

**北 京 科 技 大 学**

**计算机网络课程设计报告**

学　　院： \_\_\_\_\_\_\_计通学院\_\_\_\_\_\_\_\_\_

班　　级： \_\_\_\_\_\_\_计172\_\_\_\_\_\_\_\_\_\_\_\_

曹璐然 郭真铃 金玉卿 艾丽菲热·达吾提

姓　　名： \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

41724059 41724057 41724051 41724035

学　　号： \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

成　　绩： \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

指导教师签字： \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**2020 年 5 月**

**北京科技大学实验报告**

学院：计通学院 专业：计算机与通信工程 班级：计172

曹璐然，郭真铃

金玉卿，艾丽菲热·达吾提

2020 4

姓名： 学号： 实验日期： 年 月

41724059, 41724057

41724051,41724035

**实验名称：**计算机网络课程设计

**实验目的：**

将书本上抽象的概念与具体实现技术结合，通过网络软件编程的实践，深入理解理论课上学习到的计算机网络基本原理和重要协议，通过自己动手编程封装与发送数据分组，加深对网络协议的理解，掌握协议数据单元的结构和协议工作原理及其对协议栈的贡献。

**实验仪器：**

* 实验硬件设备：win10电脑三台，连接到Internet
* 实验软件要求：（使用的操作系统、开发工具和编程语言）

操作系统：win10

开发工具：WinPcap、VistualStudio 2017、WireShark

编程语言：C、C++

* 小组成员及分工：

曹璐然，负责视频处理、实验一、文档

郭真铃，负责实验一、实验二、文档

金玉卿，负责录屏、实验一、文档

艾丽菲热·达吾提，负责实验一、录屏、文档

**实验原理：**

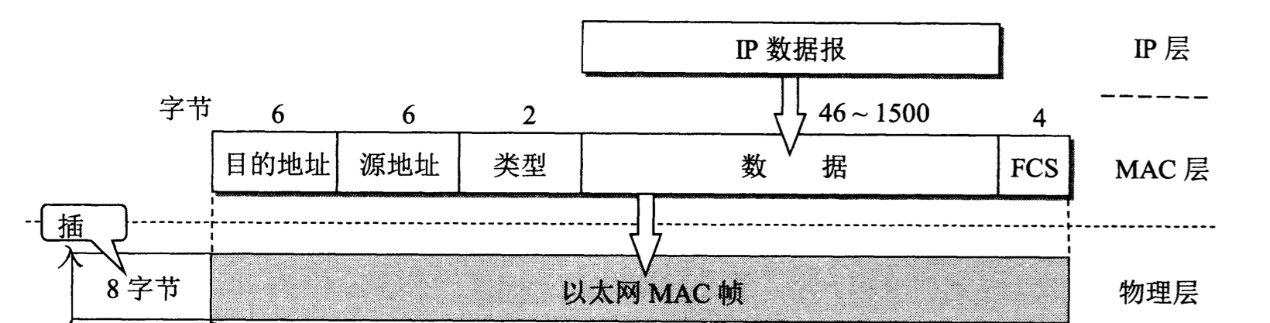
（1）题目1实验原理：

数据包的封装发送和解析（ARP/ICMP/TCP），网络协议栈的多种协议都有自己的功能，协议包括语义、语法和同步三个要素，不同的网络协议其分组的首部格式不同，必须按照协议规定的格式封装（发送）和理解（接收）数据分组首部，才能使得不同站点的计算机按照规定的方式相互通信。ARP协议是IP地址和MAC地址解析协议；ICMP是控制IP数据包传递的协议；TCP是面向连接的可靠的传输层协议，它们均有自己固定的分组首部格式。

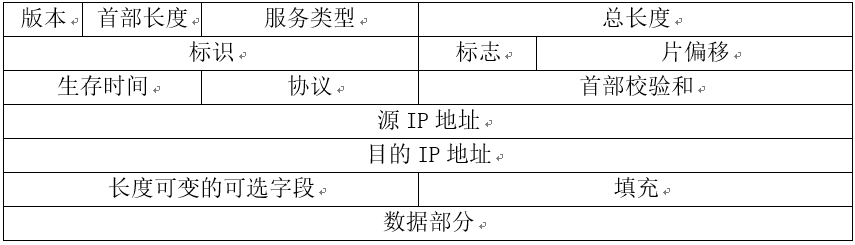
本实验中使用WinPcap技术或Socket技术，根据ARP/ICMP/TCP协议数据单元的结构和封装规则，封装数据帧发送到局域网中。另外要捕获网络中的ARP/ICMP/TCP数据包，解析数据包的内容，并显示结果，同时写入日志文件。

需要封装和解析的格式如下：

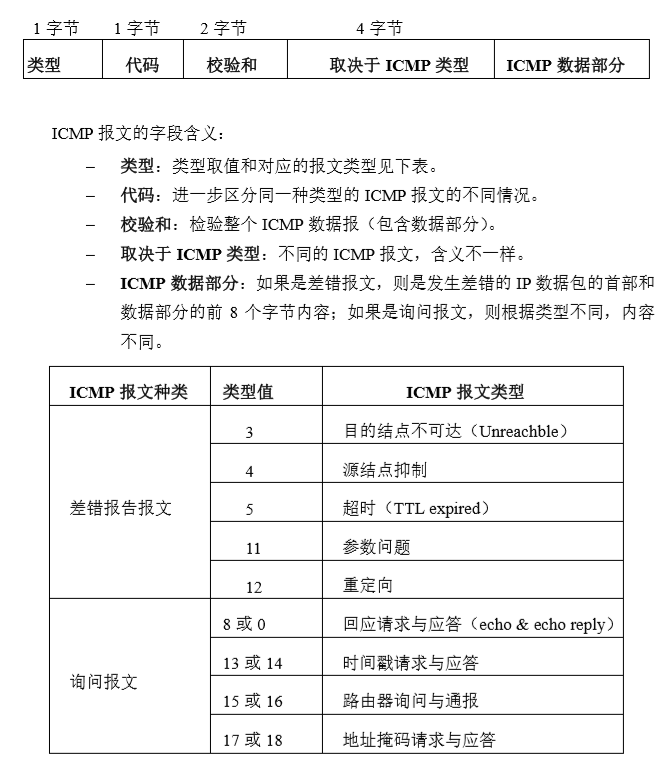
MAC帧：



IP数据报：



ICMP格式



（2）题目2实验原理：

实验二是应用网络开发技术开发网络小应用程序。要求网络应程序必须是基于网络应用层协议开发（也可直接基于IP协议或UDP协议），必须采用网络低层开发技术，不能直接调用高级语言封装好的插件等。因此本组选择的是使用Winsock来实现子网内的文件传送。该实验要求设计并实现一个局域网内部的文件传送工具，使用TCP协议进行可靠文字传输。由于需要发现局域网内其它“上线”的电脑，因此需要在用到UDP协议进行广播发送“上线”通知。

**实验内容与步骤：**

**题目1：数据分组的发送和解析（ARP/ICMP/TCP）**

（1）需求分析

使用WinPcap技术根据ICMP协议数据单元的结构和封装规则，封装数据帧发送到局域网中，并捕获网络中的ICMP数据包，解析数据包的内容，显示结果，同时写入日志文件。

（2）设计方案

1）程序功能模块划分

将ICMP数据包的发送和解析分成两部分，即receive\_icmp负责完成数据包的解析，sent\_icmp负责完成数据包的发送。

a）receive\_icmp：

将ICMP数据包的解析分成三部分，第一部分main函数完成准备工作，获得设备列表并打印网络适配器列表，设置过滤器并释放设备列表，调用第二部分；第二部分packet\_handler函数为回调函数，当收到每一个数据包时会被libpcap所调用，打印数据包的时间戳和长度，IP数据包头部，IP地址及tcp端口，并调用第三部分；第三部分write\_log函数主要完成将结果写入日志文件的作用，首先在当前程序的目录下创建和追加用于读写的空的日志文件Receive.log，对协议进行拆分，判断协议的类型进行打印。

b）sent\_icmp：

将ICMP数据包的发送分成，第一部分main函数做准备工作，获得设备列表并打印网络适配器列表，调用ICMP数据包的封装函数，最后发送数据包；第二部分check函数完成对ICMP数据包的封装，主要包括ICMP数据包的封装。ICMP报文格式如下：

0 1 2 3

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Type | Code | Checksum |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| unused |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Internet Header + 64 bits of Original Data Datagram |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

struct icmphdr

{

u\_int8\_t type; /\* message type \*/

u\_int8\_t code; /\* type sub-code \*/

u\_int16\_t checksum;

union

{

struct

{

u\_int16\_t id;

u\_int16\_t sequence;

} echo; /\* echo datagram \*/

u\_int32\_t gateway; /\* gateway address \*/

struct

{

u\_int16\_t \_\_unused;

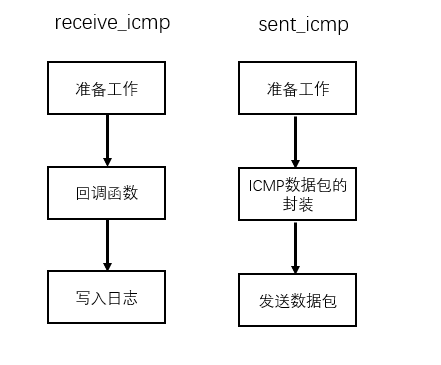
u\_int16\_t mtu;

} frag; /\* path mtu discovery \*/

} un;

};

2）模块间联系



3）数据结构

// 4 字节的 ip 地址

typedef struct ip\_address {

u\_char byte1;

u\_char byte2;

u\_char byte3;

u\_char byte4;

}ip\_address;

// ipv4 首部

typedef struct ip\_header {

u\_char ver\_ihl; // 版本 (4 bits) + 首部长度 (4 bits)

u\_char tos; // 服务类型(8 bits)

u\_short tlen; // 总长(16 bits)

u\_short identification; // 标识(16 bits)

u\_short flags\_fo; // 标志位(3 bits) + 段偏移量 (13 bits)

u\_char ttl; // 存活时间(8 bits)

u\_char proto; // 协议(8 bits)

u\_short crc; // 首部校验和(16 bits)

ip\_address saddr; // 源地址(32 bits)

ip\_address daddr; // 目的地址(32 bits)

u\_int op\_pad; // 选项与填充(32 bits)

};

// icmp的头部

typedef struct icmp\_header

{

u\_char type; // 类型(8 bit)

u\_char code; //代码(8 bit)

u\_short checksum; //校验和(16 bits)

u\_short id; // 标识符 (16 bits)

u\_short seq; // 序列号 (16 bits)

};

// mac地址

typedef struct mac\_address {

u\_char byte1;

u\_char byte2;

u\_char byte3;

u\_char byte4;

u\_char byte5;

u\_char byte6;

};

// mac头部

typedef struct MAC\_header {

mac\_address daddr;// 目的地址(48 bits)

mac\_address saddr; // 源地址(48 bits)

u\_short type; //类型（16 bit）

};

char errbuf[PCAP\_ERRBUF\_SIZE];

// ICMP数据包

u\_char packet[packet\_size];

type：一个 8 位类型字段，表示 ICMP 数据包类型;

code：一个 8 位代码域，表示指定类型中的一个功能，如果一个类型中只有一种功能，代码域置为 0;

checksum：数据包中 ICMP 部分上的一个 16 位检验和;

ICMP报文中没有PORT，其是根据id+sequence来进行目标判断的。

4）关键问题解决方法

①如何计算IP数据报首部校验和以及ICMP的校验和？

初始计算校验和字段时该字段全部用0填充，IP头部以16位为一个单位，逐个模2加（相当于异或），将得到的结果取反，作为校验和放入校验和字段。注意：IP数据报的检验和是只计算IP数据报首部，而ICMP的校验和是计算ICMP首部和数据

void check(u\_char\* packet, int len, u\_char\* check\_num) {

unsigned long check\_sum = 0;

for (int k = 0; k < len - 1;) {

unsigned long tmp = (packet[k] << 8) + packet[k + 1];

check\_sum += tmp;

k = k + 2;

}

//处理奇数字节情况

if (len % 2 == 0) {

check\_sum += (packet[len - 1] << 8);

}

//处理进位部分

check\_sum = (check\_sum >> 16) + (check\_sum & 0xffff);

check\_sum += (check\_sum >> 16);

//取反 赋值

check\_num[0] = ~(u\_char)(check\_sum >> 8);

check\_num[1] = ~(u\_char)(check\_sum);

