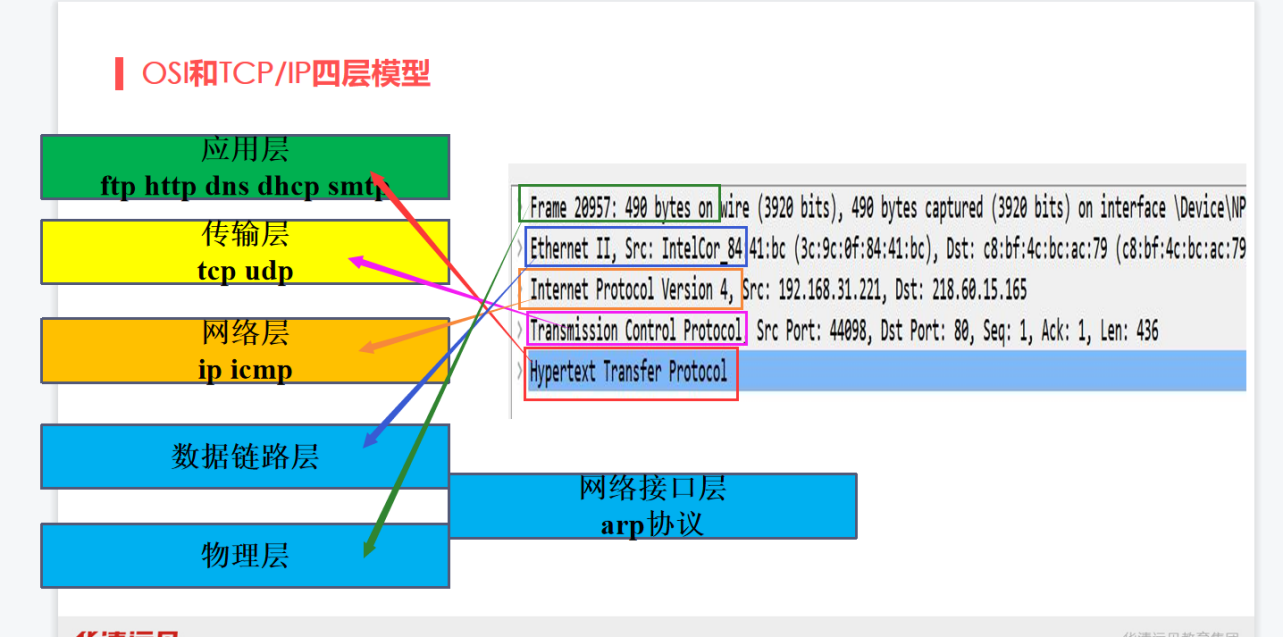
### 3.3 端口过滤

tcp.port == 80, 显示源主机或者目的主机端口为80的数据包列表。  
tcp.srcport == 80, 只显示TCP协议的源主机端口为80的数据包列表。  
tcp.dstport == 80，只显示TCP协议的目的主机端口为80的数据包列表。  
//也可以将上面的tcp改成udp

### 3.4 逻辑运算符

&& and   
|| or   
ip.addr == 192.168.1.104 and icmp

## 4. 分析数据包

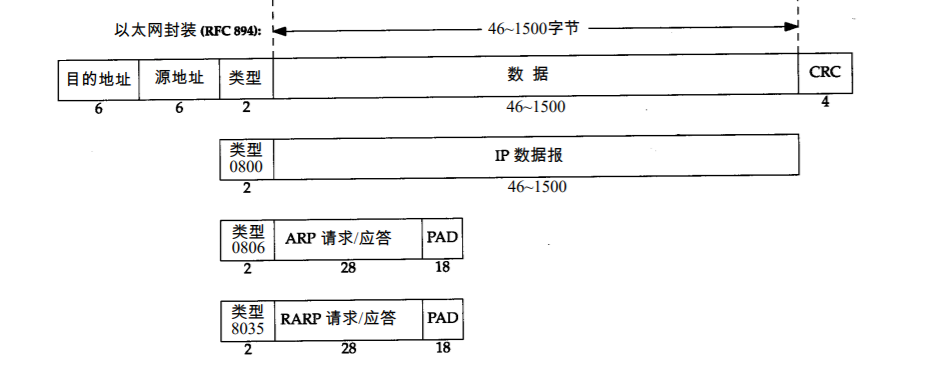


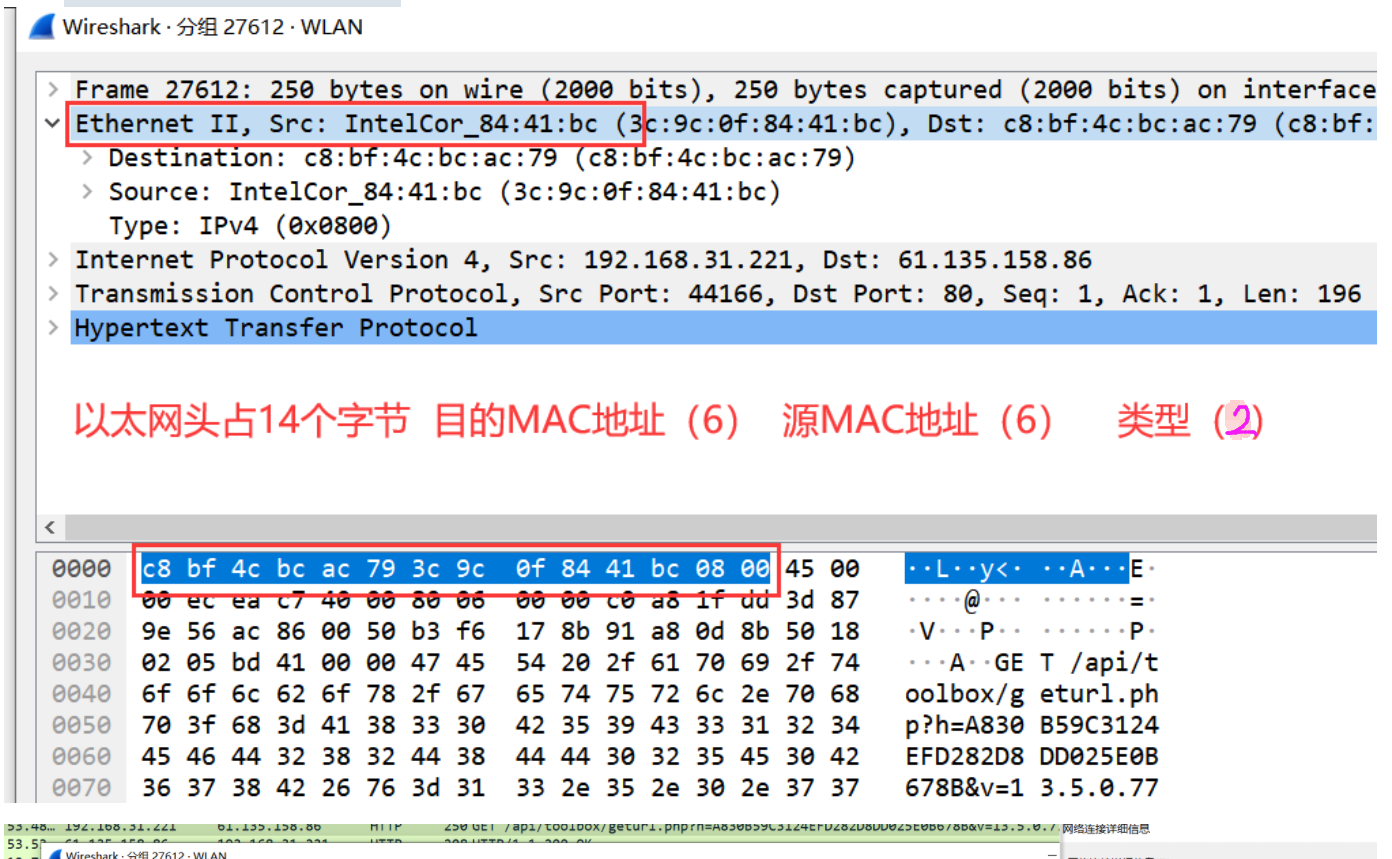
### 4.1数据包详细信息

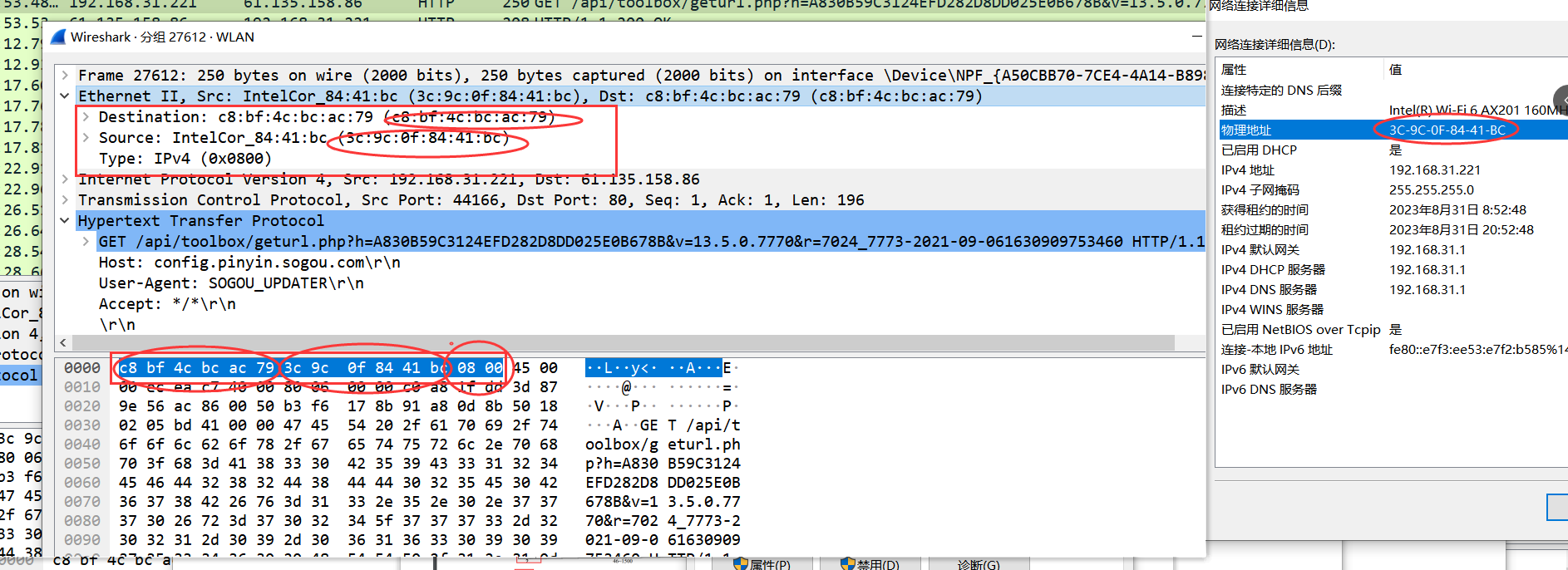
(1)Frame: 物理层的数据帧概况  
(2)Ethernet II: 数据链路层以太网帧头部信息  
(3)Internet Protocol Version 4: 互联网层IP包头部信息  
(4)Transmission Control Protocol: 传输层T的数据段头部信息，此处是TCP  
(5)Hypertext Transfer Protocol: 应用层的信息，此处是HTTP协议

