## 网络技术与应用第五次实验报告

网络空间安全学院 物联网工程 2111673 岳志鑫

**一、实验要求**

简单路由器程序设计实验的具体要求为：

（1）设计和实现一个路由器程序，要求完成的路由器程序能和现有的路由器产品（如思科路由器、华为路由器、微软的路由器等）进行协同工作。

（2）程序可以仅实现IP数据报的获取、选路、投递等路由器要求的基本功能。可以忽略分片处理、选项处理、动态路由表生成等功能。

（3）需要给出路由表的手工插入、删除方法。

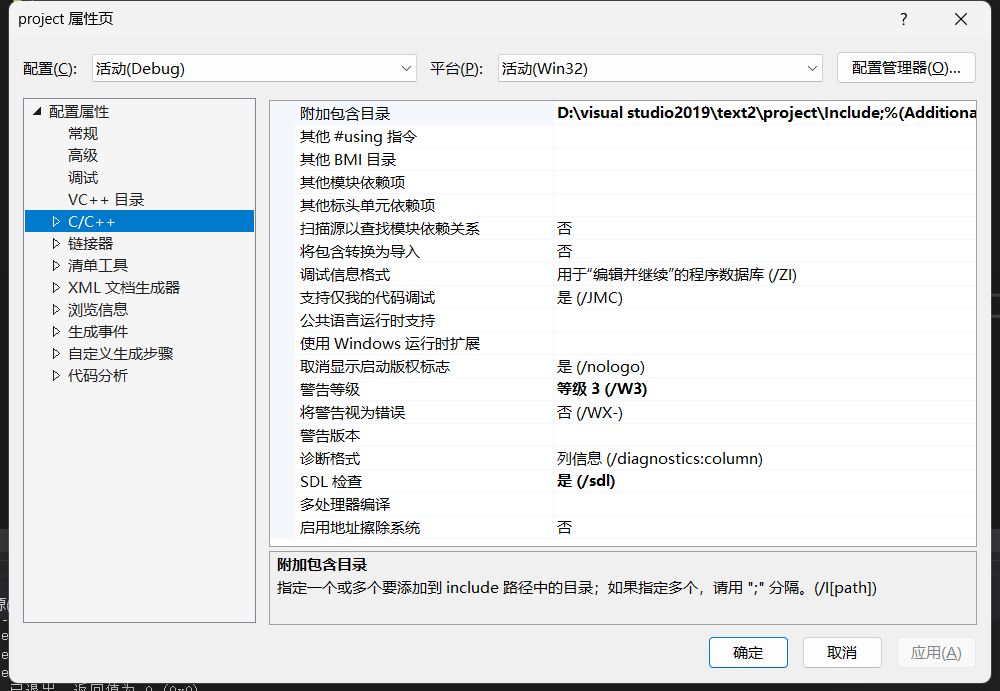
（4）需要给出路由器的工作日志，显示数据报获取和转发过程。

（5）完成的程序须通过现场测试，并在班（或小组）中展示和报告自己的设计思路、开发和实现过程、测试方法和过程。

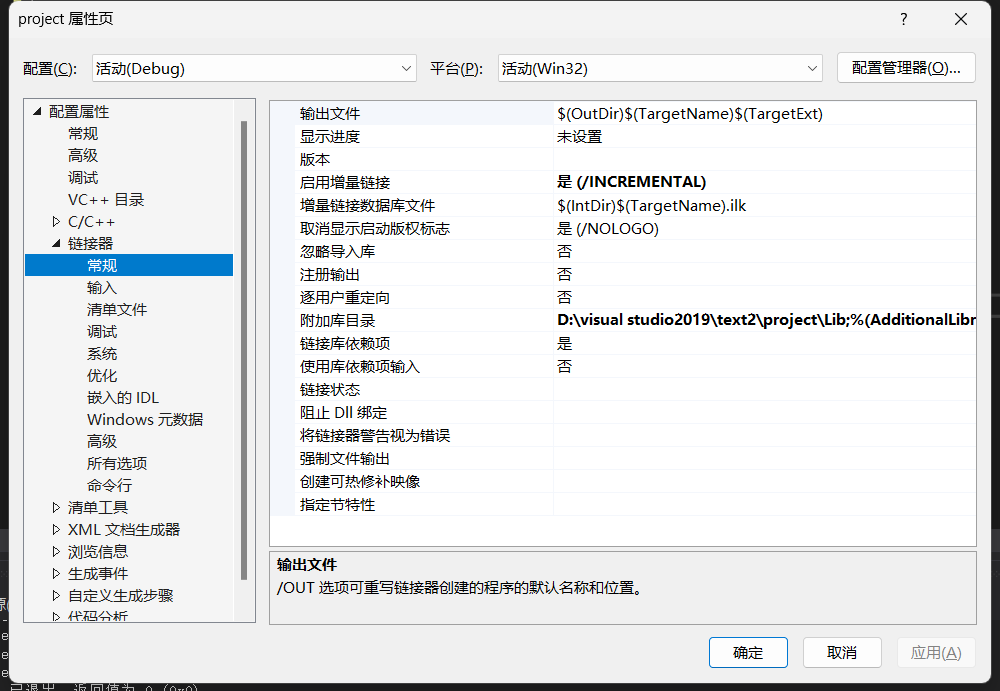