}

②如何实现命令行输入目的ip地址？

用字符数组tDIPA存储输入的ip地址，并按照”.”来划分ip地址的四个部分赋值给字符数组DIPA，注意考虑输入格式不正确的情况。

printf("Please Enter the destination IP address: ");

u\_char DIPA[4] = { 0 };

char tDIPA[16];

scanf\_s("%s", tDIPA, 16);

int iip = 0;

for (int i = 0; i < 16; i++)

{

if (tDIPA[i] == '\0')

break;

if (tDIPA[i] == '.')

iip++;

else

{

DIPA[iip] \*= 10;

DIPA[iip] += tDIPA[i] - 48;

}

if (iip == 4)

{

printf("Wrong ip, too many number");

break;

}

}

if (iip < 3)

{

printf("Wrong ip, the ip is short");

}

else

{

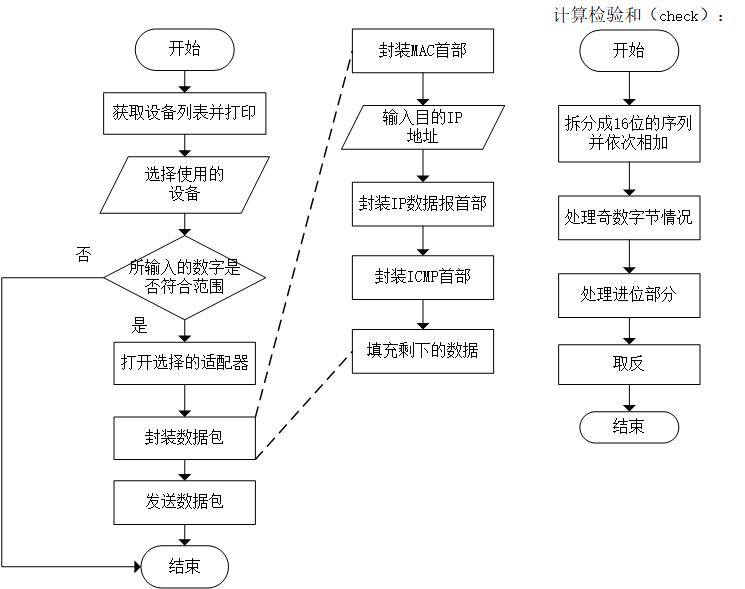
for (int i = 0; i < 4; i++)

packet[30 + i] = DIPA[i];

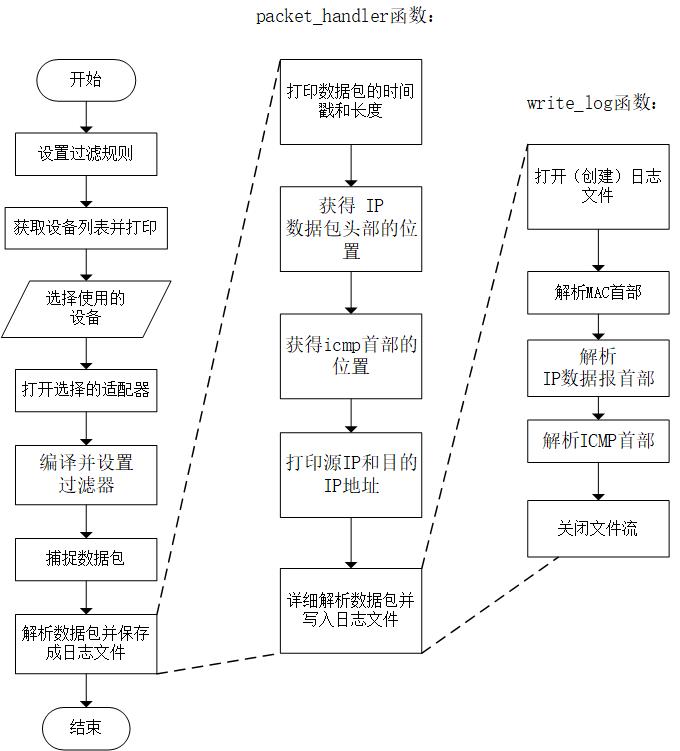
}

（3）程序结构和流程图

sent\_icmp：



receive\_icmp：



（4）测试方案（程序清单）

①发送至自己ICMP数据包

②发送至他人ICMP数据包

**题目2：子网内文件传输**

我们实现了两个版本，第一个为简易版本，虽然功能简单，但我们认为使用起来十分的便捷，灵活度高，顾简述一下设计思路：

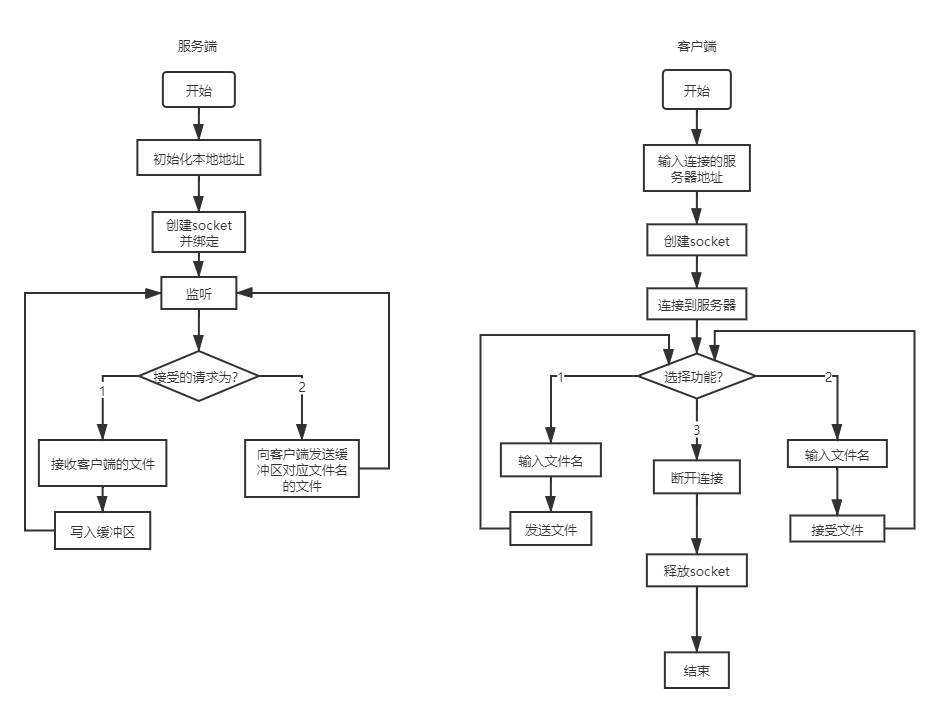
设计一个服务端和一个客户端。

服务端：首先创建socket并连接。连接后不断监听有无客户端发来发送文件或下载文件的请求。当有发送文件来时，将文件保存到缓冲区中，通过文件名进行区分。当有客户端想要下载文件时，将对应文件名的文件打成数据包并发送。

客户端：连接到服务器端，可以选择发送文件或接收文件。发送文件则输入文件名，找到相应文件后打包发到服务器端的缓冲区。下载文件则发给服务器端文件名，服务器会回复含有文件的数据包，然后将文件写到本地。

此版本任何用户都可以从服务器获取要下载的文件，并不受限于发送者发给几个用户，对于接收方和发送方来说功能平等，使用灵活。

其流程图如下：



此版本我们也将在视频中有所体现。

下面详述版本二：

（1）需求分析

①获取当前在线的用户

②添加用户为好友

③删除好友

③选择发送文件的好友

④进行文件传输

⑤用户下线

（2）设计方案

1.程序整体分为以下几个模块：

main模块：执行初始化，发送udp广播告知其他用户已上线，启动监听线程，程序功能选择功能。

子模块：

①add：添加想要传送文件的ip地址到好友列表数组中。

本模块将要添加的ip地址作为参数，判断其不在好友列表容器中后，将其添加进去。

②del：将已下线的ip从好友列表删除。

本模块将要删除的ip地址作为参数，判断其在好友列表容器中后，将其删除。

③sendOnLine：向发送udp的客户端发起TCP请求，回应上线消息。

创建socket后与服务器进行连接，给监听时接收到的别的客户端的上线信息发出一个TCP请求，则刚上线的客户端可知道当前有哪些用户在线。

④sendUnLine：发送下线消息。

下线时为了让其他用户不再给自己发文件，下线时向当前好友列表中的所有IP地址依次发起TCP连接并发送下线消息。

⑤sentfile：打开并发送文件。

参数为发送对象的ip和文件名。创建socket并连接服务器后，发送将要发送文件的label。然后打开文件并向服务器发送文件至缓冲区。

⑥serviceTCP：监听发送方端口的消息，接收发来的文件。

创建一个本地服务端并连接，然后循环监听是否有数据送达。当传来的是用户状态label时，根据状态信息利用add或del函数对好友列表进行操作。实现自动更新好友列表。当传来文件时，将其从缓冲区下载到本地，并清除缓冲区，实现文件自动下载。

⑦initNetwork：初始化socket。

⑧serverudp：监听广播udp

创建socket，监听客户端的udp，并调用add模块将已上线的用户ip加入设备列表，并使用sendOnLine发起TCP请求，以便告知发送广播udp的发送方，自己用户的IP地址。

⑨clientudp：发送udp

用户上线时，主模块调用此函数通过向子网内用户发送udp多播告知其他用户，并等待其回复。或在更新好友列表时调用此函数。

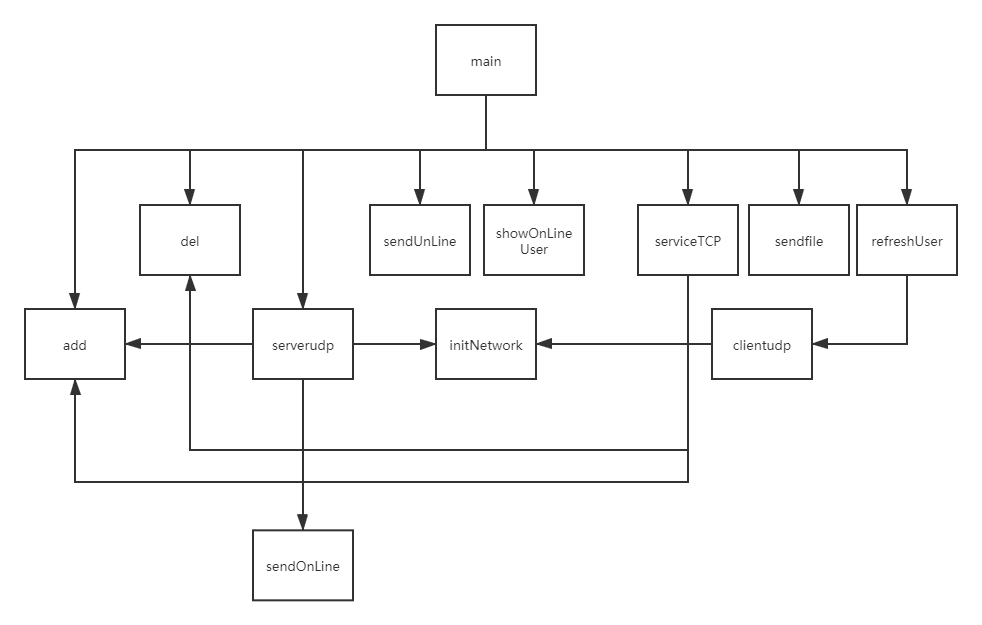
⑩refreshUser：刷新用户。

更新好友列表机调用clientudp模块。

①①showOnLineUser：打印在线好友列表。

循环打印出好友列表容器中的ip。

2.模块间的调用关系：



3.数据结构：

好友列表信息存放在容器中：

vector<char\*> onlineList;//在线好友列表

文件名采用char数组存放：

char file\_name[FILE\_NAME\_MAX\_SIZE + 1];

缓冲区采用char数组存放：

char buffer[BUFFER\_SIZE];

设定消息类型：

#define OnLine '1'//告知他人当前机子上线了

#define UnLine '2'//告知他人当前机子下线了

#define SentFile '3'//发送文件

4.关键问题解决办法：

①如何解决好友列表刷新问题

客户端上线的时候会通过UDP协议广播发送一个上线消息，当其它客户端收到该消息后，会先将该UDP的发送方的IP地址加入到好友列表中，然后发送一个TCP请求连接给发送方，向发送方发送一个上线消息，发送方接收到TCP请求和上线消息后，会将接收方的IP地址加入到好友列表中。这样便解决了上线通知的问题。

当客户端下线时，会向当前好友列表中的所有IP地址依次发起TCP连接并发送下线消息，收到该消息的客户端会将下线客户端的IP地址从当前好友列表中删除。

void clientudp(char\* IP,char sendLabel) {

initNetwork();