### 4.2 分析以太网包头

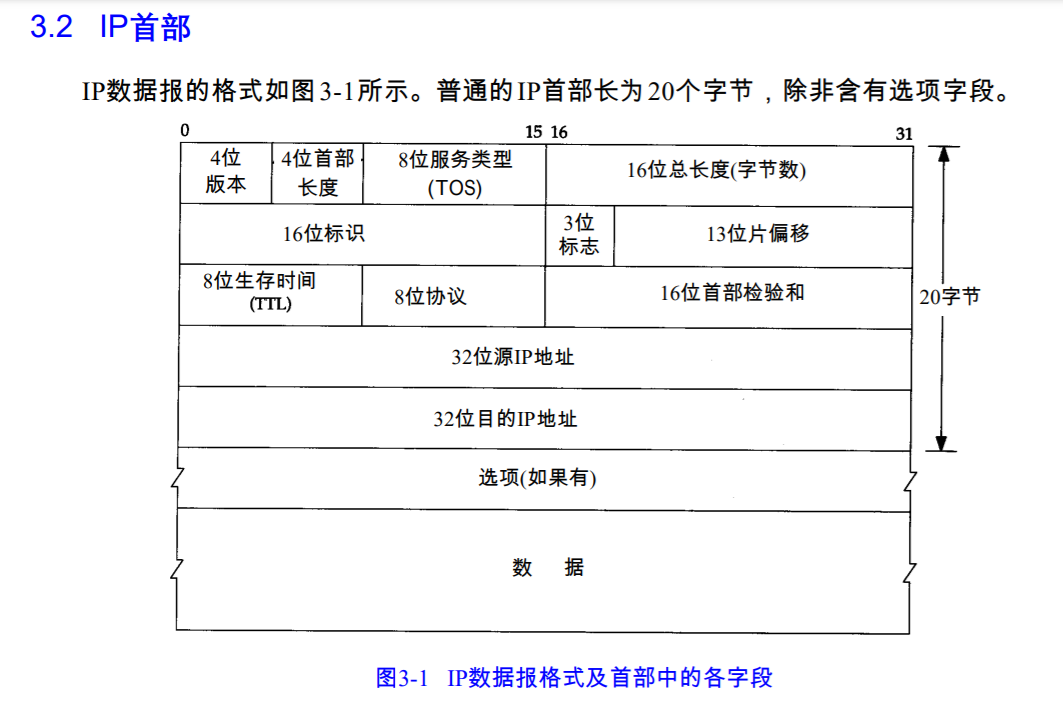
第2章 链 路 层  
目的 MAC地址  
源 MAC地址







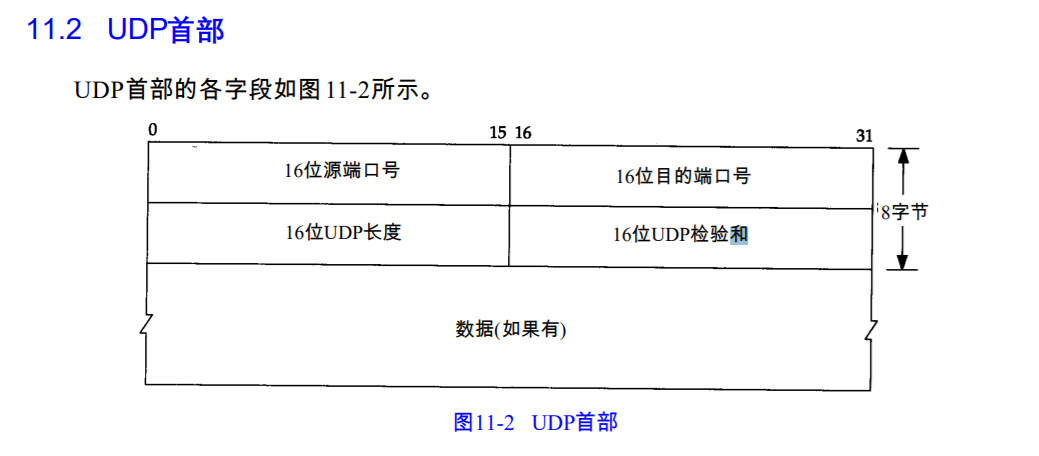
### 4.3 分析IP报文



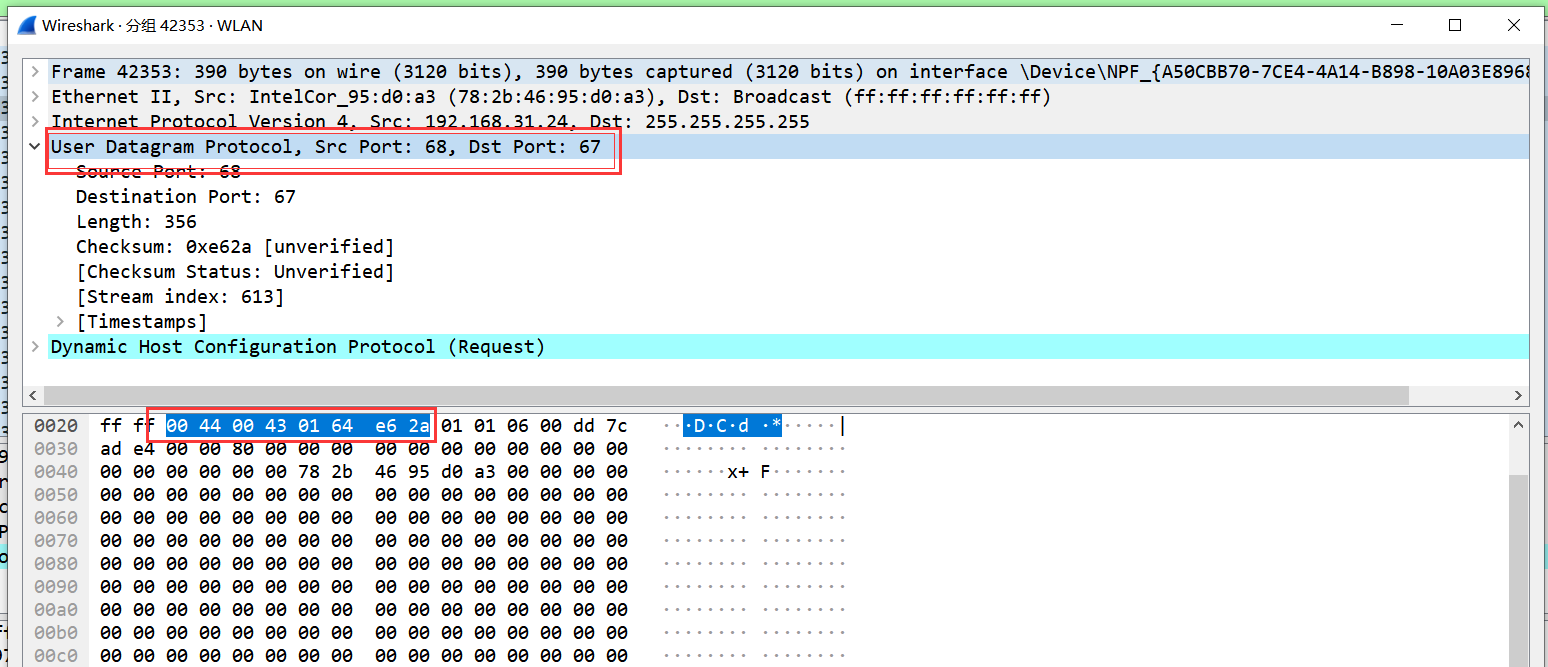
IP头 占 20 字节  
IP头 源IP地址  
 目的IP地址  
   
   
第3章IP：网际协议.PDF  
以太网最大的传输单元称为MTU 1500字节  
IP头: 源IP地址 目的IP地址  
IP头 占 20 字节  
版本字段：占4位。表示IP协议的版本。通信双方使用的IP协议的版本必须一致，目前广泛使用的IP协议版本号为4(即IPv4)  
首部长度字段：占4位。表示IP数据报首部的长度。该字段的取值以4字节为单位。范围(5-10)。  
区分服务：占8位。一般不使用  
总长度：占16位。IP数据报的总长度(首部+数据载荷)。以字节为单位。最大取值为65535(2^16)。  
 首部长度=0101\*4=20字节  
 假设总长度=0000001111111100=1020字节  
 数据载荷长度=总长度-首部长度=1020-20=1000字节  
标识：占16个位。属于同一个数据报的个分片数据报属于具有相同的标识。  
标志：占3位，个比特含义如下：  
■ DF位：1表示不允许分片；0表示允许分片。  
■ MF位：1表示“后面还有分片”，0表示“这是最后一个分片”  
■ 保留位：必须为0；  
片偏移：占13位，以8字节为单位。指分片数据报的数据载荷部分偏移其首地址数据报位置有多少个单位(8字节为1个单位)；  
 总长度 标识 MF DF 片偏移  
 原始数据报 3800+20 12345 0 0 0  
   
 分片1数据报 1400+20 12345 1 0 0/8  
 分片2数据报 1400+20 12345 1 0 1400/8  
 分片3数据报 1400+20 12345 0 0 2800/8