**二、实验过程**

**1.环境配置**

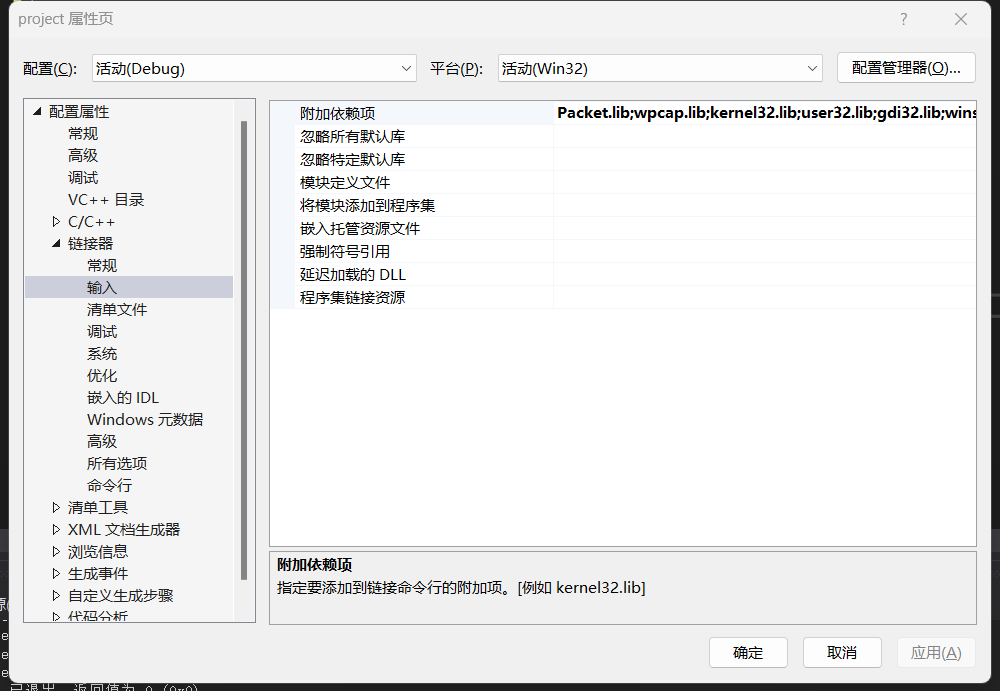
项目→属性→C/C++→常规→附加包含目录：添加sdk中的Include目录



项目→属性→链接器→常规→附加库目录：添加sdk中的Lib目录



项目→属性→链接器→输入→附加依赖项：添加Packet.lib;wpcap.lib;

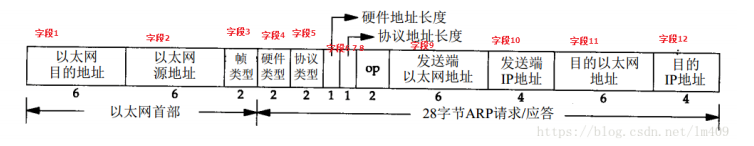


将Include路径添加到Project属性→VC++目录→包含目录，Lib路径添加到Project属性→VC++目录→库目录



**2.自定义结构体**

ARP数据包构成如下：



根据此可以写出ARP的结构体：

typedef struct ARP\_Frame//ARP数据

{

Frame\_Header FrameHeader;