SOCKET udp\_client\_socket = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

if (udp\_client\_socket == SOCKET\_ERROR) {

std::cout << "Failed to create socket!" << std::endl;

return ;

}

bool so\_broadcast = true;

int ret = setsockopt(udp\_client\_socket, SOL\_SOCKET, SO\_BROADCAST, (char\*)&so\_broadcast, sizeof(so\_broadcast));

if (ret < 0)

{

printf("无法把套接字设为广播类型\n");

return;

}

sockaddr\_in brocast;

char buf[BUFFER\_SIZE] = { 0 };

brocast.sin\_family = AF\_INET;

brocast.sin\_addr.s\_addr = INADDR\_BROADCAST;//广播地址

brocast.sin\_port = htons(broRevPort);

memset(buf, 0, BUFFER\_SIZE);

buf[0] = sendLabel;

ret = sendto(udp\_client\_socket, buf, strlen(buf), 0, (sockaddr\*)&brocast, sizeof(sockaddr));

if (ret == SOCKET\_ERROR) {

std::cout << "Failed to send!" << std::endl;

return ;

}

closesocket(udp\_client\_socket);

WSACleanup();

return ;

}

void sendUnLine(char\* sIP) {

SOCKET c\_Socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (SOCKET\_ERROR == c\_Socket)

{

printf("# 创建sock失败！ #\n");

return;

}

for (int i = 0; i < onlineList.size(); i++) {

// 指定服务端的地址

sockaddr\_in server\_addr;

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.S\_un.S\_addr = inet\_addr(onlineList[i]);

server\_addr.sin\_port = htons(PORT);

// 建立连接

if (SOCKET\_ERROR == connect(c\_Socket, (LPSOCKADDR)&server\_addr, sizeof(server\_addr)))

{

printf("# 无法连接到服务器！ #\n");

return;

}

printf("# 成功连接到服务器！ #\n");

//发送选项

char bufferSendLabel[BUFFER\_SIZE] = { 0 };

memset(bufferSendLabel, 0, BUFFER\_SIZE);

bufferSendLabel[0] = UnLine;

bufferSendLabel[1] = '\0';

if (send(c\_Socket, bufferSendLabel, BUFFER\_SIZE, 0) < 0)

{

printf("# 发送下线label失败！ #\n");

return;

}

}

closesocket(c\_Socket);

return;

}

②如何解决手动添加（删除）好友

查找当前好友列表中是否存在（不存在）该好友，若存在（不存在）则输出“当前好友列表中已经存在该IP了”（“当前好友列表中没有该IP”）；若不存在（存在）则向好友列表中添加（删除）好友并打印出“添加成功”（“删除成功”）的信息。

void add(char\* addr) {

bool f = false;

//查看当前要加入的IP地址是否已经存在

//此处不能用自带的count函数！这样匹配的是char\*的地址

for (int i = 0; i < onlineList.size(); i++) {

if (\*addr == \*onlineList[i]) {

f = true;

break;

}

}

if (!f) {

onlineList.push\_back(addr);

printf("添加成功!\n");

}

else {

printf("当前好友列表中已经存在该IP了\n");

}

return;

}

//删除列表中的下线好友

void del(char\* addr) {

bool f = false;

//查看当前要加入的IP地址是否已经存在

//此处不能用自带的count函数！这样匹配的是char\*的地址

int i = 0;

for (; i < onlineList.size(); i++) {

if (\*addr == \*onlineList[i]) {

f = true;

break;

}

}

if (f) {

onlineList.erase(onlineList.begin() + i);

printf("删除成功！\n");

}

else {

printf("当前好友列表中没有该IP\n");

}

return;

}

③如何传输文件

在已经建立连接的基础上，创建socket，并且与服务端地址绑定，开始监听来自客户端的消息，当接收到是“发送文件”的消息时，开始准备接收文件名，根据文件名接收方在程序目录下创建同名文件并打开写操作流以便接收文件，接着传输文件内容。

else if (f == 3) {

memset(bufferTCPServer, 0, BUFFER\_SIZE);

recv(m\_New\_Socket, bufferTCPServer, BUFFER\_SIZE, 0);

if (bufferTCPServer[0] == '0') {

printf("客户端上没找到文件!\n");

}

else {

memset(bufferTCPServer, 0, BUFFER\_SIZE);

if (recv(m\_New\_Socket, bufferTCPServer, BUFFER\_SIZE, 0) < 0)

{

printf("服务器接收文件名失败!\n");

break;

}

char file\_name[FILE\_NAME\_MAX\_SIZE + 1];

memset(file\_name, 0, FILE\_NAME\_MAX\_SIZE + 1);

strncpy(file\_name, bufferTCPServer, strlen(bufferTCPServer) > FILE\_NAME\_MAX\_SIZE ? FILE\_NAME\_MAX\_SIZE : strlen(bufferTCPServer));

printf("%s\n", file\_name);

printf("开始接收客户端发送过来的文件%s\n", file\_name);

FILE\* fp = fopen(file\_name, "wb");

if (NULL == fp)

{

printf("无法写入文件名为 %s的文件\n", file\_name);

}

else

{

memset(bufferTCPServer, 0, BUFFER\_SIZE);

int length = 0;

while ((length = recv(m\_New\_Socket, bufferTCPServer, BUFFER\_SIZE, 0)) > 0)

{

if (fwrite(bufferTCPServer, sizeof(char), length, fp) < length)

{

printf("文件名为%s的文件写入失败！\n", file\_name);

break;

}

memset(bufferTCPServer, 0, BUFFER\_SIZE);

}

fclose(fp);

printf("成功接收来自客户端的%s文件!\n", file\_name);

}

closesocket(m\_New\_Socket);

}

}

}

④如何解决缓冲区“不干净”问题

在写入缓冲区前使用memset清空缓冲区。

//清空缓冲区

memset(bufferTCPServer, 0, BUFFER\_SIZE);

⑤如何解决一个程序里即发送又接收的问题

使用多线程，在程序启动时分别启动udp接收线程和TCP接收线程。UDP接收线程用于接收广播消息，TCP接收线程用于接收文件的接收、上线回应、下线消息。发送文件和消息启用另外的线程实现。

thread ts(serviceTCP,sIP);

ts.detach();

thread us(serverudp, sIP);

us.detach();

⑥如何解决判别出收到的消息时什么类型的消息（发送文件、用户上线、用户下线）

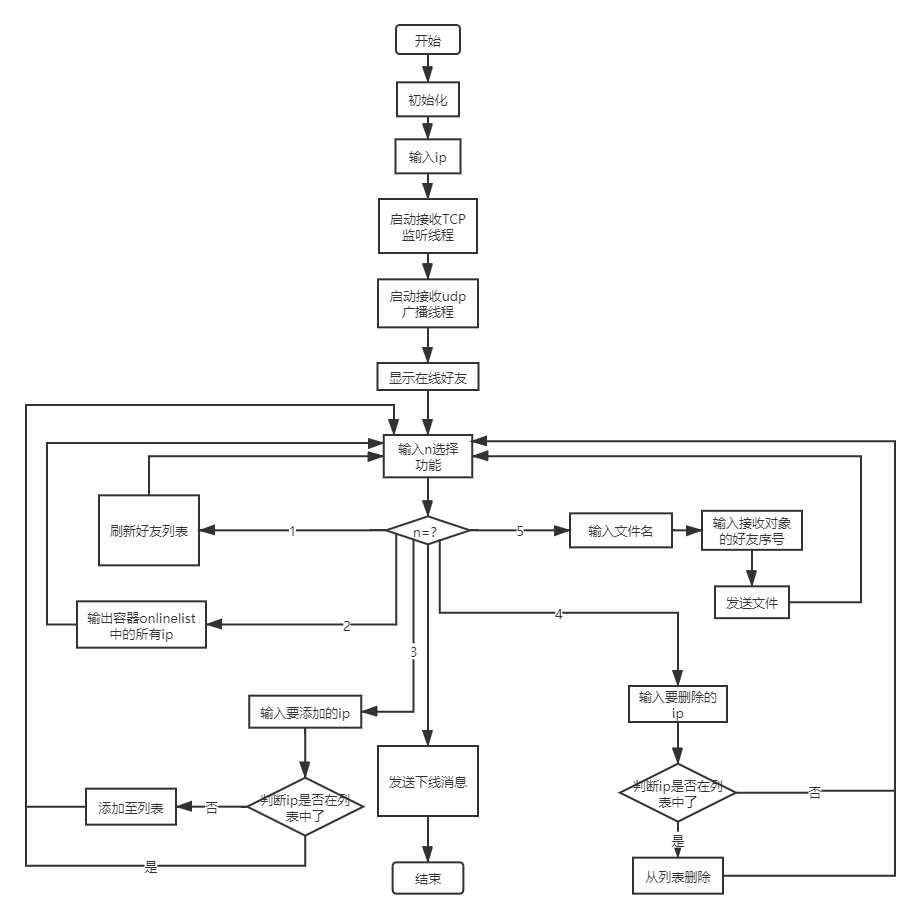
设定全局的常量，并设定发送和接收数据的第一位标明消息类型。根据消息类型给出相应的操作。

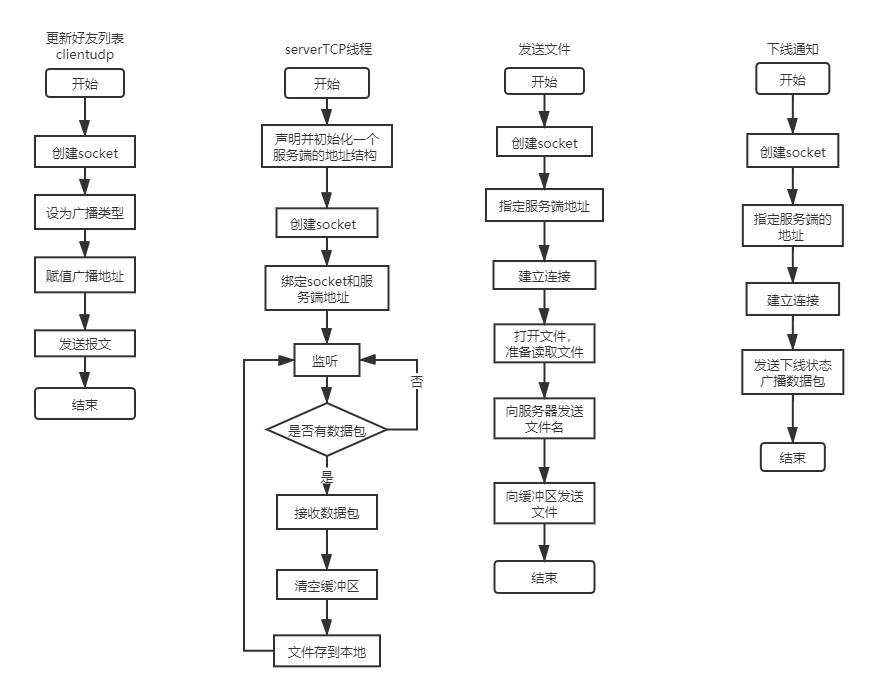
#define OnLine '1'//告知他人当前机子上线了

#define UnLine '2'//告知他人当前机子下线了

#define SentFile '3'//发送文件

（3）程序结构和流程图





（4）测试方案

①自己给自己发送文件

②给他人发送文件

③手动添加和删除好友功能实现

④自动刷新好友列表

**实验数据：**

**题目1：数据分组的发送和解析（ARP/ICMP/TCP）**

a）输入数据

receive\_icmp：

Enter the interface number (1-4)：2

sent\_icmp：

Enter the interface number (1-4)：2

Please Enter the destination IP address:192.168.0.100

b）预期输出数据

xx:xx:xx.xxxxxx len:100 192.168.0.100 -> 192.168.0.100

**题目2：子网内的文件传送**

1. 输入数据

①指定地址：

#-----------------请指定你的IP地址---------------------------#

192.168.0.109

②给自己发送文件

5

# 请输入要传送的文件名 #

test1.txt

#-----------请输入要发送文件的好友序号（以0结束）------#

1 0

③给他人发送文件

5

# 请输入要传送的文件名 #

test2.txt

#-----------请输入要发送文件的好友序号（以0结束）------#

1 0

④删除好友

4

#-------------请输入要删除的好友IP-----------------------#

192.168.0.109

⑤添加好友

3

#----------------请输入要添加的好友IP-----------------------#

192.168.0.109

⑥刷新好友列表

1

2

1. 预期输出数据

①给自己发送文件

# 成功连接到服务器！ #

test1.txt

开始接收客户端发送过来的文件test1.txt

# 发送文件test1.txt 成功! #

成功接收来自客户端的test1.txt文件!