### 4.4 UDP报文



UDP头8个字节  
源端口号  
目的端口号



### 4.5 TCP报文

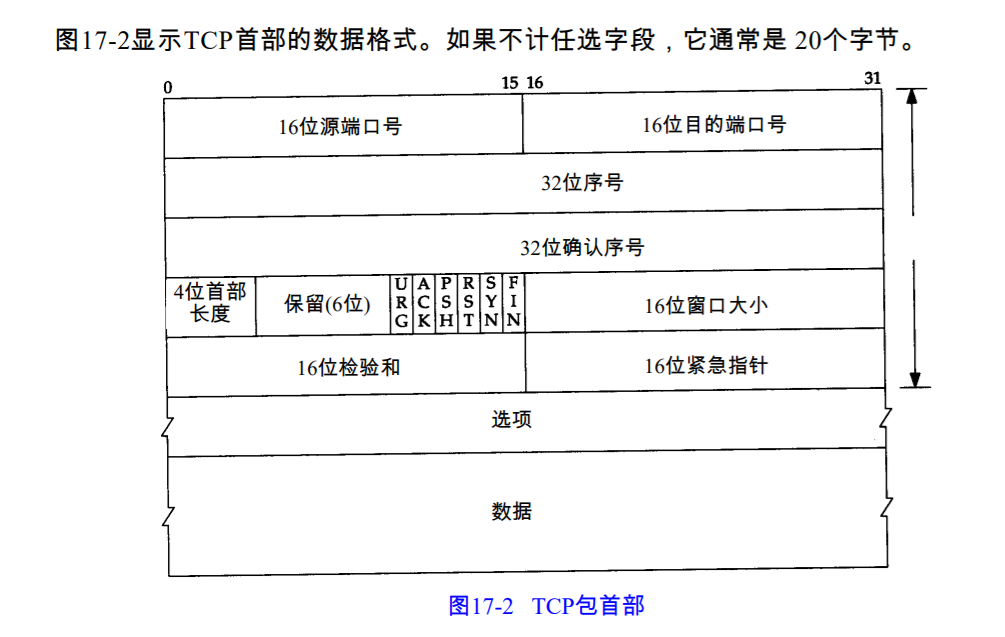


image-20230831145628374

## 5. 三次握手(重点)

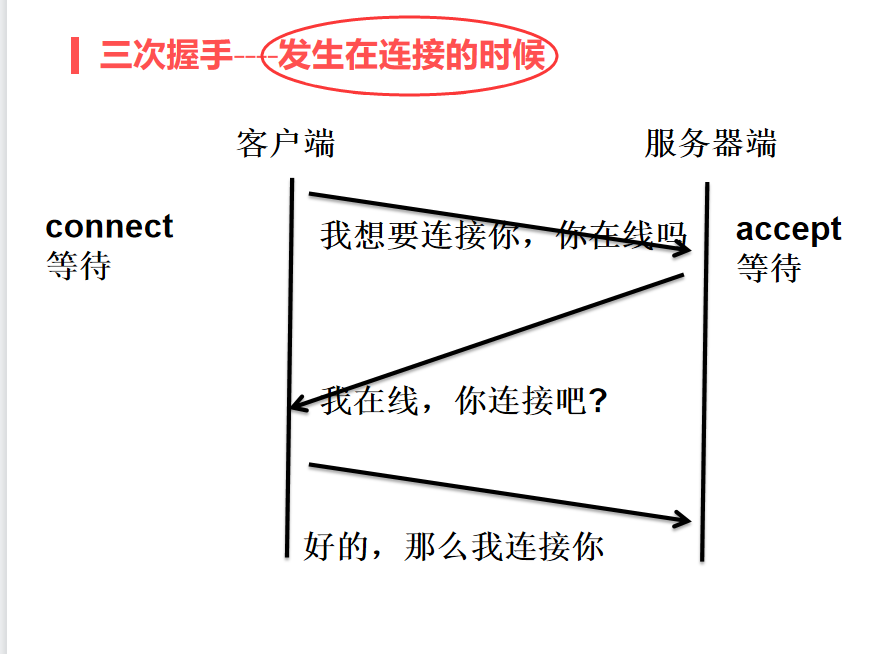


image-20230831150700143

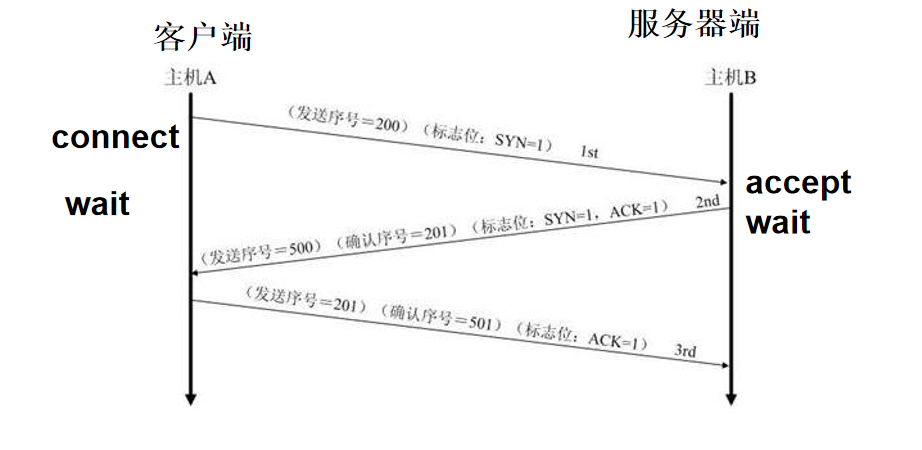
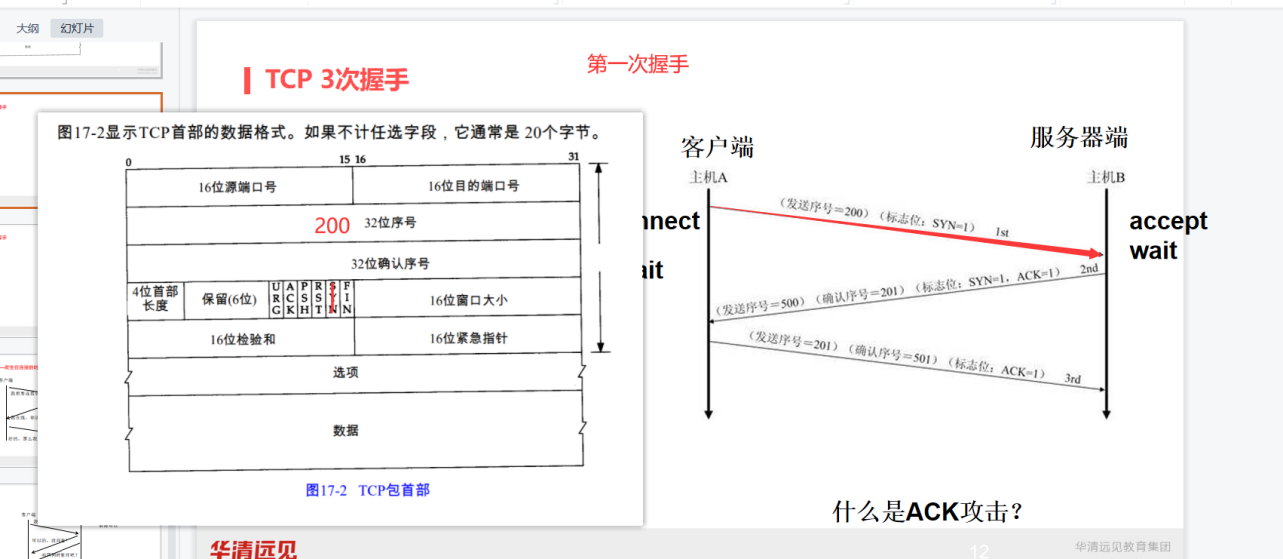


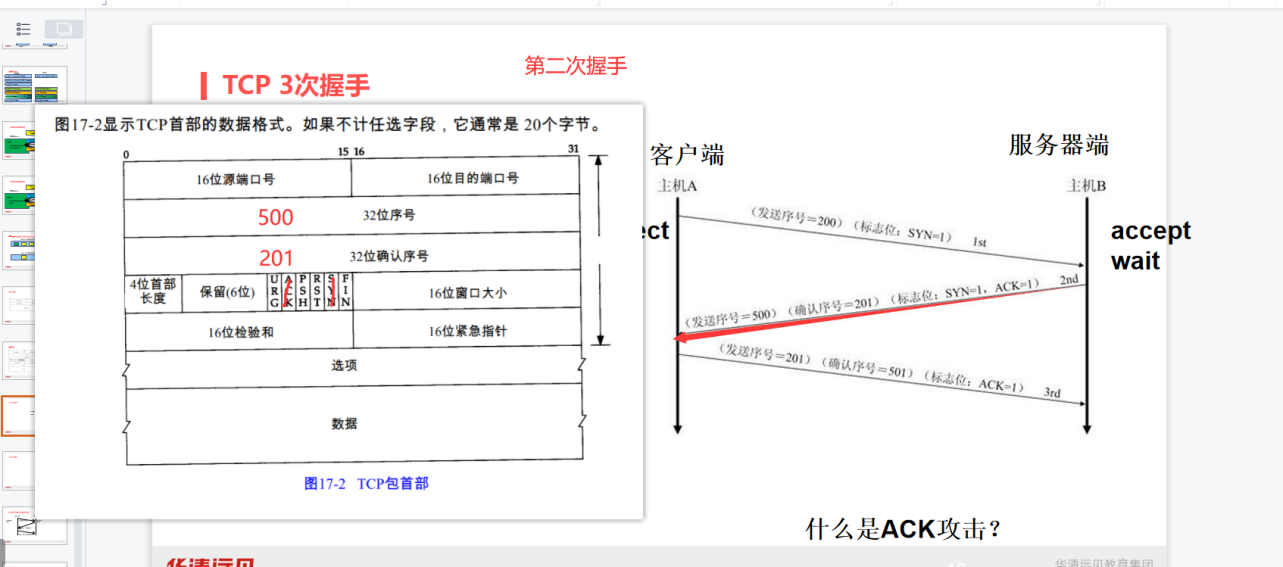
image-20230831150844546

1. 三次握手发生在什么时候??  
   
 三次握手发生客户端与服务器建立连接的过程  
 使用标志位有哪些？  
 SYN //建立连接  
 ACK //确认  
   
2. 三次握手时候，使用的SYN位 和 ACK位， 详细过程  
   
 第一次握手：  
 //客户端对服务器说：服务器你在吗？我想连接你  
 客户端发送给服务器， 发送序号假设是200(SYN j)，标志位SYN=1，代表请求连接  
 第二次握手：  
 //服务器对客户端说：我在，你连接吗？  
 服务器发送给客户端， 发送序号假设是500(SYN k)，标志位SYN=1 ACK=1， 确认序号是201(ACK j+1)   
   
 第三次握手：  
 //客户端对服务器说：好的，我连接你  
 客户端发送给服务器， 发送序号假设是201，标志位 ACK=1 确认序号是501(ACK k+1)  
   