WORD HardwareType; //硬件类型

WORD ProtocolType; //协议类型

BYTE HLen; //硬件长度

BYTE PLen; //协议长度

WORD op; //操作类型

BYTE SrcMAC[6]; //源MAC地址

DWORD SrcIP; //源IP地址

BYTE DesMAC[6]; //目的MAC地址

DWORD DesIP; //目的IP地址

};

IP数据包构成如下：



IP首部的结构体：

typedef struct IP\_Header {//IP首部

BYTE Version;//版本号

BYTE TOS;//服务类型

WORD TotLen;//总长度字段

WORD ID;//标识

WORD Flagoffset;//分段偏移

BYTE TTL;//生命周期

BYTE Protocol;//协议字段

WORD Checksum;//校验和

ULONG SrcIP;//源IP

ULONG DstIP;//目的IP

};

帧首部定义如下：

typedef struct Frame\_Header//帧首部

{

BYTE DesMAC[6]; //目的地址

BYTE SrcMAC[6]; //源地址

WORD FrameType; //帧类型

};

数据包定义：

typedef struct Data {

Frame\_Header FrameHeader;

IP\_Header IPHeader;

char buf[0x80];

};

缓冲区数据包：

typedef struct Send\_Packet {

BYTE PktData[2000];// 数据缓存

ULONG DestIP; // 目的IP地址

bool flag = 1; // 是否有效，如果已经被转发或者超时，则置0

clock\_t time; // 判断是否超时，超时则删除

};

**3.主要类**

**（1）路由表类**

在路由表中使用链表作为基本数据结构，使用head和tail指针来控制路由表的增添、删除、查找、打印等功能。在增添路由表项中通过子网掩码的大小来决定插入链表的位置，便于实现最长匹配的原则。

//路由表

class RouteTable {

public:

RouteTableItem\* head;

RouteTableItem\* tail;

int num;//路由表项数

RouteTable() {

num = 0;

head = new RouteTableItem(inet\_addr(mymask[0]), (inet\_addr(myip[0])) & (inet\_addr(mymask[0])),0);

tail = new RouteTableItem;

head->nextitem = tail;

RouteTableItem\* temp = new RouteTableItem;

temp->destnet = (inet\_addr(myip[1])) & (inet\_addr(mymask[1]));;

temp->netmask = inet\_addr(mymask[1]);

temp->type = 0;

add(temp);

}

//添加表项（直接投递在最前，前缀长的在前面）

void add(RouteTableItem\* newt) {

num++;

//直接投递

if (newt->type == 0) {

newt->nextitem = head->nextitem;//插入在head后

head->nextitem = newt;

return;

}

//根据掩码的大小插入

RouteTableItem\* cur = head;

while (cur->nextitem != tail) {

if (cur->nextitem->type != 0 && cur->nextitem->netmask <= newt->netmask) {

break;

}

cur = cur->nextitem;

}

//插入在 cur 和 cur->next 之间

newt->nextitem = cur->nextitem;

cur->nextitem = newt;

}

//删除表项

void Delete(int index) {

if (index > num) {

printf("路由表项%d超过范围!\n", index);

return;

}

if (index == 0) { //删除头部

if (head->type == 0) {

printf("默认路由不可删除!\n");

}

else {

head = head->nextitem;

}

return;

}

RouteTableItem\* cur = head;

int i = 0;

while (i < index - 1 && cur->nextitem != tail) { //指针指向删除的位置

i++;

cur = cur->nextitem;

}

if (cur->nextitem->type == 0) {

printf("默认路由不可删除!\n");

}

else {

cur->nextitem = cur->nextitem->nextitem;

}

}

//路由表打印

void print() {

printf("<==============路由表=============>\n");

RouteTableItem\* cur = head;

int i = 1;

while (cur != tail) {

printf("【第%d条路由表项】\n", i);

cur->print();

cur = cur->nextitem;

i++;