②给他人发送文件

# 成功连接到服务器！ #

# 发送文件test2.txt 成功! #

③删除好友

删除成功！

④添加好友

添加成功!

⑤刷新好友列表

# 成功连接到服务器！ #

当前好友列表中已经存在该IP了（或者输出“添加成功”）

当前在线好友：

序号 IP地址

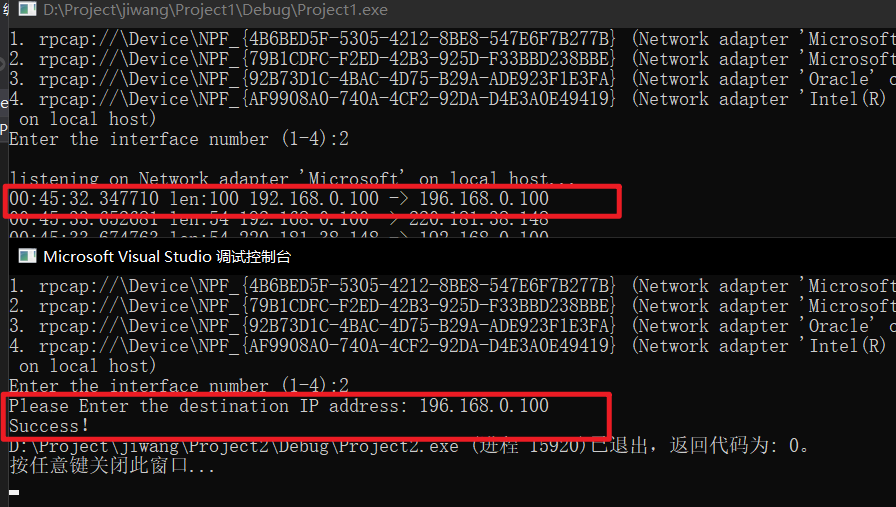
1 192.168.0.109

**实验结果与分析：**

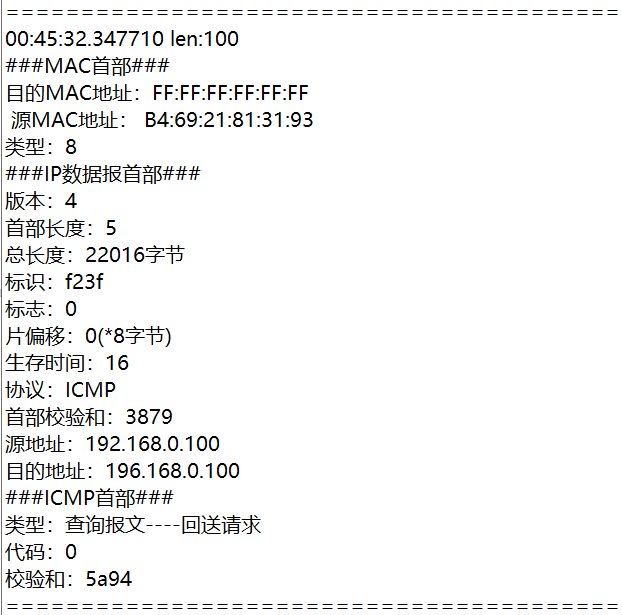
**（1）程序运行结果**

**题目1：数据分组的发送和解析（ARP/ICMP/TCP）**

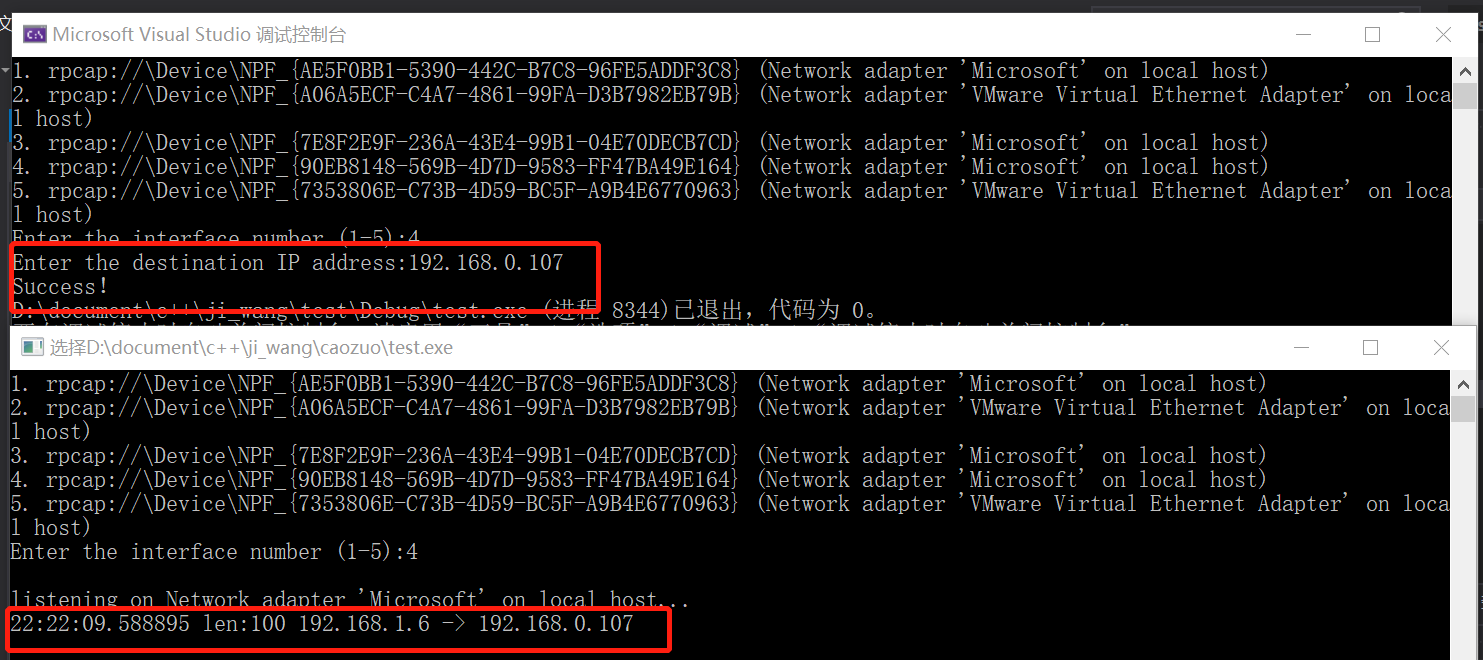
①发送给自己电脑icmp数据包



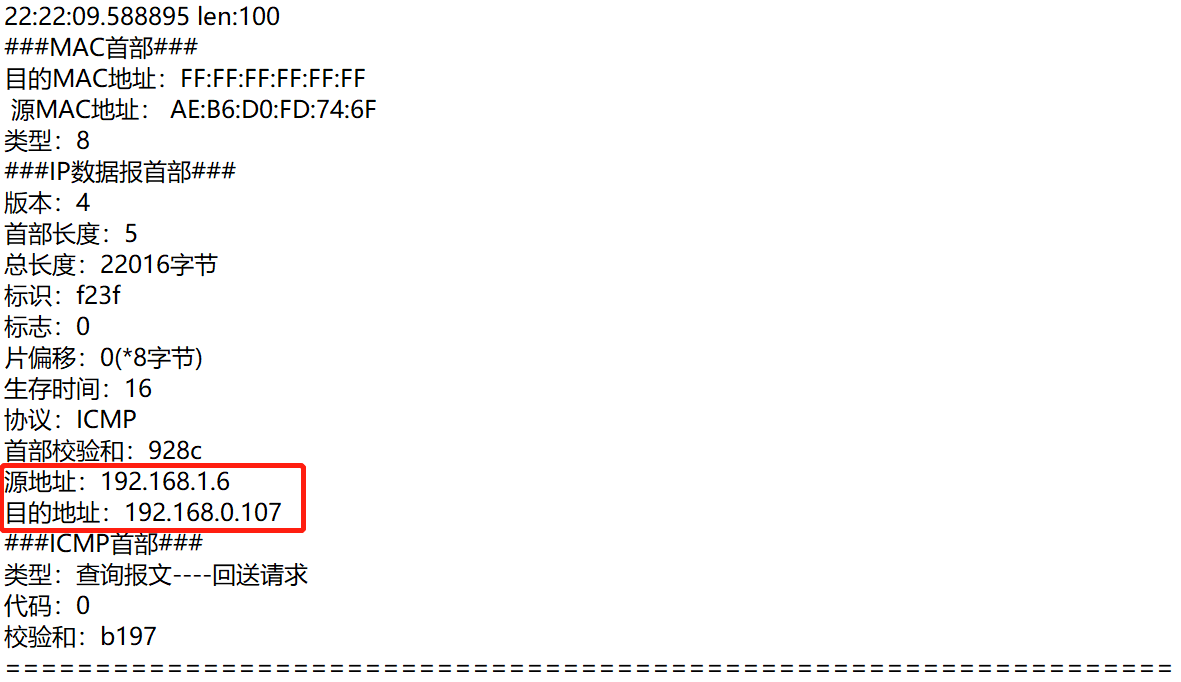
查看receive.log日志



②发送给其他用户icmp数据包

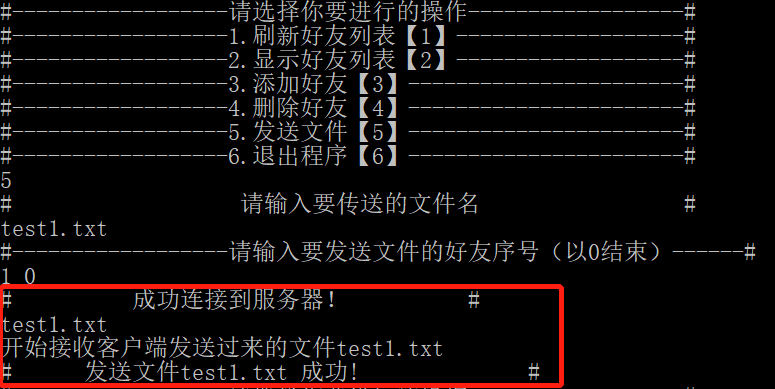


查看receive.log日志

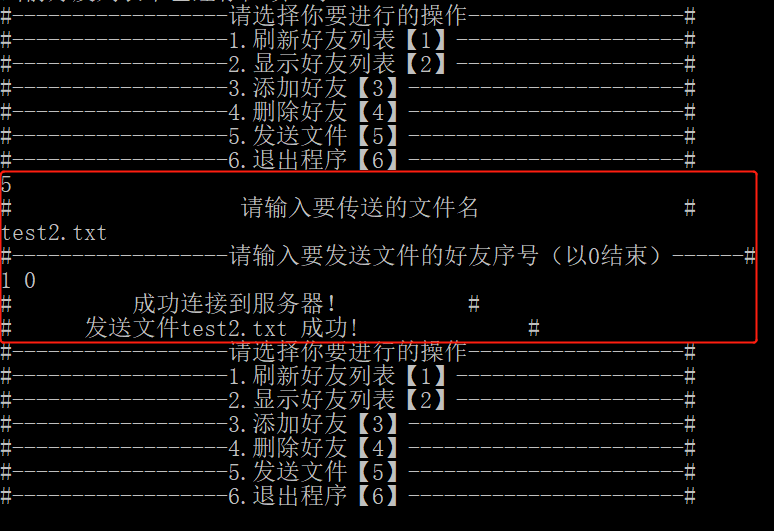


**题目2：子网内的文件传送**

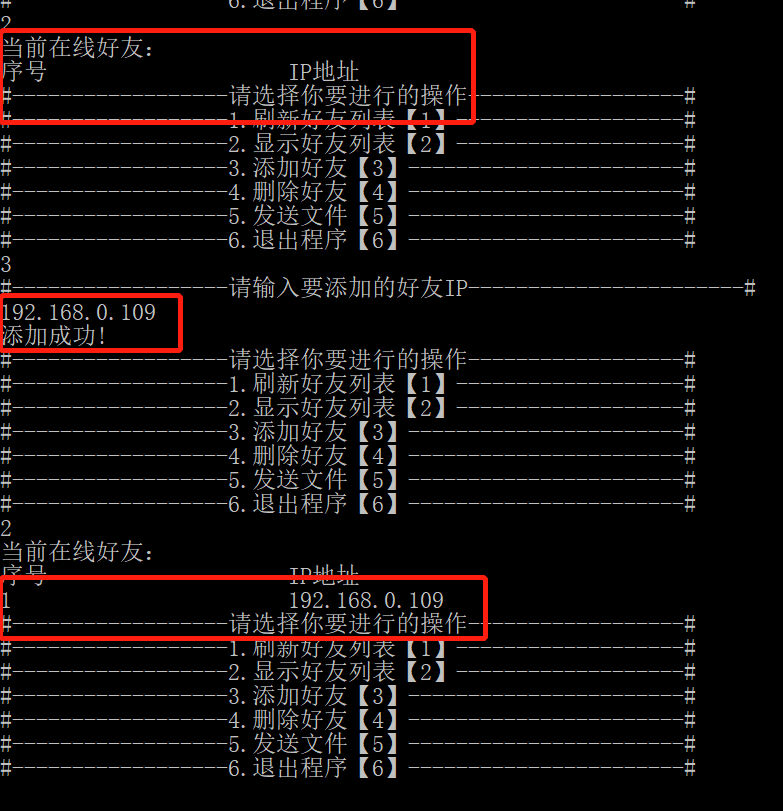
①自己给自己发送文件



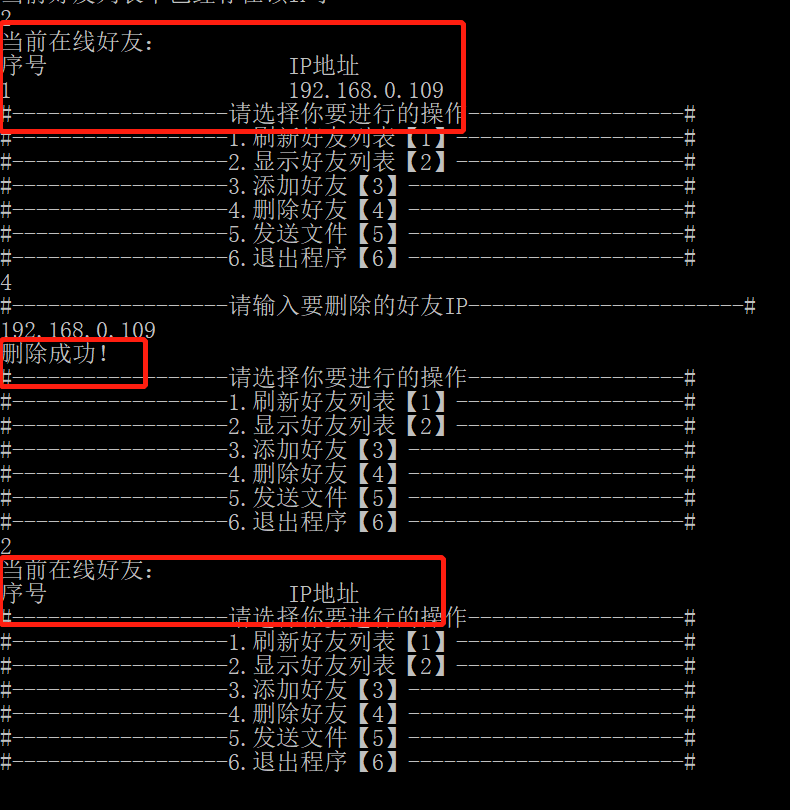
②给他人发送文件



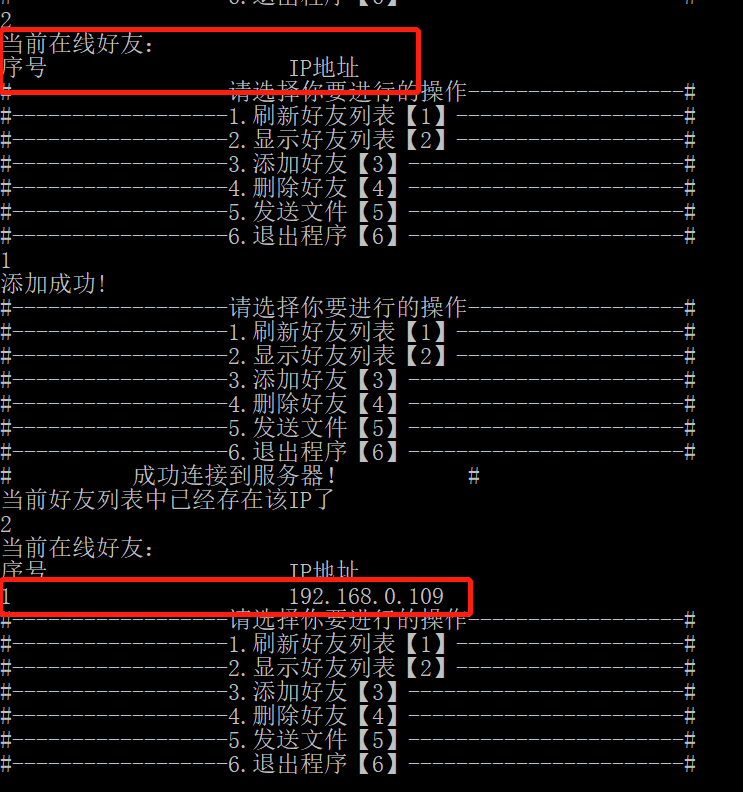
③手动添加好友



④删除好友



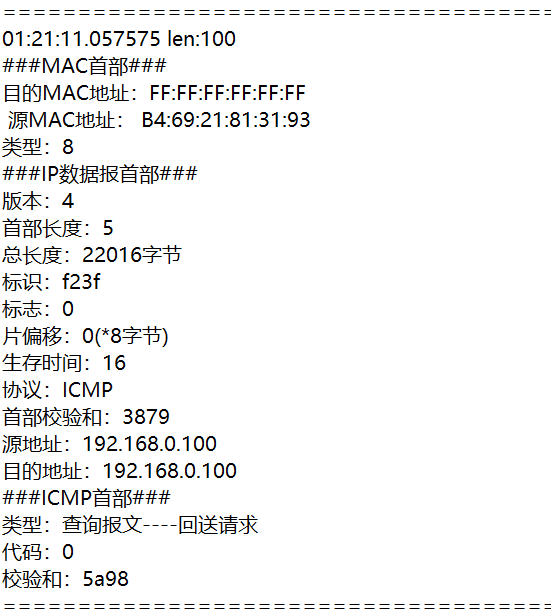
⑤自动刷新好友列表

****

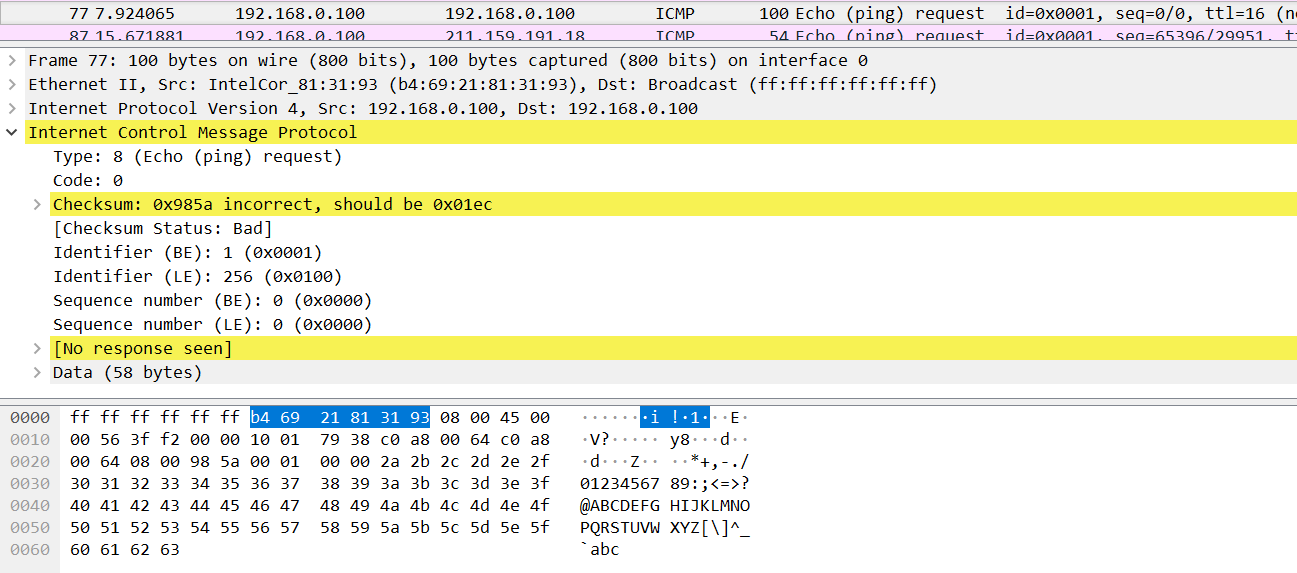
**（2）运行结果分析**

**题目1：数据分组的发送和解析（ARP/ICMP/TCP）**

由sent\_icmp代码中封装的ICMP数据报packet[N]可知，目的MAC地址设为全F，源MAC地址为B4-69-21-81-31-93，源ip地址为192.168.0.100，输入的目的ip地址为192.168.0.100。ICMP类型为8，IP数据报中IP版本为4，长度为5，标识等信息均与下图receive.log日志文件内容一致，可说明代码运行没问题，正常发出。



通过wireshark抓包软件进行核对本地计算机发出与收到的数据分组，如下图所示，可以看到，抓包获取的数据分组符合预期，说明可以捕获机器发出的数据分组。

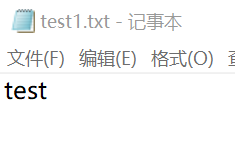


**题目2：子网内的文件传送**

①自己给自己发送文件

从结果上看可以发现文件成功传送到了当前程序的目录下，且打开文件后，文件内容和发送前一致。





②给他人发送文件

从结果上看，成功发送给其他人了，在接收方的目录下可以看见所发送的文件，打开后文件内容和发送方一致，可见发送成功。





③手动添加好友

从结果截图上看，对比添加前后打印出的好友列表，可以发现确实成功添加好友了。

④删除好友

从结果截图上看，对比删除前后打印出的好友列表，可以发现确实成功删除好友了。

⑤自动刷新好友列表

从结果截图上看，对比刷新好友列表前后打印出的好友列表，可以发现确实成功通过自动刷新功能，成功将处于同一个子网内运行该程序的好友添加到当前列表中了。

（3）实验结论