3. 请问什么是ACK攻击??  
 ACK攻击发送在第3次握手的时候，握了2次手之后，不进行第3次握手，服务器处于短暂等待，浪费资源

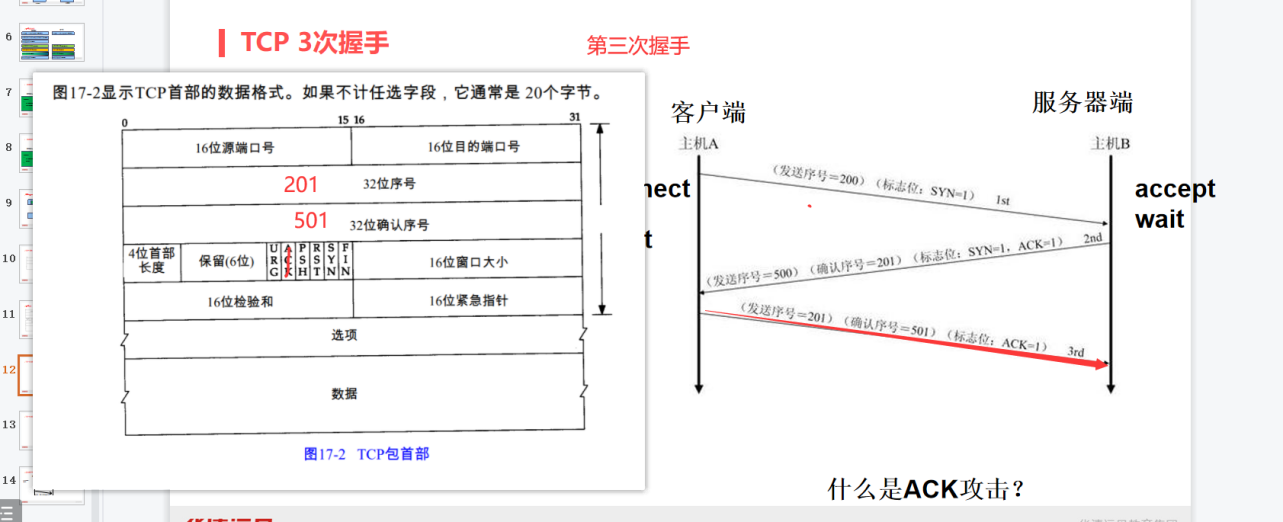
### 5.1 第一次握手



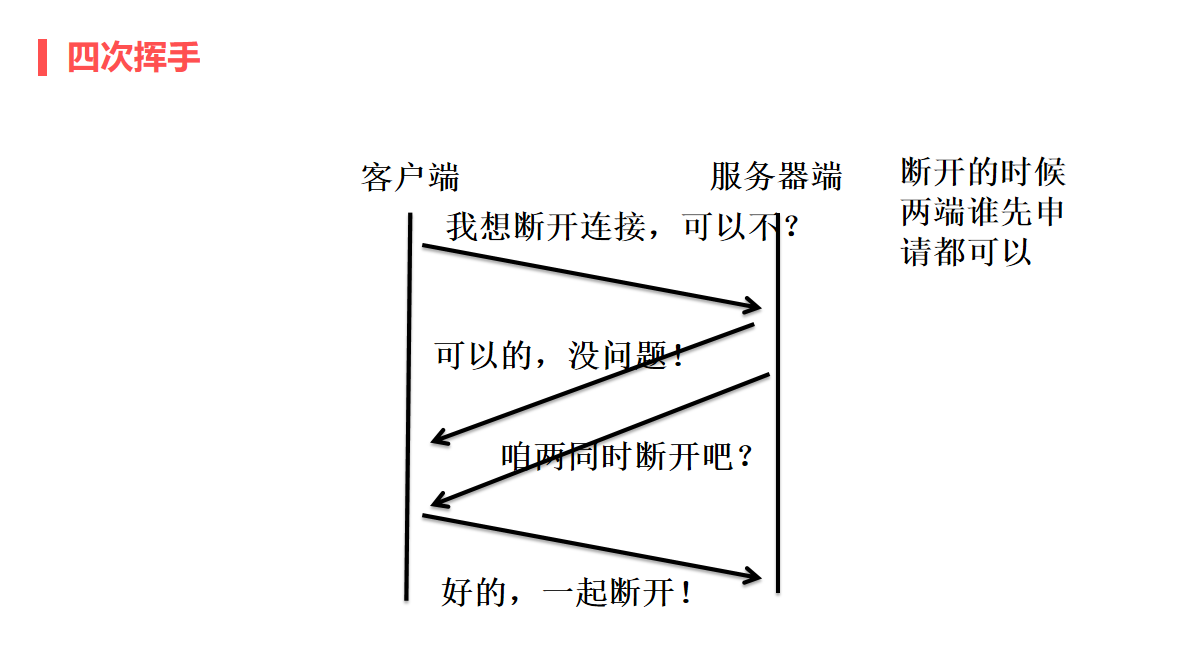
### 5.2 第二次握手

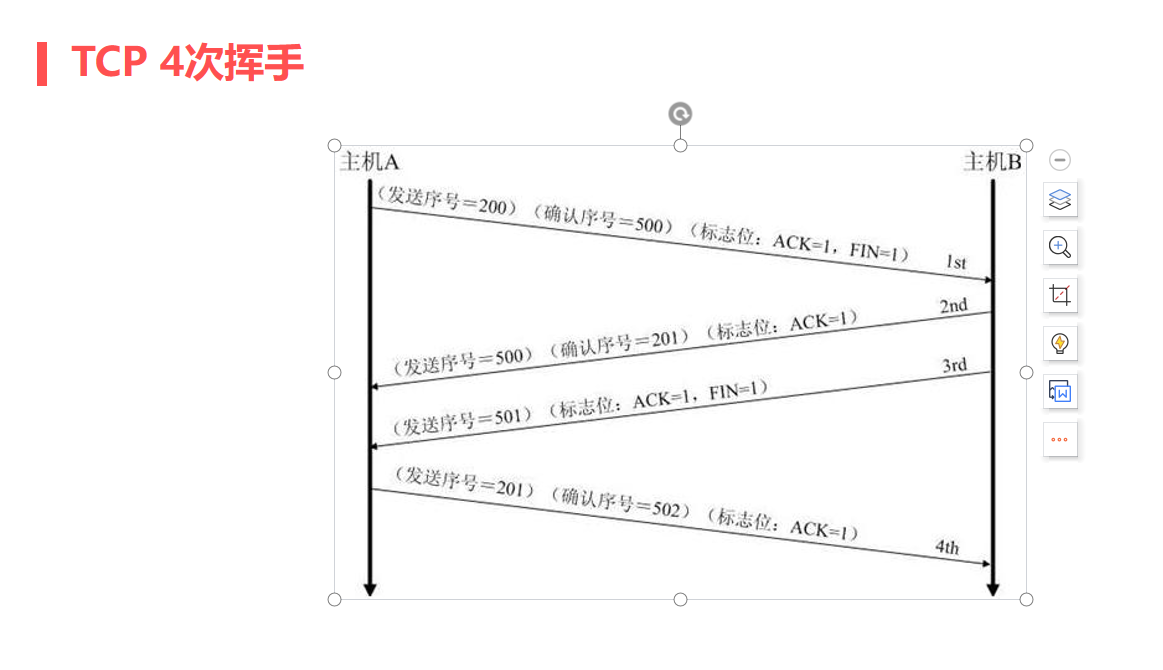


### 5.3 第三次握手



## 6. 四次挥手(重点)





1. 四次挥手发生在什么时候??  
 发生在断开连接的时候  
   
2. 断开连接的时候，使用的是FIN位和ACK位  
   
 第一次挥手：  
 //发起者对应答者说：你在吗？我想要断开连接  
 发起端给应答端，发送序号假设200， 标志位FIN=1  
 第二次挥手：  
 //应答者对发起者说：我在，可以断开连接  
 应答端给发起端，发送序号假设500， 标志位ACK=1， 确认序号201  
 第三次挥手：  
 //应答者对发起者说:咱两一起断开呗？??  
 应答端给发起端，发送序号假设501， 标志位ACK=1 FIN=1  
 第四次挥手:  
 //发起者对应答者说：好的，咱们一起断开  
 发起端给应答端，发送序号假设是201，标志位 ACK=1， 确认序号502

### 面试问题：

1. TCP和UDP有什么区别？？  
2. 为什么TCP安全可靠？？？  
3. 什么ACK攻击？？   
4. 请解释下三次握手过程  
5. 请解释下四次挥手过程  
6. 客户端连接服务器时发送( )包，服务器收到客户端连接请求发送( )包  
 A ACK B FIN C SYN D SYN+ACK  
7. 客户端和服务器的初始序号为(), 那么客户端SYN包序号为( ), 服务器的SYN包序号为( )  
 A 2000 B 2001 C 4000 D 4001

# 三. 心跳包

## 1. 抛出问题

如何判断客户端与服务器处于断开连接的状态？？？  
通过recv的返回值 <= 0，如果 <=0 说明服务器出问题了，但recv 返回值 <= 0的判断可能会漏掉某些情况，因为recv <= 0 并不能检测到所有断开情况

## 2. 心跳包作用

用于检测连接是否正常，还可以避免各种网络故障的出现，保证网络通信的稳定性和可靠性。在实际应用中，心跳包通常被广泛运用于游戏、视频、音频等实时性较高的应用中。

## 3. 心跳包原理

心跳包是由客户端定时发送给服务器端的，服务器端接收到心跳包后会对其进行回应。如果心跳包发送失败或者长时间没有得到回应，就说明连接出现了问题，需要及时进行处理。  
   
客户端 定期(每隔 0.1秒)给服务器发送1包数据，服务收到后回应，  
如果客户端检测到，发送了多包数据，服务器仍然没有回应，说明，服务器出问题了，  
客户端重连，直到服务器恢复为止

## 4. 功能展示

流程:  
 服务器功能: 等待客户端连接  
 一旦有客户端发送connect, 回应ok  
 客户端功能: 连接成功后，每隔 0.1秒钟发送connect, 接收服务器的ok  
 如果长时间收不到ok, 客户端重新连接服务器

#### client.c

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/socket.h>  
#include <netinet/in.h>  
int count = 3; //客户端接收不到服务器应答的次数，如果 == 0, 说明服务器不在线  
int server\_on = 0; //是否连接成功标志 0，断开 1 连接  
int i = 0;//计数  
int main(int argc, char \*argv[])  
{  
 if(argc != 3)  
 {  
 printf("忘记传递参数 IP Port ./client 192.168.31.170 55555\n");  
 exit(-1);  
 }  
 int fd;  
 struct sockaddr\_in youaddr;   
 youaddr.sin\_family = AF\_INET;  
 youaddr.sin\_port = htons(atoi(argv[2]));  
 youaddr.sin\_addr.s\_addr = inet\_addr(argv[1]);  
 char buf[100] = "connect";  
 while(1)  
 {   
 sleep(1);  
 if(!server\_on)//主要是断开状态，就会进入if条件句if(server\_on == 0)  
 {  
 fd = socket(AF\_INET, SOCK\_STREAM, 0);  
 int ret = connect(fd, (struct sockaddr \*)&youaddr, sizeof(youaddr));  
 if(ret == 0) //返回值 == 0代表连接成功  
 server\_on = 1; //连接成功   
 else  
 {  
 printf("connect error %d\n", i);  
 close(fd); //连接失败，因为是无限循环，1秒钟后再重连  
 i++;  
 continue; //只要连接失败就执行continue，程序就不会往下走  
 }  
 }  
 //程序能走到这里，说明 客户端肯定已经连接服务器成功  
 strcpy(buf, "connect");  
 send(fd, buf, sizeof(buf), 0); //发送connect  
 memset(buf, 0, sizeof(buf));  
 usleep(100000);// 1s == 1000000us  
  
 if(server\_on)  
 {  
 if(recv(fd, buf, sizeof(buf), MSG\_DONTWAIT) > 0) //MSG\_DONTWAIT 代表的是非阻塞接收  
 {  
 printf("buf is %s\n", buf);  
 if(strcmp(buf, "ok") != 0) //没收到ok  
 {  
 count--;  
 if(count == 0)//说明服务器已经有3次没有应答了  
 {  
 printf("server is disconnect\n");  
 server\_on = 0;//断开  
 }  
 }  
 else //收到了ok  
 count = 3;  
 }  
 else  
 {  
 count = 0;  
 printf("server is no answer\n");   
 server\_on = 0;  
 close(fd);  
 }  
 }  
 }  
 close(fd);   
}

#### server.c

#include <stdio.h>  
#include <string.h>  
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <fcntl.h>  
int fd1;  
int main()  
{  
 int newfd, ret;  
 char buf[100] = { 0 };   
 int fd = socket(AF\_INET, SOCK\_STREAM, 0);  
   
 //fcntl函数，对socket套接字的属性进行了设置   
 fcntl(fd, F\_SETFL, fcntl( fd, F\_GETFD, 0 )|O\_NONBLOCK );  
   
 struct sockaddr\_in myaddr;  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(55555);  
 myaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY); //192.168.31.179  
 bind(fd, (struct sockaddr \*)&myaddr, sizeof(myaddr));  
 listen(fd, 5);  
   
 while(1)  
 {  
 //代码没有变  
 newfd = accept(fd, NULL, NULL); //非阻塞等待客户端连接  
 if(newfd > 0)  
 {  
 printf("fd %d connect newfd %d\n", fd, newfd);  
 fd1 = newfd;  
 }  
 ret = recv(fd1, buf, sizeof(buf), MSG\_DONTWAIT);//非阻塞等待客户端发送数据  
 if(ret > 0)  
 {  
 printf("recv %s\n", buf);  
 if(strcmp(buf, "connect") == 0) //收到了connect  
 {  
 strcpy(buf, "ok");  
 send(fd1, buf, sizeof(buf), MSG\_DONTWAIT); //回应ok  
 }  
 }  
 usleep(100000); //0.1 s (usleep 是以微秒为单位)  
 }  
   
 close(fd);  
 close(newfd);  
}

## 5. 循环服务器

上面的服务器采用的是循环服务器，  
循环服务器要求: accept 和 recv 都是 非阻塞的  
非阻塞 方式接收数据:  
  
1. 非阻塞方式连接及接收  
  
 有些场合需要非阻塞  
 非阻塞: accept recv read ....  
 可以让这些函数不等  
 有时候，一个线程一直阻塞，效率有点低，  
 那么可以有两个解决办法  
 1) 不阻塞  
 2) 阻塞一小段时间(1秒)  
  