}

}

//查找，最长前缀,返回下一跳的ip

DWORD find(DWORD destip) {

DWORD result;

RouteTableItem\* cur = head;

while (cur != tail) {

result = destip & cur->netmask;

if (result == cur->destnet) {

if (cur->type == 1) {

return cur->nextip;//转发

}

else if (cur->type == 0) {

return destip;//直接投递

}

}

cur = cur->nextitem;

}

printf("没有找到对应的路由表项!\n");

return -1;

}

};

**（2）ARP表类**

通过数组来储存ARP表，由于本次实验规模不大，所以初始化定义的数组大小为50。在ARP表类中定义了添加和查找的函数，通过路由表的计数arpnum来决定数组的下标。

class ARPTable

{

public:

DWORD IP;//IP

BYTE mac[6];//MAC

//添加

void add(DWORD ip, BYTE mac[6])

{

arp\_table[arpnum].IP = ip;

for (int i = 0; i < 6; i++)

{

arp\_table[arpnum].mac[i] = mac[i];

}

arpnum++;

}

//查找

int find(DWORD ip, BYTE mac[6])

{

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

{

if (ip == arp\_table[i].IP)

{

for (int j = 0; j < 6; j++)

{

mac[j] = arp\_table[i].mac[j];

}

return 1;

}

}

return 0;

}

}arp\_table[50];//最大数50

1. **主要函数**

**（1）转发**

在communicate函数中进行数据包以太网帧中的MAC地址的更换，然后打印数据包内容并发送数据包。

void communicate(Data data, BYTE nextmac[])

{

//拷贝数据包

Data\* temp = (Data\*)&data;

memcpy(temp->FrameHeader.SrcMAC, temp->FrameHeader.DesMAC, 6);//源MAC为本机MAC

memcpy(temp->FrameHeader.DesMAC, nextmac, 6);//目的MAC为下一跳MAC

temp->IPHeader.TTL -= 1;//TTL-1

if (temp->IPHeader.TTL < 0)return;//丢弃

SetChecksum(temp);//重新设置校验和

printf("<==============转发===============>\n");

//打印IP数据包

printf("源MAC地址:\t ");

for (int i = 0; i < 5; i++) {

printf("%02X-", temp->FrameHeader.SrcMAC[i]);

}

printf("%02X\n", temp->FrameHeader.SrcMAC[5]);

printf("目的MAC地址:\t");

for (int i = 0; i < 5; i++) {

printf("%02X-", temp->FrameHeader.DesMAC[i]);

}

printf("%02X\n", temp->FrameHeader.DesMAC[5]);

printf("源IP地址:\t ");

in\_addr addr;

memcpy(&addr, &temp->IPHeader.SrcIP, sizeof(temp->IPHeader.SrcIP));

printf("%s ", inet\_ntoa(addr));

printf("\n");

printf("目的IP地址:\t ");

memcpy(&addr, &temp->IPHeader.DstIP, sizeof(temp->IPHeader.DstIP));

printf("%s ", inet\_ntoa(addr));

printf("\n");

printf("TTL: %d\n", temp->IPHeader.TTL); // 十进制输出

pcap\_sendpacket(point, (const u\_char\*)temp, 74);//发送数据报

}

**（2）获取目的主机MAC地址**

在此函数中进行ARP数据包的组装，通过传入的ip地址进行发送ARP包，在接收消息的时候就可以将收到的数据包中的源MAC地址记录下来，达到获取MAC地址的目的。

void getdestmac(DWORD ip, BYTE mac[])

{

//初始化ARP数据包

ARP\_Frame rev\_ARPFrame;