ICMP协议是网络层的重要协议，通过封装与发送这些数据包，加深了对网络协议的理解，掌握了ICMP协议数据结构和工作原理及其对协议栈的贡献。从安装WinPcap开始，一步步学习Pcap网络编程技术，根据ICMP协议数据的数据帧，解析协议数据的内容，并将结果显示，同时写入日志文件，最后开启Wireshark抓包软件，在标准输出中显示捕获ICMP报文的首部字段的内容，

通过本次的实验，对应用层有关网络编程有了初步的了解，掌握了使用Winsock实现UDP发送广播包、TCP传输文件。甚至还了解到了多线程的相关知识。总而言之，通过该课程，我们小组四人收获颇丰，网络相关编程能力有一定的增长。

（4）讨论

①可以尝试通过MFC来给程序添加一个比较漂亮的图形界面，目前的界面还处于相对“原始”的命令行的界面。未来若要改进，应该会使用MFC实现图形界面。

②可以让实验一的解析能解析更多类型的包，通过用户选择筛选条件，改变程序中的筛选条件和调用相应的解析程序，来实现解析不同的包。

**重要程序代码：**

**题目1：数据分组的发送和解析（ARP/ICMP/TCP）**

sent\_icmp.cpp：

#define WIN32

#include "pcap.h"

#include <stdlib.h>

#pragma comment ( lib, "wpcap.lib")

#include "stdio.h"

#pragma warning (disable: 4996)

#define packet\_size 100

#include<iostream>

void check(u\_char\* packet, int len, u\_char\* check\_num) {

unsigned long check\_sum = 0;

//每16位 相加

for (int k = 0; k < len-1;) {

unsigned long tmp = (packet[k] << 8) + packet[k + 1];

check\_sum += tmp;

k = k + 2;

}

//处理奇数字节情况

if (len % 2 == 0) {

check\_sum += (packet[len - 1] << 8);

}

//处理进位部分

check\_sum = (check\_sum >> 16) + (check\_sum & 0xffff);

check\_sum += (check\_sum >> 16);

//取反 赋值

check\_num[0] = ~(u\_char)(check\_sum >> 8);

check\_num[1] = ~(u\_char)(check\_sum);

}

int main()

{

pcap\_t\* fp;

char errbuf[PCAP\_ERRBUF\_SIZE];

u\_char packet[packet\_size];

int i;

int j = 0;

u\_char check\_num[2];//存放校验结果

pcap\_if\_t\* alldevs;

pcap\_if\_t\* d;

int inum;

char\* tmp = new char[strlen(PCAP\_SRC\_IF\_STRING) + 1];

strcpy(tmp, PCAP\_SRC\_IF\_STRING);

if (pcap\_findalldevs\_ex(tmp, NULL, &alldevs, errbuf) == -1)