2. 如何实现非阻塞方式接收  
  
 文件I/O (不仅仅socket通信)  
 在打开文件后设置 O\_NONBLOCK (非阻塞方式)  
 //对socket套接字的属性进行了设置 ,让accept函数变成了非阻塞函数  
 fcntl(fd, F\_SETFL, fcntl( fd, F\_GETFD, 0 )|O\_NONBLOCK );   
   
 int fd = socket(AF\_INET, SOCK\_STREAM, 0); //socket描述符  
 (1) int flags = fcntl(fd, F\_GETFD, 0); //F\_GETFD 取出socket默认属性, 给flag  
 fcntl(file ----> control), 文件控制  
 F\_GETFD----> 读出文件处理方式标志位 (O\_RDWR | O\_TRUNC | .....)  
 取出文件标志位  
 (2) fcntl(fd, F\_SETFL, flags | O\_NONBLOCK);   
 给文件标志位增加 O\_NONBLOCK( 非阻塞 )  
 F\_SETFL ---> 设置文件处理方式标志位  
 accept 函数变成非阻塞方式

## 6. recv非阻塞

ssize\_t recv(int sockfd, void \*buf, size\_t len, int flags);  
flags ---> 通常是0 以阻塞方式接收  
 如果以非阻塞方式接收数，flags 设置为 MSG\_DONTWAIT  
   
recv(newfd, buf, sizeof(buf), MSG\_DONTWAIT);

#include <stdio.h>  
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <unistd.h>  
#include <fcntl.h>  
  
int main()  
{  
 char buf[100];  
 int fd = socket(AF\_INET, SOCK\_STREAM, 0); //SOCK\_STREAM 不要写成SOCK\_DGRAM  
 struct sockaddr\_in myaddr;  
 int flags = fcntl(fd, F\_GETFD, 0); //(1) 取出标志位  
 fcntl(fd, F\_SETFL, flags | O\_NONBLOCK); // (2)重新设置标志位  
 myaddr.sin\_family = AF\_INET;  
 myaddr.sin\_port = htons(33333);  
 myaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY); //192.168.30.252  
 bind(fd, (struct sockaddr \*)&myaddr, sizeof(myaddr));  
 listen(fd, 5); //设置同时连接最大值5个  
 int newfd = accept(fd, NULL, NULL); //非阻塞等待客户端连接  
 recv(newfd,buf, sizeof(buf), MSG\_DONTWAIT);//非阻塞接收  
 printf("11111111111111 %d\n", newfd); //-1 (正常 4)， 因为没有客户端连接，所以-1  
}  
//运行程序结果   
// 11111111111111111 瞬间打印，程序结束，证明accept函数和recv函数没有阻塞功能

# 四. 线程池

## 1. 抛出问题

多线程并发服务器，创建n多个线程，不断进行创建线程，销毁线程，浪费系统资源   
可以用线程池，提前创建好多个线程，利用休眠、唤醒来代替创建和销毁

## 2. 线程池作用

1. 降低资源消耗。通过重复利用已创建的线程降低线程创建、销毁线程造成的消耗。  
2. 提高响应速度。当任务到达时，任务可以不需要等到线程创建就能立即执行。  
3. 提高线程的可管理性。线程是稀缺资源，如果无限制的创建，不仅会消耗系统资源，还会降低系统的稳定性，使用线程池可以进行统一的分配、调优和监控

## 3. 线程池实现

#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
#include <sys/types.h>  
#include <pthread.h>  
#include <assert.h>  
/\*组成：  
线程池管理 用于创建及管理线程池  
工作线程 线程池中的线程--用于处理执行任务  
任务接口 提供处理任务的接口 供工作线程调度执行任务  
任务队列 存放待处理的任务 工作线程从中取出任务并执行   
  
\*/  
/\*  
\*线程池里所有运行和等待的任务都是一个Thread\_worker  
\*由于所有任务都在链表里，所以是一个链表结构  
\*/   
typedef struct worker  
{  
 /\*回调函数，任务运行时会调用此函数，注意也可声明成其它形式\*/  
 void \*(\*process) (void \*arg); //函数指针 指向的函数类型是 void \* (void \*arg); //process 指针保存就是线程执行的函数  
 void \*arg;/\*回调函数的参数\*/  
 struct worker \*next; //指向下一个节点的指针  
} Thread\_worker;  
  
  
/\*线程池结构\*/  
typedef struct  
{   
 //实现冬眠，需要互斥锁和唤醒条件配合使用  
 pthread\_mutex\_t queue\_lock; //互斥锁   
 pthread\_cond\_t queue\_ready; //唤醒条件  
 /\*链表结构，线程池中所有等待任务\*/  
 Thread\_worker \*queue\_head;//任务1-》任务2-》任务3 链表的头指针  
 /\*是否销毁线程池\*/  
 int shutdown;  
 //线程池中所有线程的tid  
 pthread\_t \*threadid; //多个ID，ID通过malloc申请  
 /\*线程池中允许的活动线程数目\*/  
 int max\_thread\_num;  
 /\*当前等待队列的任务数目\*/  
 int cur\_queue\_size;  
} Thread\_pool;  
  
  
int pool\_add\_worker (void \*(\*process) (void \*arg), void \*arg);  
void \*thread\_routine (void \*arg);  
  
  
static Thread\_pool \*pool = NULL;  
  
void pool\_init (int max\_thread\_num)//int max\_thread\_num = 3  
//创建线程池并初始化   
//参数：线程池最大线程数、任务队列最大任务数   
{  
 pool = (Thread\_pool \*) malloc (sizeof (Thread\_pool));  
   
   
 pthread\_mutex\_init (&(pool->queue\_lock), NULL);//初始化互斥锁   
 pthread\_cond\_init (&(pool->queue\_ready), NULL);//初始化唤醒条件  
  
 pool->queue\_head = NULL; //链表的头指针为空  
  
 pool->max\_thread\_num = max\_thread\_num; //3 ，函数调用的时候，传递的是3  
 pool->cur\_queue\_size = 0; //空，没有任务  
  
 pool->shutdown = 0; ///0 不销毁 1 销毁  
  
 pool->threadid =(pthread\_t \*) malloc (max\_thread\_num \* sizeof (pthread\_t)); //申请3个ID大小的空间，用来保存线程的ID  
 int i = 0;  
 for (i = 0; i < max\_thread\_num; i++)//批量创建线程max\_thread\_num个   
 {   
 pthread\_create (&(pool->threadid[i]), NULL, thread\_routine,NULL);  
 }  
}  
  
  
/\*向线程池中加入任务\*/  
//向任务队列中添加任务   
//参数 任务指针   
int pool\_add\_worker (void \*(\*process) (void \*arg), void \*arg)  
{  
 /\*构造一个新任务\*/  
 Thread\_worker \*newworker =(Thread\_worker \*) malloc (sizeof (Thread\_worker));  
 newworker->process = process;  
 newworker->arg = arg;  
 newworker->next = NULL;  
  
 pthread\_mutex\_lock (&(pool->queue\_lock));//加锁   
 /\*将任务加入到等待队列中\*/  
 Thread\_worker \*member = pool->queue\_head;  
   
   
 if (member != NULL)//链表不为空 对列 尾插 执行任务 头删   
 {  
 while (member->next != NULL)  
 member = member->next;   
 //循环结束后，让member指针，指向了链表的最后一个节点，相当于我们之前学的ptail尾巴指针  
   
 member->next = newworker; //尾插法  
 }  
 else // member == NULL 当最开始链表为空的时候，新创建的节点就是头节点  
 {  
 pool->queue\_head = newworker;  
 }  
  
 pool->cur\_queue\_size++;  
 pthread\_mutex\_unlock (&(pool->queue\_lock));  
 //唤醒一个等待线程 如果所有线程都在忙碌，这句没有任何作用  
 // 唤醒阻塞在条件变量上的所有线程的顺序由调度策略决定，如果线程的调度策略是SCHED\_OTHER类型的，系统将根据线程的优先级唤醒线程。  
 //如果没有线程被阻塞在条件变量上，那么调用pthread\_cond\_signal()将没有作用。  
  
//添加任务后，唤醒线程，干活  
 pthread\_cond\_signal (&(pool->queue\_ready)); //发送信号给正在阻塞的线程，阻塞的线程解除阻塞，执行process  
 return 0;  
}  
  
  
/\*销毁线程池，等待队列中的任务不会再被执行，但是正在运行的线程会一直  
把任务运行完后再退出\*/  
int pool\_destroy ()  
{  
 if (pool->shutdown)  
 return -1;/\*防止两次调用\*/  
 pool->shutdown = 1;  
  
 //唤醒所有等待线程，线程池要销毁  
 pthread\_cond\_broadcast (&(pool->queue\_ready));  
  
 /\*阻塞等待线程退出，否则就成僵尸了\*/  
 int i;  
 for (i = 0; i < pool->max\_thread\_num; i++)  
 {   
 pthread\_join (pool->threadid[i], NULL);  
 }  
 free (pool->threadid);  
  
 /\*销毁等待队列\*/  
 Thread\_worker \*head = NULL;  
 while (pool->queue\_head != NULL)  
 {  
 head = pool->queue\_head;  
 pool->queue\_head = pool->queue\_head->next;  
 free (head);  
 }  
 /\*条件变量和互斥量也别忘了销毁\*/  
 pthread\_mutex\_destroy(&(pool->queue\_lock));  
 pthread\_cond\_destroy(&(pool->queue\_ready));  
   
 free (pool);  
 /\*销毁后指针置空是个好习惯\*/  
 pool=NULL;  
 return 0;  
}  
  
  
//工作线程的主要工作为处理任务，当任务队列不为空的时候，工作线程直接从队列头取出一个任务并执行；  
//当任务队列为空的时候，工作线程将阻塞直到有任务添加进来；  
void \*thread\_routine (void \*arg)  
{  
 printf ("starting thread 0x%x\n", (unsigned int)pthread\_self ());  
 while (1)  
 {  
 pthread\_mutex\_lock (&(pool->queue\_lock));  
 //如果等待队列为0并且不销毁线程池，则处于阻塞状态  
 if (pool->cur\_queue\_size == 0 && !pool->shutdown)  
 {  
 printf ("thread 0x%x is waiting\n", (unsigned int)pthread\_self ());  
   
 //处于阻塞等待，休眠状态，被唤醒   
 pthread\_cond\_wait (&(pool->queue\_ready), &(pool->queue\_lock)); //让线程休眠  
 }  
  
 //线程池销毁  
 if (pool->shutdown)  
 {  
 pthread\_mutex\_unlock (&(pool->queue\_lock));  
 printf ("thread 0x%x will exit\n", (unsigned int)pthread\_self ());  
 pthread\_exit (NULL);  
 }  
  
 printf ("thread 0x%x is starting to work\n", (unsigned int)pthread\_self ());//被叫醒了，开始干活  
   
 //等待队列长度减去1，并取出链表中的头元素  
 pool->cur\_queue\_size--;  
   
 Thread\_worker \*worker = pool->queue\_head;//头删 执行任务   
   
 pool->queue\_head = pool->queue\_head->next;  
 //pool->queue\_head = worker->next;  
   
 pthread\_mutex\_unlock (&(pool->queue\_lock));  
  
 /\*调用回调函数，执行任务\*/  
 (\*(worker->process)) (worker->arg);//通过结构体里面的函数指针，调用函数myprocess  
 free (worker);  
 worker = NULL;  
 }  
 pthread\_exit (NULL);  
}  
   
void \*myprocess (void \*arg) //这个myprocess才是真正干活的线程函数，将 整数打印输出  
{  
 //pthread\_self () 获取线程自己ID  
 //getpid(); 获取自己进程的ID  
 printf ("threadid is 0x%x, working on task %d\n", (unsigned int)pthread\_self (),\*(int \*) arg);  
 sleep (1);  
 return NULL;  
}  
  
int main (int argc, char \*\*argv)  
{  
 pool\_init (3); //创建3个线程，默认是休眠的  
 int \*workingnum = (int \*) malloc (sizeof (int) \* 10); //int a[10]  
 int i;  
 for (i = 0; i < 10; i++)  
 {  
 pool\_add\_worker (myprocess, &workingnum[i]); //给线程添加任务，执行myprocess  
 }  
 sleep (5);  
 pool\_destroy (); //销毁线程  
 free (workingnum);  
 return 0;  
}