/\*获取目的主机的MAC地址\*/

for (int i = 0; i < 6; i++) {

rev\_ARPFrame.FrameHeader.DesMAC[i] = 0xff; //广播地址

rev\_ARPFrame.FrameHeader.SrcMAC[i] = mac[i]; //MAC地址

rev\_ARPFrame.DesMAC[i] = 0x00; //设置为0

rev\_ARPFrame.SrcMAC[i] = mymac[i]; //本机MAC地址

}

rev\_ARPFrame.FrameHeader.FrameType = htons(0x0806);//帧类型为ARP

rev\_ARPFrame.HardwareType = htons(0x0001); //硬件类型为以太网

rev\_ARPFrame.ProtocolType = htons(0x0800);//协议类型为IP

rev\_ARPFrame.HLen = 6;//硬件地址长度为6

rev\_ARPFrame.PLen = 4;//协议类型长度为4

rev\_ARPFrame.op = htons(0x0001);//操作为ARP请求

rev\_ARPFrame.SrcIP = inet\_addr(myip[0]);//设置发送方ip地址

rev\_ARPFrame.DesIP = ip;

//发送数据包

pcap\_sendpacket(point, (u\_char\*)&rev\_ARPFrame, sizeof(ARP\_Frame));

}

**（3）接收数据的线程**

在接收线程中持续接听消息，接收到消息就对其进行判断。

如果是ARP数据包就查看其MAC地址和IP地址的映射是否已经储存在自己的arp表中，如果未储存就储存在arp表中，然后把符合新储存的arp表中的映射的缓存数据包发送；如果储存了就直接遍历缓冲区发送数据包。

如果目的mac是自己的mac且数据包是IP格式则首先检查校验和是否正确，然后再判断数据包的目的网络是否与自己直接相连，如果直接相连就查找ARP表，如果查到存在映射关系就直接投递，如果没有映射关系就先储存在缓冲区中，然后发送ARP请求目的IP与MAC地址的映射关系，等到获取了之后再投递；如果不是直接相连就查找ARP表是否存在下一跳IP地址和MAC地址的映射关系，如果存在就转发给下一跳，不存在也先储存在缓冲区中，然后发送ARP请求下一跳IP与MAC地址的映射关系，获取了再转发

DWORD WINAPI receive(LPVOID lparam)

{

ARPTable arptable;

RouteTable rtable = \*(RouteTable\*)(LPVOID)lparam;

while (1)

{

pcap\_pkthdr\* pkt\_header;

const u\_char\* packetData;

//等待接收消息

while (1)

{

int result = pcap\_next\_ex(point, &pkt\_header, &packetData);

if (result)

{

break;//接收到消息

}

}

Frame\_Header\* header = (Frame\_Header\*)packetData;

//数据包是ARP格式

if (ntohs(header->FrameType) == 0x806)

{

ARP\_Frame\* data = (ARP\_Frame\*)packetData;//格式化收到的包为帧首部+ARP首部类

printf("<==============接收ARP===============>\n");

//打印ARP数据包

// 提取MAC地址（0-6字节）

printf("源MAC地址:\t ");

for (int i = 0; i < 5; i++) {

printf("%02X-", data->FrameHeader.SrcMAC[i]);

}

printf("%02X\n", data->FrameHeader.SrcMAC[5]);

printf("目的MAC地址:\t");

for (int i = 0; i < 5; i++) {

printf("%02X-", data->FrameHeader.DesMAC[i]);

}

printf("%02X\n", data->FrameHeader.DesMAC[5]);

printf("源IP地址:\t ");

in\_addr addr;

memcpy(&addr, &data->SrcIP, sizeof(data->SrcIP));

printf("%s ", inet\_ntoa(addr));

printf("\n");

printf("目的IP地址:\t ");

memcpy(&addr, &data->DesIP, sizeof(data->DesIP));

printf("%s ", inet\_ntoa(addr));

printf("\n");

//收到ARP响应包

if (data->op == ntohs(0x0002)) {

BYTE tempmac[6];

//该映射关系已经存到arp表中，不做处理

if (arptable.find(data->SrcIP, tempmac)) {

}

//不在arp表中，插入

else

{

arptable.add(data->SrcIP, data->SrcMAC);

}

//遍历缓冲区，看是否有可以转发的包

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

{

in\_addr addr;

memcpy(&addr, &Buffer[i].DestIP, sizeof(Buffer[i].DestIP));

printf("destip:%s", inet\_ntoa(addr));

memcpy(&addr, &data->SrcIP, sizeof(data->SrcIP));

printf("srcip:%s", inet\_ntoa(addr));

if (Buffer[i].flag == 0)continue;

if (clock() - Buffer[i].time >= 6000) {//超时

Buffer[i].flag = 0;

continue;