// 获得设备列表

{

fprintf(stderr, "Error in pcap\_findalldevs: %s\n", errbuf);

exit(1);

}

for (d = alldevs; d; d = d->next)

{ // 打印网络适配器列表

printf("%d. %s", ++j, d->name);

if (d->description)

printf(" (%s)\n", d->description);

else

printf(" (No description available)\n");

}

if (j == 0)

{

printf("nNo interfaces found! Make sure WinPcap is installed.\n");

return -1;

}

printf("Enter the interface number (1-%d):", j);

scanf\_s("%d", &inum); //scanf C语言的标准输入函数

if (inum < 1 || inum > j) //判断输入是否正确 是否是1-i之间的数字

{

printf("nInterface number out of range.\n");

pcap\_freealldevs(alldevs); //释放设备列表

return -1;

}

//跳转到已选网络适配卡

for (d = alldevs, j = 0; j < inum - 1; d = d->next, j++);

if ((fp = pcap\_open(d->name, 65536, PCAP\_OPENFLAG\_PROMISCUOUS, 1000, NULL, errbuf)) == NULL)//打开网络适配卡

{

fprintf(stderr, "nUnable to open the adapter. %s is not supported by WinPcapn");

pcap\_freealldevs(alldevs); // 释放设备列表

return -1;

}

/\*-------------封装数据--------------------------\*/

/\* 设置MAC的目的地址为9E-B6-D0-FD-74-6F 网卡1\*/

packet[0] = 0xFF;

packet[1] = 0xFF;

packet[2] = 0xFF;

packet[3] = 0xFF;

packet[4] = 0xFF;

packet[5] = 0xFF;

/\* 设置MAC源地址为 AE-B6-D0-FD-74-6F 网卡2 \*/

packet[6] = 0xAE;

packet[7] = 0xB6;

packet[8] = 0xD0;

packet[9] = 0xFD;

packet[10] = 0x74;

packet[11] = 0x6F;

/\*指明上层用的是IP数据报\*/

packet[12] = 8;

packet[13] = 0;

/\*至此MAC帧头部封装完毕\*/

/\*IP数据报首部\*/

packet[14] = 0x45;//版本：4 长度：5

packet[15] = 0x00;//区分服务 不用

packet[16] = 0x00;//总长度：86字节 即0x0056

packet[17] = 0x56;

packet[18] = 0x3f;//标识：0x3ff2

packet[19] = 0xf2;

packet[20] = 0x00;//标志：0 这是数据报片的最后一个

packet[21] = 0x00;//片偏移：0

packet[22] = 0x10;//生存时间TTL

packet[23] = 0x01;//协议：ICMP 1

packet[24] = 0x00;//首部检验和：先置为0

packet[25] = 0x00;

//源地址 IP地址 192.168.1.6

packet[26] = 192;

packet[27] = 168;

packet[28] = 1;

packet[29] = 6;

//输入目的ip地址

printf("Enter the destination IP address:");

char tDIPA[16];

int iip = 0;

u\_char DIPA[4] = { 0,0,0,0 };

std::cin >> tDIPA;

for (int i = 0; i < 16; i++)

{

if (tDIPA[i] == '\0')

break;

if (tDIPA[i] == '.')

iip++;

else

{

DIPA[iip] \*= 10;

DIPA[iip] += tDIPA[i] - 48;

}

if (iip == 4)

{

printf("Wrong ip, too many number");

break;

}

}

if (iip < 3)

{

printf("Wrong ip, the ip is short");

return -1;

}

else

{

for (int i = 0; i < 4; i++)

packet[30 + i] = DIPA[i];

}

//计算首部校验和（IP数据报）

check(packet, 20, check\_num);

packet[24] = check\_num[0];

packet[25] = check\_num[1];

/\*ICMP首部\*/

packet[34] = 0x08;//类型：询问报文 回送请求

packet[35] = 0x00;//代码

packet[36] = 0; //校验和

packet[37] = 0;

packet[38] = 0; //标识

packet[39] = 1;

packet[40] = 0;//序列号

packet[41] = 0;

/\*填充剩下的内容\*/

for (i = 42; i < packet\_size; i++)

{

packet[i] = i % 256;

}

/\*计算ICMP的校验和 要包括数据在内\*/

check(packet, packet\_size - 34, check\_num);

packet[36] = check\_num[0];

packet[37] = check\_num[1];

/\* 发送数据包 \*/

if (pcap\_sendpacket(fp, packet, packet\_size) != 0)

{

fprintf(stderr, "\nError sending the packet: \n", pcap\_geterr(fp));

return 0;

}

printf("Success！");

return 0;

}

receive\_icmp.cpp：

#define WIN32

#include "pcap.h"

#pragma comment ( lib, "wpcap.lib")

#include "stdio.h"

#include "time.h"

#pragma warning (disable: 4996)

using namespace std;

typedef struct ip\_address { // 4 字节的 ip 地址

u\_char byte1;

u\_char byte2;

u\_char byte3;

u\_char byte4;

}ip\_address;

// ipv4 首部

typedef struct ip\_header {

u\_char ver\_ihl; // 版本 (4 bits) + 首部长度 (4 bits)

u\_char tos; // 服务类型(8 bits)

u\_short tlen; // 总长(16 bits)

u\_short identification; // 标识(16 bits)

u\_short flags\_fo; // 标志位(3 bits) + 段偏移量 (13 bits)

u\_char ttl; // 存活时间(8 bits)

u\_char proto; // 协议(8 bits)

u\_short crc; // 首部校验和(16 bits)

ip\_address saddr; // 源地址(32 bits)

ip\_address daddr; // 目的地址(32 bits)

u\_int op\_pad; // 选项与填充(32 bits)

};

typedef struct icmp\_header

{

u\_char type; // 类型(8 bit)

u\_char code; //代码(8 bit)

u\_short checksum; //校验和(16 bits)

u\_short id; // 标识符 (16 bits)

u\_short seq; // 序列号 (16 bits)

};

typedef struct mac\_address {

u\_char byte1;

u\_char byte2;

u\_char byte3;

u\_char byte4;

u\_char byte5;

u\_char byte6;

};

typedef struct MAC\_header {

mac\_address daddr;// 目的地址(48 bits)

mac\_address saddr ; // 源地址(48 bits)

u\_short type; //类型（16 bit）

};

void packet\_handler(u\_char\* param, const struct pcap\_pkthdr\* header, const u\_char\* pkt\_data); // 回调函数原型

void write\_log(char\* timestr, const struct pcap\_pkthdr\* header, MAC\_header\* mah, ip\_header\* ih, icmp\_header\* ich);//写入日志文件

// 主函数

int main()

{

pcap\_if\_t\* alldevs;

pcap\_if\_t\* d;

int inum;

int i = 0;

pcap\_t\* adhandle;

char errbuf[PCAP\_ERRBUF\_SIZE];

u\_int netmask;

char packet\_filter[] = "icmp and ip"; //packet\_filter：过滤规则表达式

struct bpf\_program fcode;

;

//将PCAP\_SRC\_IF\_STRING转换为char\*

char\* tmp = new char[strlen(PCAP\_SRC\_IF\_STRING) + 1];

strcpy(tmp, PCAP\_SRC\_IF\_STRING);

//

if (pcap\_findalldevs\_ex(tmp, NULL, &alldevs, errbuf) == -1)

// 获得设备列表

{

fprintf(stderr, "Error in pcap\_findalldevs: %s\n", errbuf);

exit(1);

}

for (d = alldevs; d; d = d->next)

{ // 打印网络适配器列表

printf("%d. %s", ++i, d->name);

if (d->description)

printf(" (%s)\n", d->description);

else

printf(" (No description available)\n");

}

if (i == 0)

{

printf("\nNo interfaces found! Make sure WinPcap is installed.\n");

return -1;

}

printf("Enter the interface number (1-%d):", i);

scanf\_s("%d", &inum); //scanf C语言的标准输入函数

if (inum < 1 || inum > i) //判断输入是否正确 是否是1-i之间的数字

{

printf("\nInterface number out of range.\n");

pcap\_freealldevs(alldevs); //释放设备列表

return -1;

}

//跳转到已选网络适配卡

for (d = alldevs, i = 0; i < inum - 1; d = d->next, i++);

if ((adhandle = pcap\_open(d->name, 65536, PCAP\_OPENFLAG\_PROMISCUOUS, 1000, NULL, errbuf)) == NULL)//打开网络适配卡

{

fprintf(stderr, "\nUnable to open the adapter. %s is not supported by WinPcapn");

pcap\_freealldevs(alldevs); // 释放设备列表

return -1;

}

//// 检查数据链路层，只考虑以太网

//if (pcap\_datalink(adhandle) != DLT\_EN10MB)

//{

// fprintf(stderr, "\nThis program works only on Ethernet networks.\n");

// pcap\_freealldevs(alldevs); // 释放设备列表

// return -1;

//}

if (d->addresses != NULL)

// 获得接口第一个地址的网络掩码

netmask = ((struct sockaddr\_in\*) (d->addresses->netmask))->sin\_addr.S\_un.S\_addr;

else //如果接口没有地址， 则假设一个 C 类的掩码

netmask = 0xffffff;

if (pcap\_compile(adhandle, &fcode, packet\_filter, 1, netmask) < 0) //编译过滤器 执行成功返回 0，否则返回-1。 //fcode：pcap\_compile（）返回结果

{

fprintf(stderr, "\nUnable to compile the packet filter. Check the syntax.\n");

pcap\_freealldevs(alldevs); // 释放设备列表

return -1;

}

if (pcap\_setfilter(adhandle, &fcode) < 0) //设置过滤器

{

fprintf(stderr, "nError setting the filter.\n");

pcap\_freealldevs(alldevs); // 释放设备列表

return -1;

}

printf("\nlistening on %s...\n", d->description);

pcap\_freealldevs(alldevs); // 释放设备列表

pcap\_loop(adhandle, 0, packet\_handler, NULL); // 开始捕捉 用回调方式捕获数据包

return 0;

}

/\*

回调函数，当收到每一个数据包时会被 libpcap 所调用

param： pcap\_loop 函数传导的 user；

header： 捕获数据包的首部指针；

pkt\_data ： 捕获数据包的内容。

\*/

void packet\_handler(u\_char\* param, const struct pcap\_pkthdr\* header, const u\_char\* pkt\_data)

{

struct tm\* ltime;

char timestr[16];

ip\_header\* ih;

icmp\_header\* ich;

u\_int ip\_len;

MAC\_header\* mah = (MAC\_header\*)pkt\_data;

u\_short sport, //源端口

dport; //目的端口

time\_t local\_tv\_sec;

local\_tv\_sec = header->ts.tv\_sec; //将时间戳转换成可识别的格式

/\*

c 库函数 struct tm \*localtime(const time\_t \*timer) 使用 timer 的值来填充 tm 结构。timer 的值被分解为 tm 结构，并用本地时区表示。

timer -- 这是指向表示日历时间的 time\_t 值的指针。

返回值：该函数返回指向 tm 结构的指针，该结构带有被填充的时间信息。下面是 tm 结构的细节：

struct tm {

int tm\_sec; //秒，范围从 0 到 59

int tm\_min; //分，范围从 0 到 59

int tm\_hour; // 小时，范围从 0 到 23

int tm\_mday; // 一月中的第几天，范围从 1 到 31

int tm\_mon; // 月份，范围从 0 到 11

int tm\_year; // 自 1900 起的年数

int tm\_wday; // 一周中的第几天，范围从 0 到 6

int tm\_yday; // 一年中的第几天，范围从 0 到 365

int tm\_isdst; // 夏令时

};

\*/

ltime = localtime(&local\_tv\_sec);

/\*

c 库函数 size\_t strftime(char \*str, size\_t maxsize, const char \*format, const struct tm \*timeptr) 根据 format 中定义的格式化规则，格式化结构 timeptr 表示的时间，并把它存储在 str 中。

str -- 这是指向目标数组的指针，用来复制产生的 c 字符串。

maxsize -- 这是被复制到 str 的最大字符数。

format -- 这是 c 字符串，包含了普通字符和特殊格式说明符的任何组合。这些格式说明符由函数替换为表示 tm 中所指定时间的相对应值。格式说明符：

https://www.runoob.com/cprogramming/c-function-strftime.html

如果产生的 c 字符串小于 size 个字符（包括空结束字符），则会返回复制到 str 中的字符总数（不包括空结束字符），否则返回零。

\*/

strftime(timestr, sizeof timestr, "%H:%M:%S", ltime); //打印 小时：分钟：秒

//打印数据包的时间戳和长度

printf("%s.%.6d len:%d ", timestr, header->ts.tv\_usec, header->len);