## 4. 条件唤醒+互斥锁解决同步问题

# @复习

1. wireshark软件：网络抓包工具  
 怎么抓包的？？  
 (1)选取一个网卡  
 (2)在显示过滤器可以通过 协议 IP地址 端口号进行过滤数据包  
  
2. DHCP分配IP地址过程  
   
 (1) 客户端发送广播 发送 DHCP Discover报文  
 (2) 多个DHCP服务器收到DHCP Discover报文之后，会回一个 DHCP Offer报文，并带有IP地址  
 (3) 客户端发送广播 发送 DHCP Request报文，告诉DHCP服务器，选取了哪个  
 (4) 被选中的DHCP服务器 发送一个 DHCP ACK报文， 确认租约时间  
  
3. 过滤规则  
 通过协议 // http dns dhcp tcp udp icmp  
 IP地址 // ip.addr == 192.168.31.111 ip.src == 192.168.31.111 ip.dst == 192.168.31.111  
 端口号 // tcp.port == 80 udp.port == 80 tcp.srcport == 80 tcp.desport == 80  
 用逻辑关系运算符 // ip.addr == 192.168.31.111 and icmp  
  
4. 抓到包  
   
 物理层 // 数据帧  
 数据链路层 //以太网头  
 网络层 //IP头  
 传输层 //Tcp头 或 UDP头  
 应用层 //dhcp头 http头  
   
5. 数据链路层 //以太网头   
   
 以太网头占 14个字节  
 目的MAC地址 源MAC地址 类型  
   
6. 网络层 //IP头，为数据包选择传输路径 选择路由  
   
 IP头占20个字节  
 源IP地址 目的IP地址  
  
7. 传输层 // 实现端对端的通信  
   
 UDP头 //8个字节  
 TCP头 //20个字节  
 源端口号 目的端口号  
   
8. TCP的三次握手和四次挥手   
 1.TCP三次握手发生在哪个时候？？  
 连接的时候，使用SYN和ACK标志位，客户端的connect函数 与 服务器端的accept函数  
 (1)第一次握手：客户端发送给服务器  
 //服务器，你在吗？我想连接你  
 发送序号是200， 标志位SYN=1  
 (2)第二次握手：服务器发送给客户端  
 //我在,你连接我吧？  
 发送序号500， 标志位 SYN=1 ACK=1， 确认序号201  
 (3)第三次握手：客户端发送给服务器  
 //好的，那么我连接你  
 发送序号201， 标志位 ACK=1，确认序号501  
   
 2.TCP四次挥手发生在哪个时候？？  
 断开连的时候，使用FIN和ACK标志位，客户单close函数 与 服务器端colose函数  
 (1)第一次挥手：发起端给应答端  
 //在吗？我想要断开连接  
 发送序号是200， 标志位FIN=1  
 (2)第二次挥手：应答端给发起端  
 //我在，可以断开  
 发送序号500， 标志位 ACK=1， 确认序号201  
 (3)第三次挥手: 应答端给发起端  
 //咱两一起断开？？  
 发送序号501， 标志位 FIN=1，ACK=1  
 (4)第四次挥手：发起端给应答端  
 //好的，一起断开  
 发送序号201，标志位ACK=1，确认序号502  
   
 3.什么是ACK攻击：发送在TCP三次握手的第三次，握完2次手后，第3次不握手

# ---------------------------------

# Day05

# 一. 数据库

## 1. 数据库分类

### （1）大型数据库

oracle甲骨文公司 oracle数据库  
IBM公司 DB2  
银行、电信、qq、百度、京东、阿里

### （2）中型数据库

微软 sql server

### （3）小型数据库

mysql (Web网站)  
sqlite (用在嵌入式设备、手机)  
安装使用特别方便

## 2. 安装数据库

sqlite(平台ubuntu)  
(1) 用下载好的安装包安装  
  
 libsqlite3-0\_3.7.2-1ubuntu0.1\_i386.deb (.deb的安装包可以用dpkg -i 来进行安装)  
 libsqlite3-0\_3.7.2-1ubuntu0.1\_i386\_1.deb  
 libsqlite3-dev\_3.7.2-1ubuntu0.1\_i386\_2.deb  
 //将三个.deb结尾的安装包拖入虚拟机，依次安装.deb包  
   
 //方法一：\*.deb结尾的安装包全部安装  
   
 sudo dpkg -i \*.deb  
   
 //方法二：逐个安装  
 sudo dpkg -i libsqlite3-0\_3.7.2-1ubuntu0.1\_i386\_1.deb   
 sudo dpkg -i libsqlite3-dev\_3.7.2-1ubuntu0.1\_i386\_2.deb  
 sudo dpkg -i sqlite3\_3.7.2-1ubuntu0.1\_i386\_3.deb  
   
 //安装三个包后，执行下面命令，查看当前sqlite3数据库版本   
 sqlite3 -version

## 3. 数据库相关概念

(1)数据库  
 db // database   
 将所有的数据，各种类型，描述各种事物的数据放到一起，  
 如：my.db文件,就是一个数据库文件,数据库文件以.db结尾  
 数据库文件以.db结尾  
   
 \*\*\*\*.db  
   
(2)数据表  
  
 描述某一个事物的基本信息的表  
 举例：  
 图书管理系统  
 学生信息表  
   
 编号 姓名 性别 年龄 班级 爱好  
 001 阿丹 女 19 19121 睡觉  
 002 阿三 男 20 19121 打豆豆  
  
 图书信息表  
   
 图书编号 书名 出版社 作者 分类  
   
(3)字段   
 列的名字称为字段  
   
 编号  
 姓名  
 性别  
 年龄  
 班级  
 爱好

## 4. 数据库的使用

1. 数据库扩展名：数据库文件的扩展名一定以.db结尾  
2. 以.开头的是命令  
3. 以;号结尾的是sqlite语句

### 4.1 打开数据库

打开数据库(或者创建数据库)  
sqlite3 my.db //如果当前目录存在，my.db，直接打开，不存在，创建并打开

### 4.2 数据库命令

输入 .开头的命令(非sql语句命令)  
 .quit //退出   
 .help //查看帮助文档   
 .table //查看数据库的数据表   
 ctrl + l //清屏

### 4.2 数据库操作语句

#### (1)创建一个表

create table 表名 (字段 类型，字段 类型，......);  
  
1. 创建一个学生信息表stu\_info  
 学生信息表：字段有：姓名，学号，体重  
   
  
 sqlite> create table stu\_info (name varchar(20),age integer,weight float);  
 sqlite> .table  
 stu\_info  
 sqlite>  
   
2. 查询表中的所有内容   
 select \* from stu\_info; // \* 代表显示所有字段

#### (2)插入一行信息

1.向学生信息表中插入数据  
   
 sqlite> insert into stu\_info values('zhangsan',19,98.8);  
  
2.查询表中的内容  
  
 (1)显示所有字段记录  
   
 select 和 from 之间 代表的是要显示的字段   
   
 sqlite> select \* from stu\_info; //\*代表所有字段  
  
 sqlite> .header on //显示的时候加上表头 字段的名字  
 sqlite> .mode column //显示的时候按列对齐  
   
 (2)显示指定字段  
 //显示哪几个列，取决于 select 和 from中间写什么  
 sqlite> select \* from stu\_info; // \* 代表显示所有字段  
 sqlite> select name,age from stu\_info; //只显示 name 和 age 字段  
 sqlite> select name from stu\_info; //只显示 name 字段  
   
 上下键可以调用出历史sql语句和命令  
   
3. 查询一条指定记录  
 sqlite> select \* from stu\_info where age = 20; //显示年龄是20岁的人信息  
 sqlite> select name from stu\_info where age = 20; //显示年龄是20岁的人，只显示name字段

#### #练习1:

a、创建学生成绩表(stu\_score 字段如下)  
   
 学生编号 数学 语文 英语  
 char(10) intger integer integer  
 number math chinese english  
   
b、向学生成绩中添加三条记录  
  
 001 100 60 70  
 002 80 90 90  
 003 90 70 80   
   
c、查询出编号003的记录和数学成绩为100的记录  
  
 create table stu\_score (number varchar(20), math integer, chinese integer,english integer);  
 insert into stu\_score values('001',100,60,70);  
 insert into stu\_score values('002',80,90,90);  
 insert into stu\_score values('003',90,70,80);  
 select \* from stu\_score where number = '003';  
 select \* from stu\_score where math = 100;

#### (3)其他各种查询

1. 查询年龄是20的记录  
  
 sqlite> select \* from stu\_info where age = 20; //显示年龄是20岁的信息  
   
2. 查询年龄 > 19的记录  
  
 sqlite> select \* from stu\_info where age > 19; //显示年龄大于19岁的信息  
   
3. 查询记录 只想看年龄大于19岁的姓名、年龄字段  
  
 sqlite> select name,age from stu\_info where age > 19; //显示年龄大于19岁的信息  
   
4. 如果想按顺序输出所有记录，按成绩排序  
  
 sqlite> select \* from stu\_info order by score; //按照成绩 从小到大的顺序显示信息  
 sqlite> select \* from stu\_info order by score desc;//按照成绩 dest 从大到小的顺序显示信息  
  
5. 模糊查询(假设有一个人名我记不住了，但能记住名字中的一个字， 就可以用模糊查询)  
 //马什么梅  
 //什么冬梅  
 //马东什么  
 sqlite> select \* from stu\_info where name like 'da%'; //只能搜索以da开头的名字  
 sqlite> select \* from stu\_info where name like '%lang';//只能搜索以lang结尾的名字  
 sqlite> select \* from stu\_info where name like '%an%';//名字中包含an的(包括以an开头和以an结尾)  
   
6. 多条件查询  
   
 想查询年龄大于 25 并且 名字中带'a' 的学生信息  
 sqlite> select \* from stu\_info where age > 25 and name like '%a%';  
   
 and ----> &&  
 or ----> ||  
   
7. 统计  
   
 1) 统计表中有多少人  
   
 sqlite> select count(\*) from stu\_info;  
   
 2) 统计表中年龄为19岁的学生有多少人  
   
 sqlite> select count(\*) from stu\_info where age = 19;   
   
 3) 统计所有人的年龄总和  
  
 sqlite> select sum(age) from stu\_info;   
  
 4) 统计所有人的年龄平均值  
  
 sqlite> select avg(age) from stu\_info;

#### (4)删除指定记录

1. 删除满足指定条件的目录  
   
 sqlite> delete from stu\_info where name = 'asan'; //删除名字是asan的信息  
   
2. 删除所有记录(清空表)  
  
 sqlite> delete from stu\_info; //由于没有条件限制，相当于清空表

#### (5)修改记录

更新 表名 修改的字段 指定某一条  
   
sqlite> update stu\_score set chinese = 100 where number = '002';  
 将学号为002的学生，语文成绩修改为100分

#### (6)多表联合查询

两张表联合查询，前提条件，必须有一个共有字段，将两张表关联在一起  
学生成绩表stu\_score   
number(varchar(20)) math chinese english   
001 100 80 70  
002 90 80 70  
003 60 80 70  
  
//学生信息表stu\_info   
number(varchar(20) name age   
001 asan 18  
002 haha 19  
003 xixi 17  
   
查询显示 数学成绩 学生的基本信息 满足条件为 数学成绩大于80  
  
select stu\_score.math ,stu\_info.\* from stu\_info, stu\_score where stu\_info.number = stu\_score.number and stu\_score.math > 80;  
   