}

if (Buffer[i].DestIP == data->SrcIP)

{

Data\* data\_send = (Data\*)Buffer[i].PktData;

Data temp = \*data\_send;

communicate(temp, data->SrcMAC);

Buffer[i].flag = 0;

break;

}

}

}

}

//目的mac是自己的mac且数据包是IP格式

if (compare(header->DesMAC, mymac) && ntohs(header->FrameType) == 0x800)

{

Data\* data = (Data\*)packetData; //格式化收到的包

//如果校验和不正确，则直接丢弃不进行处理

if (!Check(data))

{

printf("校验和出错\n");

continue;

}

printf("<==============接收IP===============>\n");

//打印IP数据包

printf("源MAC地址:\t ");

for (int i = 6; i < 12; ++i) {

printf("%02X", packetData[i]);

if (i < 11) printf("-");

}

printf("\n");

printf("目的MAC地址:\t");

for (int i = 0; i < 6; ++i) {

printf("%02X", packetData[i]);

if (i < 5) printf("-");

}

printf("\n");

printf("源IP地址:\t ");

for (int i = 26; i < 30; ++i) {

printf("%d", packetData[i]);

if (i < 29) printf(".");

}

printf("\n");

printf("目的IP地址:\t ");

for (int i = 30; i < 34; ++i) {

printf("%d", packetData[i]);

if (i < 33) printf(".");

}

printf("\n");

printf("TTL: %d\n", data->IPHeader.TTL); // 十进制输出

if (data->IPHeader.DstIP == inet\_addr(myip[0]) || data->IPHeader.DstIP == inet\_addr(myip[1]))//路由器两个网卡都可以接受

{

printf("发送给自己的数据包,交由电脑处理\n");

continue;

}

printf("<=================================>\n");

DWORD destip = data->IPHeader.DstIP; //目的IP地址

DWORD nextdestip = rtable.find(destip);//查找下一跳IP地址

if (nextdestip == -1)

{

printf("路由表项缺失！\n");

continue;//如果没有则直接丢弃或直接递交至上层

}

else

{

in\_addr next;

next.s\_addr = nextdestip;

printf("下一跳IP：%s\n", inet\_ntoa(next));

Data\* temp2 = (Data\*)packetData;

Data temp = \*temp2;

BYTE mac[6];

//直接投递

if (nextdestip == destip)

{

//如果ARP表中没有所需内容，则需要获取ARP

if (!arptable.find(destip, mac))

{

int flag2 = 0;

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

{

if (Buffer[i].flag == 0) //如果缓冲区中有已经被转发的，将数据包复制到该转发完成的数据包（覆盖用过的地方，节省空间）

{

flag2 = 1;

memcpy(Buffer[i].PktData, packetData, pkt\_header->len);

Buffer[i].flag = 1;

Buffer[i].time = clock();

Buffer[i].DestIP = destip;

getdestmac(destip, mac);

break;

}

}

if (flag2 == 0 && bufsize < 50) //缓冲区上限50

{

memcpy(Buffer[bufsize].PktData, packetData, pkt\_header->len);

Buffer[bufsize].flag = 1;

Buffer[bufsize].time = clock();

Buffer[bufsize].DestIP = destip;

bufsize++;

getdestmac(destip, mac);

}

else {

printf("缓冲区溢出！\n");

}

}

else if(arptable.find(destip, mac))

{

communicate(temp, mac);//转发

}

}

else //不是直接投递

{

if (!arptable.find(nextdestip, mac))

{

int flag3 = 0;

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

{

if (Buffer[i].flag == 0)

{

flag3 = 1;

memcpy(Buffer[i].PktData, packetData, pkt\_header->len);

Buffer[i].flag = 1;

Buffer[i].time = clock();

Buffer[i].DestIP = nextdestip;

getdestmac(nextdestip, mac);

break;

}

}

if (flag3 == 0 && bufsize < 50)

{

memcpy(Buffer[bufsize].PktData, packetData, pkt\_header->len);