//获得 IP 数据包头部的位置

ih = (ip\_header\*)(pkt\_data + 14);// 收到的数据包是MAC帧，MAC帧的头部为14个字节（目的地址-6字节 --> 源地址-6字节 --> 类型-2字节），然后才是mac帧的数据部分（即开始ip数据报）

ip\_len = (ih->ver\_ihl & 0xf) \* 4; //得到ip的首部长度\*4（因为首部长度的单位是4字节） 最终得到ip数据报以字节为单位的首部的长度

ich = (icmp\_header\*)((u\_char\*)ih + ip\_len); // 获得icmp首部的位置

//将网络字节序列转换成主机字节序列

//打印 IP 地址

printf("%d.%d.%d.%d -> %d.%d.%d.%d\n",

ih->saddr.byte1,

ih->saddr.byte2,

ih->saddr.byte3,

ih->saddr.byte4,

ih->daddr.byte1,

ih->daddr.byte2,

ih->daddr.byte3,

ih->daddr.byte4

);

write\_log(timestr, header, mah,ih,ich);

}

//将结果写入日志文件

void write\_log(char\* timestr, const struct pcap\_pkthdr\* header, MAC\_header\* mah, ip\_header\* ih, icmp\_header\*ich) {

FILE\* fp;

fp = fopen("Receive.log", "a");//在当前程序的目录下创建和追加用于读写的空的日志文件：Receive.log

fprintf(fp, "%s.%.6d len:%d \n###MAC首部###\n目的MAC地址：%X:%X:%X:%X:%X:%X \n 源MAC地址： %X:%X:%X:%X:%X:%X \n类型：%X\n",

timestr,

header->ts.tv\_usec,

header->len,

mah->daddr.byte1,

mah->daddr.byte2,

mah->daddr.byte3,

mah->daddr.byte4,

mah->daddr.byte5,

mah->daddr.byte6,

mah->saddr.byte1,

mah->saddr.byte2,

mah->saddr.byte3,

mah->saddr.byte4,

mah->saddr.byte5,

mah->saddr.byte6,

mah->type

);

u\_char version = ih->ver\_ihl >> 4;//拆分版本

u\_char head\_len = ih->ver\_ihl & 0x0f;//拆分首部长度

u\_char flag = ih->flags\_fo >> 13;//标志

u\_short mov\_to = ih->flags\_fo & 0x1fff;//片偏移

char\* s;

//拆分协议

switch (ih->proto)

{

case 1:

s = new char[strlen("ICMP") + 1];

strcpy(s, "ICMP");

break;

case 2:

s = new char[strlen("IGMP") + 1];

strcpy(s, "IGMP");

break;

case 4:

s = new char[strlen("IP") + 1];

strcpy(s, "IP");

break;

case 6:

s = new char[strlen("TCP") + 1];

strcpy(s, "TCP");

break;

case 8:

s = new char[strlen("EGP") + 1];

strcpy(s, "EGP");

break;

case 9:

s = new char[strlen("IGP") + 1];

strcpy(s, "IGP");

break;

case 17:

s = new char[strlen("UDP") + 1];

strcpy(s, "UDP");

break;

case 41:

s = new char[strlen("IPv6") + 1];

strcpy(s, "IPv6");

break;

case 50:

s = new char[strlen("ESP") + 1];

strcpy(s, "ESP");

break;

case 89:

s = new char[strlen("OSPF") + 1];

strcpy(s, "OSPF");

break;

default:

s = new char[strlen("Unkown") + 1];

strcpy(s, "Unkown");

break;

}

fprintf(fp, "###IP数据报首部###\n版本：%d\n首部长度：%d\n总长度：%d字节\n标识：%x\n标志：%x\n片偏移：%0x(\*8字节)\n生存时间：%d\n协议：%s\n首部校验和：%x\n源地址：%d.%d.%d.%d\n目的地址：%d.%d.%d.%d\n",

version,

head\_len,

ih->tlen,

ih->identification,

flag,

mov\_to,

ih->ttl,

s,

ih->crc,

ih->saddr.byte1,

ih->saddr.byte2,

ih->saddr.byte3,

ih->saddr.byte4,

ih->daddr.byte1,

ih->daddr.byte2,

ih->daddr.byte3,

ih->daddr.byte4

);

switch (ich->type)

{

case 0:

s = new char[strlen("查询报文----应答") + 1];

strcpy(s, "查询报文----应答");

break;

case 3:

s = new char[strlen("差错报告报文----目的不可达") + 1];

strcpy(s, "差错报告报文----目的不可达");

break;

case 4:

s = new char[strlen("控制报文----源抑制") + 1];

strcpy(s, "控制报文----源抑制");

break;

case 5:

s = new char[strlen("控制报文----路由重定向") + 1];

strcpy(s, "控制报文----路由重定向");

break;

case 8:

s = new char[strlen("查询报文----回送请求") + 1];

strcpy(s, "查询报文----回送请求");

break;

case 10:

s = new char[strlen("查询报文----路由器请求") + 1];

strcpy(s, "查询报文----路由器请求");

break;

case 11:

s = new char[strlen("差错报告报文----超时") + 1];

strcpy(s, "差错报告报文----超时");

break;

case 12:

s = new char[strlen("差错报告报文----参数出错") + 1];

strcpy(s, "差错报告报文----参数出错");

break;

case 13:

s = new char[strlen("查询报文----时间戳请求") + 1];

strcpy(s, "查询报文----时间戳请求");

break;

case 14:

s = new char[strlen("查询报文----时间戳应答") + 1];

strcpy(s, "查询报文----时间戳应答");

break;

case 17:

s = new char[strlen("查询报文----掩码请求") + 1];

strcpy(s, "查询报文----掩码请求");

break;

case 18:

s = new char[strlen("查询报文----掩码应答") + 1];

strcpy(s, "查询报文----掩码应答");

break;

case 19:

s = new char[strlen("差错报告报文----通告") + 1];

strcpy(s, "差错报告报文----通告");

break;

default:

s = new char[strlen("Unkown") + 1];

strcpy(s, "Unkown");

break;

}

fprintf(fp, "###ICMP首部###\n类型：%s\n代码：%x\n校验和：%x\n",

s,

ich->code,

ich->checksum

);

fprintf(fp, "=================================================================\n");

fclose(fp);

}

**题目2：子网内的文件传送**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <WinSock2.h>

#include<iostream>

#include <thread>

#include<vector>

#include <cstdlib>

#include<io.h>

using namespace std;

#pragma warning (disable: 4996)

#pragma comment(lib, "WS2\_32")

#define PORT 8001 ///端口号

#define BUFFER\_SIZE 1024 //缓冲区的大小

#define FILE\_NAME\_MAX\_SIZE 512 //文件名的长度

#define broRevPort 7000

#define OnLine '1'//告知他人当前机子上线了

#define UnLine '2'//告知他人当前机子下线了

#define SentFile '3'//发送文件

vector<char\*> onlineList;//在线好友列表

//向好友列表添加IP

void add(char\* addr) {

bool f = false;

//查看当前要加入的IP地址是否已经存在

//此处不能用自带的count函数！这样匹配的是char\*的地址

for (int i = 0; i < onlineList.size(); i++) {

if (\*addr == \*onlineList[i]) {

f = true;

break;

}

}

if (!f) {

onlineList.push\_back(addr);

printf("添加成功!\n");

}

else {

printf("当前好友列表中已经存在该IP了\n");

}

return;

}

//删除列表中的下线好友

void del(char\* addr) {

bool f = false;

//查看当前要加入的IP地址是否已经存在

//此处不能用自带的count函数！这样匹配的是char\*的地址

int i = 0;

for (; i < onlineList.size(); i++) {

if (\*addr == \*onlineList[i]) {

f = true;

break;

}

}

if (f) {

onlineList.erase(onlineList.begin() + i);

printf("删除成功！\n");

}

else {

printf("当前好友列表中没有该IP\n");

}

return;

}

//向发送udp的客户端发起TCP请求

void sendOnLine(char \* addr) {

SOCKET c\_Socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (SOCKET\_ERROR == c\_Socket)

{

printf("# 创建sock失败！ #\n");

return;

}

// 指定服务端的地址

sockaddr\_in server\_addr;

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.S\_un.S\_addr = inet\_addr(addr);

server\_addr.sin\_port = htons(PORT);

// 建立连接

if (SOCKET\_ERROR == connect(c\_Socket, (LPSOCKADDR)&server\_addr, sizeof(server\_addr)))

{

printf("# 无法连接到服务器！ #\n");

return;

}

printf("# 成功连接到服务器！ #\n");

char bufferSendLabel[BUFFER\_SIZE] = { 0 };

memset(bufferSendLabel, 0, BUFFER\_SIZE);

bufferSendLabel[0] = OnLine;

bufferSendLabel[1] = '\0';

if (send(c\_Socket, bufferSendLabel, BUFFER\_SIZE, 0) < 0)

{

printf("# 发送上线回应失败！ #\n");

return;

}

closesocket(c\_Socket);

return;

}

//向用户发送下线通知

void sendUnLine(char\* sIP) {

SOCKET c\_Socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (SOCKET\_ERROR == c\_Socket)

{

printf("# 创建sock失败！ #\n");

return;

}

for (int i = 0; i < onlineList.size(); i++) {

// 指定服务端的地址

sockaddr\_in server\_addr;

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.S\_un.S\_addr = inet\_addr(onlineList[i]);

server\_addr.sin\_port = htons(PORT);

// 建立连接

if (SOCKET\_ERROR == connect(c\_Socket, (LPSOCKADDR)&server\_addr, sizeof(server\_addr)))

{

printf("# 无法连接到服务器！ #\n");

return;

}

printf("# 成功连接到服务器！ #\n");

//发送选项

char bufferSendLabel[BUFFER\_SIZE] = { 0 };

memset(bufferSendLabel, 0, BUFFER\_SIZE);

bufferSendLabel[0] = UnLine;

bufferSendLabel[1] = '\0';

if (send(c\_Socket, bufferSendLabel, BUFFER\_SIZE, 0) < 0)

{

printf("# 发送下线label失败！ #\n");

return;

}

}

closesocket(c\_Socket);

return;

}

//发送文件

void sentfile(char\*deIP, char\* file\_name) {

// 创建socket

// AF\_INET: IPv4 网络协议的套接字类型;

// SOCK\_STREAM:提供面向连接的稳定数据传输，即TCP协议;

SOCKET c\_Socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (SOCKET\_ERROR == c\_Socket)

{

printf("# 创建sock失败！ #\n");

//system("pause");

//exit(1);

return;

}

// 指定服务端的地址

sockaddr\_in server\_addr;

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.S\_un.S\_addr = inet\_addr(deIP);

server\_addr.sin\_port = htons(PORT);

// 建立连接

if (SOCKET\_ERROR == connect(c\_Socket, (LPSOCKADDR)&server\_addr, sizeof(server\_addr)))

{

printf("# 无法连接到服务器！ #\n");

return;

}

printf("# 成功连接到服务器！ #\n");

//发送选项

char buffer[BUFFER\_SIZE];

memset(buffer, 0, BUFFER\_SIZE);

buffer[0] = SentFile;

buffer[1] = '\0';

if (send(c\_Socket, buffer, BUFFER\_SIZE, 0) < 0)

{

printf("# 发送label失败！ #\n");

return;

}

// 打开文件，准备读取文件

FILE\* fp = fopen(file\_name, "rb"); //windows下是"rb",表示打开一个只写的二进制文件

if (NULL == fp)

{

printf("# 文件名为 %s的文件未找到！ #\n", file\_name);

memset(buffer, 0, BUFFER\_SIZE);

buffer[0] = '0';

send(c\_Socket, buffer, BUFFER\_SIZE, 0);

return;

}

else

{

memset(buffer, 0, BUFFER\_SIZE);

buffer[0] = '1';

send(c\_Socket, buffer, BUFFER\_SIZE, 0);

// 向服务器发送文件名

memset(buffer, 0, BUFFER\_SIZE);

strncpy(buffer, file\_name, strlen(file\_name) > BUFFER\_SIZE ? BUFFER\_SIZE : strlen(file\_name));

if (send(c\_Socket, buffer, BUFFER\_SIZE, 0) < 0)

{

printf("# 发送文件名失败！ #\n");

return;

}

memset(buffer, 0, BUFFER\_SIZE);

int length = 0;

while ((length = fread(buffer, sizeof(char), BUFFER\_SIZE, fp)) > 0)

{

if (send(c\_Socket, buffer, length, 0) < 0)