多表联合查询的时候，条件要加上共有字段相等 关联在一起

#### (7)删除表

drop table 表名  
drop table stu\_info; //删除表

## 5. 通过脚本执行sql语句

将所有的sql语句或命令，写到一个文件中，然后执行这个文件，就把所有语句都执行了  
create table if not exists worker\_info (name varchar(20), salary float);  
将所有数据库操作语句和命令放到一个脚本文件中,脚本文件必须以.sql结尾  
  
work.sql   
drop table if exists worker\_info; //如果worker\_info表存在，执行删除这个表  
create table worker\_info (name varchar(20), salary float);  
  
insert into worker\_info values('zhangsan',10000.23);  
insert into worker\_info values('aaaa',103000.23);  
insert into worker\_info values('bbbb',100400.23);  
insert into worker\_info values('cc',102000.23);  
  
.header on  
.mode column`  
  
select \* from worker\_info;  
  
  
ls /home/linux > a.txt //输出重定向  
ls /home/linux >> a.txt //追加重定向  
  
执行 输入重定向   
sqlite3 my.db < worker.sql

#### #练习2：

图书管理系统  
  
你用脚本创建三张表  
1 创建图书基本信息数据表 图书编号 图书名称 出版社 作者 出版日期  
2 创建办理借阅卡的学生基础表 学生编号 学生姓名 所在系 所在班  
3 创建学生借阅表 学生编号 图书编号 借阅日期 借阅时间 是否归还  
4 添加记录并查询   
  
sqlite3 my.db   
练习sql语句使用   
  
(1)查询指定出版社的图书信息  
(2)查询指定作者的图书信息   
(3)查询某个编号的学生基本信息  
(4)查询已借出图书的图书名称  
(5)修改某个编号的图书的基本信息  
(6)查询已归还图书的名字及归还者名字   
(7)删除借阅时间大于2天的借阅信息

答案:  
////////////////////////////////////////////////////////////////////////////  
图书信息表 book\_info.sql  
////////////////////////////////////////////////////////////////////////////  
drop table if exists book\_info;  
  
create table book\_info (number varchar(20), name varchar(20), press varchar(20),  
 author varchar(20),date varchar(20));  
  
insert into book\_info values('1001','shuhuzhuan','renminyoudian','shinaian','1934.3.23');  
insert into book\_info values('1002','sanguoyanyi','qinghuadaxue','luoguanzhong','1931.1.23');  
insert into book\_info values('1003','hongloumeng','huaqingyuanjian','caoxueqin','1924.3.23');  
insert into book\_info values('1004','xiyouji','renminyoudian','wuchengen','1913.3.23');  
  
.header on  
.mode column  
select \* from book\_info;  
  
////////////////////////////////////////////////////////////////////////////  
学生基本信息表 stu\_info.sql  
////////////////////////////////////////////////////////////////////////////  
  
  
drop table if exists stu\_info;  
  
create table stu\_info ( number varchar(20),name varchar(20),faculty varchar(20),class varchar(5));  
  
insert into stu\_info values('00001','zhangsan','xinxigongcheng','4');   
insert into stu\_info values('00002','lisi','meishu','2');   
insert into stu\_info values('00003','wangwu','yinyue','1');   
insert into stu\_info values('00004','zhaoliu','tiyu','3');   
  
  
.header on  
.mode column  
select \* from stu\_info;   
   
////////////////////////////////////////////////////////////////////////////  
学生借阅表 bro\_info.sql  
////////////////////////////////////////////////////////////////////////////  
  
drop table if exists bro\_info;  
  
create table bro\_info (stu\_num varchar(20),book\_num varchar(20),bro\_date varchar(20),  
 bro\_time varchar(20),whether varchar(5));  
  
insert into bro\_info values('00001','1001','2017.9.23','13 : 24','yes');  
insert into bro\_info values('00002','1002','2017.7.23','15 : 24','no');  
insert into bro\_info values('00003','1003','2017.8.23','13 : 27','no');  
insert into bro\_info values('00004','1004','2017.1.23','18 : 12','yes');  
  
.header on  
.mode column  
select \* from bro\_info;  
  
////////////////////////////////////////////////////////////////////////////  
操作  
////////////////////////////////////////////////////////////////////////////  
  
select \* from book\_info where press = 'qinghuadaxue';  
select \* from book\_info where author = 'wuchengen';  
select \* from stu\_info where number = '00002';  
select book\_info.name from book\_info,bro\_info where bro\_info.whether = 'no' and bro\_info.book\_num = book\_info.number;  
update book\_info set date = '2000.2.14' where author = 'luoguanzhong';  
select book\_info.name,stu\_info.name from book\_info,stu\_info,bro\_info where bro\_info.whether = 'yes' and book\_info.number = bro\_info.book\_num and stu\_info.number = bro\_info.stu\_num;  
delete from bro\_info where bro\_date > 2;

# 二. C语言操作数据库

## 1. 打开数据库sqlite3\_open

(1)头文件及函数原型  
 #include <sqlite3.h>  
 int sqlite3\_open(char \*path, sqlite3 \*\*db);  
  
 //调用  
 sqlite3 \*db = NULL; //用来指向打开的数据库文件 数据库操作句柄  
 sqlite3\_open("./my.db",&db); //如果my.db文件存在，直接打开，如果不存在，创建并打开  
   
  
   
(2)功能：  
 打开数据库文件，如果文件存在直接打开，不存在创建并打开  
  
(3)参数说明：  
   
 path 要打开的数据库，如果不存在，就创建 ，如果存在，直接打开//可以相对路径也可以是绝对路径  
 db [出参]， 数据打开成功，从这个参数传出一个指向数据库的指针   
   
(4)返回值：  
 0 成功  
 非0 失败

## 2.关闭数据库sqlite3\_close

int sqlite3\_close(sqlite3 \*db);  
  
sqlite3\_close(db); //参数：数据库操作句柄  
  
  
功能：关闭sqlite数据库  
返回值：成功返回0，失败返回错误码

### 案例代码：

#### my.h

#ifndef \_MY\_H  
#define \_MY\_H  
  
#include <stdio.h>  
#include <string.h>  
#include <time.h>  
#include <stdlib.h>  
#include <fcntl.h>  
#include <unistd.h>  
#include <errno.h>  
#include <dirent.h>  
#include <pthread.h> // p process进程 thread 线程  
#include <semaphore.h> //semaphore信号量  
#include <signal.h>   
#include <sys/shm.h> //sh share m memory  
#include <sys/msg.h> //msg message   
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <sqlite3.h>  
  
#endif

## 3. 执行数据库操作的sql语句

(1)头文件及函数原型  
 //sqlite3\_exec 执行数据库操作语句， 创建 增 删 改  
 #include <sqlite3.h>  
 int sqlite3\_exec(sqlite3 \*db, const char \*sql, sqlite3\_callback callback, void \*data, char \*\*errmsg);  
   
 //调用  
 char sql[200] = "create table stu\_info (name varchar(20), age integer, score float);"  
 sqlite3\_exec(db, sql, NULL, NULL, NULL);  
 sqlite3\_exec(db, "drop table stu\_info;", NULL, NULL, NULL);  
  
  
(3)功能：  
 可以执行数据库操作语句  
 通常用来执行查询以外的操作语句(创建、增、删、改)  
   
(4)参数:  
 db ， open的第二个参数，db指向打开的数据库  
 sql， 操作数据库的语句 "delete from stu\_info;"   
 callback 如果执行select , select 通过这个值获取， 可以写NULL  
 void \*data 可以写NULL  
 errmsg ： 如果数据库操作出错，此处保存错误信息  
(5)返回值：  
 成功返回0，失败返回错误编号

#include "my.h"  
//gcc test.c -lsqlite3  
int main(int argc, const char \*argv[])  
{  
 //1.打开数据库  
 sqlite3\* db = NULL;  
 //如果you.db存在，直接打开，不存在，创建并打开  
 int ret = sqlite3\_open("./you.db", &db);  
 if(ret != 0)// ret == 0打开成功  
 {  
 perror("sqlite3\_open failed");  
 exit(-1);  
 }  
 printf("sqlite3\_open sucessful!!\n");  
  
 //char sql[100] = "create table if not exists stu\_info (name varchar(20), age integer, score float);";  
 //char sql[100] = "insert into stu\_info values('asan',19,98.5);";  
 char sql[100] = "delete from stu\_info";  
 char\* errmsg = NULL;//指针变量用来保存 数据库操作语句执行失败的具体原因字符串的首地址  
 ret = sqlite3\_exec(db, sql, NULL, NULL, &errmsg);  
 if(ret != 0)  
 {  
 printf("sqlite3\_exec 失败的原因:%s\n",errmsg);  
 exit(-1);  
 }  
 printf("sqlite3\_exec sucessful!!\n");  
  
 //2.关闭数据库  
 sqlite3\_close(db);  
  
 return 0;  
}

## 4. 查询语句

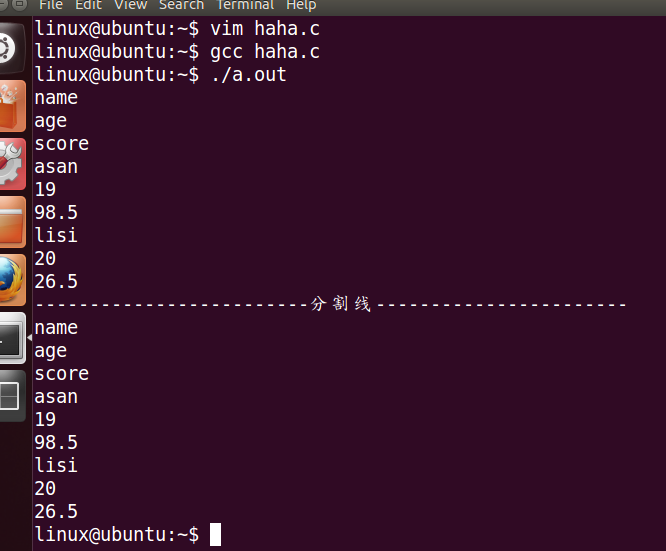
(1)头文件及函数原型  
  
 #include <sqlite3.h>  
 int sqlite3\_get\_table(sqlite3 \*db, const char \*sql, char \*\*\*resultp, int \*nrow, int \*ncolumn, char \*\*errmsg);  
   
 //调用  
 char\*\* p = NULL;  
 int row,column;  
 char\* errmsg = NULL;  
 char sql[200] = "select \* from stu\_info;"  
 sqlite3\_get\_table(db, sql, &p, &row, &column,&errmsg);  
  
(3)功能：  
  
 执行select查询语句的时候，调用sqlite3\_get\_table()  
  
(4)参数:  
  
 db：数据库句柄, open的第二个参数  
 sql：SQL语句， select  
 resultp：用来指向sql执行结果的指针  
 nrow：满足条件的记录的数目,行数 不包括表头  
 ncolumn：每条记录包含的字段数目，相当于列数  
 errmsg：错误信息指针的地址， 如果语句执行出错，那么errmsg中是错误信息  
   
(5)返回值：   
 成功返回0，失败返回错误编号

### 案例代码:

0 1 2  
name age score  
3 4 5  
asan 19 98.5  
6 7 8  
cccc 20 98.5   
9 10 11  
cddd 20 100.5  
12 13 14  
czzzz 10 100.5  
15 16 17  
  
#include "my.h"  
//gcc test.c -lsqlite3  
int main(int argc, const char \*argv[])  
{  
 //1.打开数据库  
 sqlite3\* db = NULL;  
 int ret = sqlite3\_open("./you.db", &db);  
 if(ret != 0)// ret == 0打开成功  
 {  
 perror("sqlite3\_open failed");  
 exit(-1);  
 }  
 printf("sqlite3\_open sucessful!!\n");  
  