Buffer[bufsize].flag = 1;

Buffer[bufsize].time = clock();

Buffer[bufsize].DestIP = nextdestip;

bufsize++;

getdestmac(nextdestip, mac);

}

else if (arptable.find(destip, mac))

{

communicate(temp, mac);//转发

}

}

else if (arptable.find(nextdestip, mac))

{

communicate(temp, mac);

}

}

}

}

}

}

1. **main主函数**

首先是获取网卡设备列表便于选择，选择后打开网卡设备并输出信息。获取本机的IP和MAC信息，再通过设置过滤器只接收ARP消息和IP消息，开启接收消息的线程。通过一个while循环一直接收用户的输入并给出路由表的相关操作响应。

int main() {

char errbuf[PCAP\_ERRBUF\_SIZE]; //错误信息缓冲区

/\*获取设备列表，打印信息\*/

pcap\_addr\_t\* a; //地址指针

pcap\_if\_t\* devices; //指向设备列表第一个

int i = 0; //统计设备数量

//输出错误信息

if (pcap\_findalldevs(&devices, errbuf) == -1)

{

printf("查找设备失败: %s\n", errbuf);

return 0;

}

//打印设备信息

//打印设备列表中设备信息

pcap\_if\_t\* count; //遍历用的指针

//输出设备名和描述信息

for (count = devices; count; count = count->next)//借助count指针从第一个设备开始访问到最后一个设备

{

printf("%d. %s", ++i, count->name);//输出设备信息和描述

if (count->description) {

printf("描述：(%s)\n", count->description);

}

for (a = count->addresses; a != NULL; a = a->next) {

if (a->addr->sa\_family == AF\_INET) {

char str[100];

strcpy(str, inet\_ntoa(((struct sockaddr\_in\*)a->addr)->sin\_addr));

//inet\_ntop(AF\_INET, getaddress((struct sockaddr\*)a->addr), str, sizeof(str));//将 a->addr 强制转换为 struct sockaddr\_in 类型的指针，并访问 sin\_addr 成员，其中包含了 IPv4 地址。

printf("IP地址：%s\n", str);

strcpy(str, inet\_ntoa(((struct sockaddr\_in\*)a->netmask)->sin\_addr));

//inet\_ntop(AF\_INET, getaddress((struct sockaddr\*)a->netmask), str, sizeof(str)); //将 a->netmask 强制转换为 struct sockaddr\_in 类型的指针，从a->netmask这个结构中提取子网掩码。

printf("子网掩码：%s\n", str);

strcpy(str, inet\_ntoa(((struct sockaddr\_in\*)a->broadaddr)->sin\_addr));

//inet\_ntop(AF\_INET, getaddress((struct sockaddr\*)a->broadaddr), str, sizeof(str));//将 a->netmask 强制转换为 struct sockaddr\_in 类型的指针，从a->broadaddr这个结构中提取广播地址。

printf("广播地址：%s\n", str);

}

}

}

//设备数量为0

if (i == 0) {

printf("存在错误！无查找设备！");

return 0;

}

printf("<============================================>\n");

/\*选择设备及打开网卡\*/

pcap\_if\_t\* count2; //遍历用的指针2

int num = 0;

printf("输入当前要连接的网卡序号：");

scanf("%d", &num);

while (num < 1 || num>2) {

printf("请检查网卡序号输入是否正确！");

printf("重新输入当前要连接的网卡序号：");

scanf("%d", &num);

}

count2 = devices;

for (int i = 1; i < num; i++) {//循环遍历指针选择第几个网卡

count2 = count2->next;

}

int k = 0;

//储存ip和子网掩码

for (a = count2->addresses; a != NULL; a = a->next) {

if (a->addr->sa\_family == AF\_INET) {

printf("接口卡名称:：(%s)\n", count2->name);

printf("接口卡描述：(%s)\n", count2->description);

//将 a->addr 强制转换为 struct sockaddr\_in 类型的指针，并访问 sin\_addr 成员，其中包含了 IPv4 地址。

strcpy(myip[k], inet\_ntoa(((struct sockaddr\_in\*)a->addr)->sin\_addr));

printf("IP地址：%s\n", myip);

strcpy(mymask[k], inet\_ntoa(((struct sockaddr\_in\*)a->netmask)->sin\_addr));