{

printf("# 发送文件%s失败！ #\n", file\_name);

break;

}

memset(buffer, 0, BUFFER\_SIZE);

}

fclose(fp);

printf("# 发送文件%s 成功! #\n", file\_name);

}

closesocket(c\_Socket);

return;

}

void serviceTCP(char\* IP) {

// 声明并初始化一个服务端(本地)的地址结构

sockaddr\_in server\_addr;

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.S\_un.S\_addr = INADDR\_ANY;

//server\_addr.sin\_addr.S\_un.S\_addr = IP;

server\_addr.sin\_port = htons(PORT);

// 创建socket

SOCKET m\_Socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (SOCKET\_ERROR == m\_Socket)

{

printf("# 创建sock失败！ #\n");

system("pause");

exit(1);

}

// 绑定socket和服务端(本地)地址

if (SOCKET\_ERROR == bind(m\_Socket, (LPSOCKADDR)&server\_addr, sizeof(server\_addr)))

{

printf("服务器绑定失败: %d\n", WSAGetLastError());

system("pause");

exit(1);

}

// 监听

if (SOCKET\_ERROR == listen(m\_Socket, 10))

{

printf("服务器无法监听: %d\n", WSAGetLastError());

system("pause");

exit(1);

}

sockaddr\_in client\_addr;

int client\_addr\_len = sizeof(client\_addr);

while (1)

{

//printf("正在监听来自客户端的消息...\n");

memset(&client\_addr, 0, sizeof(sockaddr\_in));

SOCKET m\_New\_Socket = accept(m\_Socket, (sockaddr\*)&client\_addr, &client\_addr\_len);

if (SOCKET\_ERROR == m\_New\_Socket)

{

printf("Server Accept Failed: %d\n", WSAGetLastError());

break;

}

char bufferTCPServer[BUFFER\_SIZE] = { 0 };

/\*

void \*memset(void \*str, int c, size\_t n) 复制字符 c（一个无符号字符）到参数 str 所指向的字符串的前 n 个字符。

参数：

str -- 指向要填充的内存块。

c -- 要被设置的值。该值以 int 形式传递，但是函数在填充内存块时是使用该值的无符号字符形式。

n -- 要被设置为该值的字节数。

返回值

该值返回一个指向存储区 str 的指针。

\*/

//清空缓冲区

memset(bufferTCPServer, 0, BUFFER\_SIZE);

/\*

功能： 从本地已经建立连接的数据报或流式套接口接收数据。格式如下：

int recv( SOCKET s, const char \* buf, int len, int flags);

参数： s：数据接收方（与发送方已建立连接）的套接字描述符；

buf：用于接收数据的字符缓冲区指针；

len：用户缓冲区长度，以字节为单位；

flags：执行此调用的方式，一般设置为 0。

返回值：执行成功返回从套接字 s 实际读入到 buf 中的数据字节总数；如果连

接中止，返回 0； 否则返回 SOCKET\_ERROR。

\*/

if (recv(m\_New\_Socket, bufferTCPServer, BUFFER\_SIZE, 0) < 0)

{

printf("服务器接收客户端的label失败!\n");

break;

}

int f = 0;

switch (bufferTCPServer[0])

{

case OnLine://上线消息

f = 1;

break;

case UnLine://下线消息

f = 2;

break;

case SentFile://发送文件消息

f = 3;

break;

default:

printf("无用的选项\n");

break;

}

if (f == 1) {

add(inet\_ntoa(client\_addr.sin\_addr));

}

else if (f == 2) {

del(inet\_ntoa(client\_addr.sin\_addr));

}

else if (f == 3) {

memset(bufferTCPServer, 0, BUFFER\_SIZE);

recv(m\_New\_Socket, bufferTCPServer, BUFFER\_SIZE, 0);

if (bufferTCPServer[0] == '0') {

printf("客户端上没找到文件!\n");

}

else {

memset(bufferTCPServer, 0, BUFFER\_SIZE);

if (recv(m\_New\_Socket, bufferTCPServer, BUFFER\_SIZE, 0) < 0)

{

printf("服务器接收文件名失败!\n");

break;

}

char file\_name[FILE\_NAME\_MAX\_SIZE + 1];

memset(file\_name, 0, FILE\_NAME\_MAX\_SIZE + 1);

/\*

char \*strncpy(char \*dest, const char \*src, size\_t n)

把 src 所指向的字符串复制到 dest，最多复制 n 个字符。当 src 的长度小于 n 时，dest 的剩余部分将用空字节填充。

参数

dest -- 指向用于存储复制内容的目标数组。

src -- 要复制的字符串。

n -- 要从源中复制的字符数。

返回值

该函数返回最终复制的字符串。

\*/

strncpy(file\_name, bufferTCPServer, strlen(bufferTCPServer) > FILE\_NAME\_MAX\_SIZE ? FILE\_NAME\_MAX\_SIZE : strlen(bufferTCPServer));

printf("%s\n", file\_name);

printf("开始接收客户端发送过来的文件%s\n", file\_name);

FILE\* fp = fopen(file\_name, "wb");

if (NULL == fp)

{

printf("无法写入文件名为 %s的文件\n", file\_name);

}

else

{

memset(bufferTCPServer, 0, BUFFER\_SIZE);

int length = 0;

while ((length = recv(m\_New\_Socket, bufferTCPServer, BUFFER\_SIZE, 0)) > 0)

{

if (fwrite(bufferTCPServer, sizeof(char), length, fp) < length)

{

printf("文件名为%s的文件写入失败！\n", file\_name);

break;

}

memset(bufferTCPServer, 0, BUFFER\_SIZE);

}

fclose(fp);

printf("成功接收来自客户端的%s文件!\n", file\_name);

}

closesocket(m\_New\_Socket);

}

}

}

closesocket(m\_Socket);

return ;

}

void initNetwork() {

// 初始化socket dll

WSADATA wsaData;

// MAKEWORD

// 原型：#define MAKEWORD(a, b) ((WORD)(((BYTE)(((DWORD\_PTR)(a)) & 0xff)) | ((WORD)((BYTE)(((DWORD\_PTR)(b)) & 0xff))) << 8))

// 作用：将两个byte型的a和b合并为一个word型，高8为是b，低8位是a；

// 返回值：一个无符号的16位整形

WORD socketVersion = MAKEWORD(2, 0);

if (WSAStartup(socketVersion, &wsaData) != 0)

{

printf("# 初始化sock dll失败! #\n");

exit(1);

system("pause");

return;

}

}

void serverudp(char\* IP) {

initNetwork();

//创建一个udp socket

SOCKET udp\_server\_socket = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

if (udp\_server\_socket == SOCKET\_ERROR) {

std::cout << "Failed to create socket!" << std::endl;

return;

}

sockaddr\_in server\_addr;

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(broRevPort);

server\_addr.sin\_addr.S\_un.S\_addr = INADDR\_ANY;

bool so\_broadcast = true;

int ret = setsockopt(udp\_server\_socket, SOL\_SOCKET, SO\_REUSEADDR, (char\*)&so\_broadcast, sizeof(so\_broadcast));

//bind(udp\_server\_socket, (sockaddr\*)&server\_addr, sizeof(server\_addr));

if (bind(udp\_server\_socket, (struct sockaddr\*) & server\_addr, sizeof(struct sockaddr)) == SOCKET\_ERROR) {

printf("bind fail\n");

return ;

}

char bufferUDPServer[BUFFER\_SIZE] = { 0 };

sockaddr\_in client\_addr;

int client\_addr\_len = sizeof(sockaddr);

while (true) {

memset(bufferUDPServer, 0, BUFFER\_SIZE);

//printf("udp listening...\n");

// 用来保存客户端的网络信息。

memset(&client\_addr, 0, sizeof(sockaddr\_in));

// 接收客户端发来的信息,server阻塞在这里，等待client发来报文。

ret = recvfrom(udp\_server\_socket, bufferUDPServer, BUFFER\_SIZE, 0, (sockaddr\*)&client\_addr, &client\_addr\_len);

if (ret != SOCKET\_ERROR) {

switch (bufferUDPServer[0])

{

case OnLine:

add(inet\_ntoa(client\_addr.sin\_addr));

sendOnLine(inet\_ntoa(client\_addr.sin\_addr));

break;

default:

break;

}

}

else {

printf("UDP receive Failed！\n");

}

}

closesocket(udp\_server\_socket);

WSACleanup();

return;

}

void clientudp(char\* IP,char sendLabel) {

initNetwork();

SOCKET udp\_client\_socket = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

if (udp\_client\_socket == SOCKET\_ERROR) {

std::cout << "Failed to create socket!" << std::endl;

return ;

}

/\*

sockaddr\_in client\_addr;

memset(&client\_addr, 0, sizeof(sockaddr\_in));

client\_addr.sin\_family = AF\_INET;

client\_addr.sin\_port = htons(PORT);

client\_addr.sin\_addr.S\_un.S\_addr = inet\_addr(IP);

\*/

bool so\_broadcast = true;

int ret = setsockopt(udp\_client\_socket, SOL\_SOCKET, SO\_BROADCAST, (char\*)&so\_broadcast, sizeof(so\_broadcast));

if (ret < 0)

{

printf("无法把套接字设为广播类型\n");

return;

}

sockaddr\_in brocast;

char buf[BUFFER\_SIZE] = { 0 };

brocast.sin\_family = AF\_INET;

brocast.sin\_addr.s\_addr = INADDR\_BROADCAST;//广播地址

brocast.sin\_port = htons(broRevPort);

memset(buf, 0, BUFFER\_SIZE);

buf[0] = sendLabel;

ret = sendto(udp\_client\_socket, buf, strlen(buf), 0, (sockaddr\*)&brocast, sizeof(sockaddr));

if (ret == SOCKET\_ERROR) {

std::cout << "Failed to send!" << std::endl;

return ;

}

closesocket(udp\_client\_socket);

WSACleanup();

return ;

}

//刷新用户

void refreshUser(char\* sIP) {

clientudp(sIP, OnLine);

//Sleep(1000);

}

//显示当前在线用户

void showOnLineUser(char\* sIP) {

//refreshUser(sIP);

//Sleep(100);

printf("当前在线好友：\n");

printf("序号 IP地址\n");

for (int i = 0; i < onlineList.size(); i++) {

printf("%d %s\n", i + 1, onlineList[i]);

}

}

int main()

{

initNetwork();

printf("#-----------------欢迎来到《局域网文件传送程序》-------------#\n");

printf("#-----------------请指定你的IP地址----------------------------#\n");

char sIP[20];

cin >> sIP;

thread ts(serviceTCP,sIP);

ts.detach();

thread us(serverudp, sIP);

us.detach();

Sleep(20);

refreshUser(sIP);

Sleep(1000);

bool f = true;

char newIP[20];

while (f) {

printf("#------------------请选择你要进行的操作------------------#\n");

printf("#------------------1.刷新好友列表【1】-------------------#\n");

printf("#------------------2.显示好友列表【2】-------------------#\n");

printf("#------------------3.添加好友【3】-----------------------#\n");

printf("#------------------4.删除好友【4】-----------------------#\n");

printf("#------------------5.发送文件【5】-----------------------#\n");

printf("#------------------6.退出程序【6】-----------------------#\n");

vector<int>SendList;

int n;

cin >> n;

switch (n)

{

case 1:

refreshUser(sIP);

break;

case 2:

showOnLineUser(sIP);

break;

case 3:

printf("#------------------请输入要添加的好友IP-----------------------#\n");

memset(newIP, 0, 20);

cin >> newIP;

add(newIP);

break;

case 4:

printf("#------------------请输入要删除的好友IP-----------------------#\n");

memset(newIP, 0, 20);

cin >> newIP;

del(newIP);

break;

case 5:

printf("# 请输入要传送的文件名 #\n");

char file\_name[FILE\_NAME\_MAX\_SIZE + 1];

memset(file\_name, 0, FILE\_NAME\_MAX\_SIZE + 1);

scanf("%s", &file\_name);

printf("#------------------请输入要发送文件的好友序号（以0结束）------#\n");

int k;

cin >> k;

while (k != 0) {

if(k<=onlineList.size())

SendList.push\_back(k);

else {

printf("没有这个序号\n");

}

cin >> k;

}

for (int i = 0; i < SendList.size(); i++) {

char\* deIP = onlineList[SendList[i] - 1];

sentfile(deIP, file\_name);

}

break;

case 6:

sendUnLine(sIP);

f = false;

break;

default:

printf("输入错误请重新输入\n");

break;

}

}

printf("#--------您已退出程序，感谢您的使用，再见！---------#\n");

system("pause");

return 0;

}

**指导教师评语：**