 //char sql[100] = "select \* from stu\_info";  
 char sql[100] = "select name,age from stu\_info where age > 19;";  
  
 char\*\* resultp = NULL;//用来保存查询结果的字符指针数组的首地址  
 int row,column;//用来保存查询结果的行数和列数,注意,行数不包含表头  
 char\* errmsg = NULL;//指针变量用来保存 数据库操作语句执行失败的具体原因字符串的首地址  
 ret = sqlite3\_get\_table(db, sql, &resultp, &row, &column, &errmsg);  
 if(ret != 0)  
 {  
 printf("sqlite3\_get\_table失败的原因:%s\n",errmsg);  
 exit(-1);  
 }  
 printf("sqlite3\_get\_table sucessful!!\n");  
 printf("row is %d column is %d\n",row,column);  
 //将查询结果的字符指针数组打印输出  
 int i;  
 //row+1 是将表头计算在内进行打印  
 for(i = 0; i < (row+1)\*column; i++)  
 {  
 printf("%s ",resultp[i]);  
 if((i+1)%column == 0)  
 printf("\n");  
 }  
   
 //2.关闭数据库  
 sqlite3\_close(db);  
  
 return 0;  
}

## 5. 三级指针

#include <stdio.h>  
#include <stdlib.h>  
  
  
//定义一个fun函数,将fun函数中字符指针数组的首地址返回,然后在main函数中打印  
  
char\*\* fun1()  
{  
 static char\* name[] = {"name","age","score","asan","19","98.5","lisi","20","26.5"};  
 return name;  
}  
  
void fun2(char\*\*\* r)//通过参数上地址传递的方式得到数组的首地址  
{  
 static char\* name[] = {"name","age","score","asan","19","98.5","lisi","20","26.5"};  
 //r保存的是main函数中q的地址,所以\*r代表的就是main函数中的q  
 \*r = name;  
}  
  
  
int main(int argc, const char \*argv[])  
{  
 //通过返回值的形式获取数组的首地址  
 char\*\* p = fun1();  
 int i;  
 for(i = 0; i < 9; i++)  
 {  
 printf("%s\n",p[i]);  
 }  
 printf("-------------------------分割线-----------------------\n");  
  
 //通过地址传递的方式来获取数组的首地址  
 char\*\* q = NULL;  
 fun2(&q);  
 for(i = 0; i < 9; i++)  
 {  
 printf("%s\n",q[i]);  
 }  
  
 return 0;  
}



## 6. 二级指针传递字符指针数组

#include <stdio.h>  
#include <stdlib.h>  
  
void showName(char\*\* p, int n)  
{  
 int i;  
 for(i = 0; i < n; i++)  
 {  
 printf("%s\n",p[i]);//p[i] == name[i]  
 }  
}  
  
void getMemory(int\*\* p)  
{  
 \*p = malloc(sizeof(int));  
 //容错判断  
}  
  
  
int main(int argc, const char \*\*argv)  
{  
 //字符指针数组,数组中的每个元素是字符指针char\*类型  
 //sizeof(name) ---> sizeof(数组) = sizeof(元素类型)\*元素个数 = sizeof(char\*) \* 5 = 20   
 //sizeof(name[0]) ---> 4 name[0]类型char\*  
 int i;  
 char\* name[5] = {"./a.out","aa","bb","cc","dd"};  
 //name 数组名就是数组的首地址  
 //name == &name[0]  
 //name[0]的类型char\*  
 //&name[0]的类型char\*\*,所以name代表的是char\*\*  
 //char\*\* p = name;  
 showName(name, 5);  
  
 int\* q = NULL;  
 getMemory(&q);  
 if(q == NULL)  
 {  
 perror("malloc failed");  
 }  
 \*q = 200;  
 free(q);  
  
 return 0;  
}

# 作业：

(1)从键盘输入表名，创建表（sqlite3\_exec()  
 sprintf(sql,"create table %s (.............)",tablename);  
(2)从键盘输入编号，姓名，年龄，性别，插入到stu\_info中,输入3个人的信息（sqlite3\_exec()）  
(3)打印输出表中所有信息（sqlite3\_get\_table()）  
(4)再输入一个年龄，删除 > 此年龄的学生（sqlite3\_exec()）  
(5)打印输出表中所有信息（sqlite3\_get\_table()

# @复习:

# 电子词典项目

# 一. 功能介绍

///////////////////////////////////////////////////////////////////////////////////  
////////////////////////////////////客户端/////////////////////////////////////////  
///////////////////////////////////////////////////////////////////////////////////  
  
//界面1  
   
 printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  
 printf("\* 电 子 词 典 项 目 \*");   
 printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  
 printf("\n\* 1.注册 2.登录 3.退出 \*\n");  
 printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  
 printf("\n请 输 入 您 的 选 择:\n\n");  
   
 (1)注册 //将要注册的用户名和密码发送给服务器  
   
 (2)登录 //将要登录的用户名和密码发送给服务器   
   
 (3)退出 //结束客户端程序   
   
//界面2   
   
 printf("\n/////////////////////////////////////////////////////");  
 printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  
 printf("\* 欢　迎　使　用　电　子　词 　 \*");  
 printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  
 printf("\n\* 1.查询单词 2.查询历史单词 3.返回上一级 4.退出 \*\n");  
 printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  
 printf("/////////////////////////////////////////////////////\n");  
 printf("\n请 输 入 您 的 选 择:\n\n");  
   
   
 (1)查询单词 //将单词发送给服务器   
   
 (2)查询历史单词//将请求类型修改，发送给服务器   
   
 (3)返回上一级 //进入菜单一界面   
   
 (4)退出 //结束程序  
   
   
///////////////////////////////////////////////////////////////////////////////////  
///////////////////服务器( 数据库sqlite3, 文件I/O, 多进程, 网络通信TCP通信 并发服务器)////////////////  
///////////////////////////////////////////////////////////////////////////////////  
//////服务器端实现的事  
  
 (1)注册   
   
 //将客户端发送过来的用户名和密码去数据库中检验，若用户名未被注册，将用户名和密码保存到数据库文件中  
 //若用户名已经在数据库中被注册了，提示客户端需要重新注册  
   
 (2)登录   
   
 //将客户端发送过来的用户名和密码去数据库中检验，若用户名和密码正确，提示客户端登录成功，客户端进入下一个界面  
 //若用户名和密码不正确，提示客户端登录失败，需要重新登录  
   
 (3)查询单词  
   
 //将客户端发送过来的单词去dict.txt文件中进行逐行比对，若查到该单词将单词及解释发送给客户端同时需要将  
 //用户名 和 查询单词 及 查询时间 保存到数据库文件中  
 //若查询到此词不存在，提示客户端单词不存在  
   
   
 (4)查询历史单词   
   
 //将数据库中保存的 用户名 和 查询单词 及 查询时间 发送给客户端

# 二. 实现过程

//////////////////////////////////////////////////////////////////  
//1.创建并发服务器   
//2.实现注册功能   
//3.实现登录功能   
//4.实现查单词功能   
//5.实现查询历史单词功能  
  
1.服务器和客户端发送和接收数据结构体   
  
#define R 1 //注册 register  
#define L 2 //登录 login  
#define Q 3 //退出 quit  
#define F 4 //查询单词 find  
#define H 5 //查询历史单词 history   
  
 typedef struct   
 {  
 int type;//1 注册， 2 登录 3 查询单词 4 查询历史记录  
 char name[30];//1 2 用户名   
 char data[256]; //客户端：1 2 密码 3 要查询单词 #服务器端：3 单词解释 4 历史记录   
 }MSG;  
  
 //标志位思想   
   
 客户端 type == R 注册请求 ---> 服务器端 收到siwtch(type) == R do\_register();  
 char name[30];// 注册用户名   
 char data[256]; //注册的用户密码  
   
 服务器实现(1)注册发送 do\_register();   
   
 //将客户端发送过来的用户名和密码去数据库中检验，若用户名未被注册，将用户名和密码保存到数据库文件中  
   
 //若用户名已经在数据库中被注册了，提示客户端需要重新注册  
   
   
 客户端发送 type == L 登陆请求 ---> 服务器端 收到siwtch(type) == L do\_login();  
 char name[30];// 登录用户名   
 char data[256]; //登录的用户密码  
   
 服务器实现(2)登录   
   
 //将客户端发送过来的用户名和密码去数据库中检验，若用户名和密码正确，提示客户端登录成功，客户端进入下一个界面  
   
 //若用户名和密码不正确，提示客户端登录失败，需要重新登录  
   
   
 客户端发送 type == F 查单词请求 ---> 服务器端 收到siwtch(type) == F do\_query();  
 char name[30];// 登录用户名   
 char data[256]; //要查询的单词cat  
   
 服务器实现((3)查询单词  
   
 //将客户端发送过来的单词去dict.txt文件中进行逐行比对，若查到该单词将单词及解释发送给客户端同时需要将  
   
 //用户名 和 查询单词 及 查询时间 保存到数据库文件中  
   
 //若查询到此词不存在，提示客户端单词不存在  
   
  
2.客户端  
  
  
 (1)菜单一，用来显示注册 登录　退出菜单界面  
  
   
 printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  
 printf("\* 电 子 词 典 项 目 \*");  
 printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  
 printf("\n\* 1.注册 2.登录 3.退出 \*\n");  
 printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  
 printf("\n请 输 入 您 的 选 择:\n\n");  
   
  
 (2)菜单二  
   
 菜单界面二框架：  
   
 printf("\n/////////////////////////////////////////////////////");  
 printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  
 printf("\* 欢　迎　使　用　电　子　词 　 \*");  
 printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  
 printf("\n\* 1.查询单词 2.查询历史单词 3.返回上一级 4.退出 \*\n");  
 printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  
 printf("/////////////////////////////////////////////////////\n");  
 printf("\n请 输 入 您 的 选 择:\n\n");  
   
   
3.服务器端  
  
 (1)数据库   
   
 表1 用户名和密码表 (tbl\_user 字段 username, password)  
   
 //tbl\_user 表名   
 //username 用户名   
 //password 用户密码   
   
 表2 历史信息表，(tbl\_history 字段 username, date\_time, his\_word查功能)  
   
 //tbl\_history 表名  
 //username 用户名   
 //data\_time 查询时间   
 //his\_word 历史单词  
   
 (2) 服务器端需要采用多线程并发服务器   
   
4.相关函数功能介绍   
  
 (1)文件I/O(标准I/O linux 文件I/O)  
  
 一次读一行  
  
 #include <stdio.h>  
 int main()  
 {   
 char buf[500] = { 0 };  
 FILE \*fp = fopen("dict.txt", "r");   
 while(fgets(buf, sizeof(buf), fp) != NULL)  
 {  
 printf("%s", buf);  
 }  
 }  
   
 //cat strcmp();  
 (2)比较单词(strncmp) //只比较两个字符串的前n个字符  
  
 #include <stdio.h>  
 #include <string.h>  
  
 int main()  
 {   
 char word[20] = { 0 }; //"cat"  
 char buf[500] = { 0 }; // cat cat单词解释  
 //strcmp(word,buf); // "cat" "cat cat单词解释"  
 printf("请您输入您要查找的单词:\n");  
 scanf("%s",word);  
 FILE \*fp = fopen("dict.txt", "r");   
 while(fgets(buf, sizeof(buf), fp) != NULL) //一次读取一行数据  
 {  
 if(strncmp(word,buf,strlen(word)) == 0) //只比较字符串的前n个字符  
 {  
 //cat   
 //cat cat单词解释   
 //strlen(word) == 3 //3 之比较两个字符串的前3个字符  
  
 printf("%s\n",buf);  
 break;  
 }  
 }  
 }

# 三. 项目实现

# my.h

# server.c

# client.c