//将 a->netmask 强制转换为 struct sockaddr\_in 类型的指针，从a->netmask这个结构中提取子网掩码。

printf("子网掩码：%s\n", mymask);

k++;

}

}

//打开网络接口

//指定获取数据包最大长度为65536,可以确保程序可以抓到整个数据包，指定时间范围为200ms

point = pcap\_open(count2->name, 65536, PCAP\_OPENFLAG\_PROMISCUOUS, 200, NULL, errbuf);

if (point == NULL) { //打开当前网络接口失败

printf("打开当前网络接口失败");

return 0;

}

else {

printf("打开当前网络接口成功！");

}

pcap\_freealldevs(devices);

//获取本机的MAC地址

//组装报文

ARP\_Frame send\_ARPFrame;

for (int i = 0; i < 6; i++) {

send\_ARPFrame.FrameHeader.DesMAC[i] = 0xFF; //DesMAC设置为广播地址

send\_ARPFrame.DesMAC[i] = 0x00; //DesMAC设置为0

//SrcMAC用不到可以不设置

}

send\_ARPFrame.FrameHeader.FrameType = htons(0x0806); //帧类型为ARP，0x8100是一个IEEE 802.1Q帧，0x86DD是一个IPv6帧，0x0800代表 IP协议帧等

send\_ARPFrame.HardwareType = htons(0x0001); //硬件类型为以太网，IEEE 802 网络是 0x0006，Bluetooth是0x00FF等

send\_ARPFrame.ProtocolType = htons(0x0800); //协议类型为IPv4，IPv6是0x86DD，

send\_ARPFrame.HLen = 6; //硬件地址长度为6

send\_ARPFrame.PLen = 4; //协议地址长度为4

send\_ARPFrame.op = htons(0x0001); //操作为ARP请求，ARP响应是 0x0002

send\_ARPFrame.DesIP = inet\_addr(myip[0]); //设置为本机IP地址

pcap\_sendpacket(point, (u\_char\*)&send\_ARPFrame, sizeof(ARP\_Frame));

struct pcap\_pkthdr\* pkt\_header;

const u\_char\* packetData;

int ret;

while ((ret = pcap\_next\_ex(point, &pkt\_header, &packetData)) >= 0)//判断获取报文

{

printf("加载中...");

if (ret == 0) { //未捕获到数据包

continue;

}

//通过报文内容比对判断是否是要发打印的ARP数据包内容

//result=1，捕获成功

else if (\*(unsigned short\*)(packetData + 12) == htons(0x0806) //帧类型为ARP（htons(0x0806)）

&& \*(unsigned short\*)(packetData + 20) == htons(0x0002)

&& \*(unsigned long\*)(packetData + 28) == send\_ARPFrame.DesIP) //操作类型为ARP响应（htons(0x0002)）

{

printf("\n");

printf("<=================================>\n");

//用mac数组记录本机的MAC地址

for (int i = 0; i < 6; i++)

{

mymac[i] = \*(unsigned char\*)(packetData + 22 + i);

}

printf("获取MAC地址为：\t ");

for (int i = 6; i < 12; ++i) {

printf("%02X", packetData[i]);

if (i < 11) printf("-");

}

printf("\n");

printf("<=================================>\n");

break;

}

}

//输出错误信息

if (ret == -1) { //调用过程发生错误

printf("捕获数据包出错\n");

pcap\_freealldevs(devices);

return 0;

}

struct bpf\_program fcode;

//通过绑定过滤器，设置只捕获IP和ARP数据报

//编辑过滤字符串

if (pcap\_compile(point, &fcode, "ip or arp", 1, bpf\_u\_int32(inet\_addr(mymask[0]))) < 0)

{

fprintf(stderr, "\n设置过滤器失败！\n");

system("pause");

return 0;

}

//绑定过滤器

if (pcap\_setfilter(point, &fcode) < 0)

{

fprintf(stderr, "\n绑定过滤器失败！\n");

system("pause");

return 0;

}

RouteTable rtable; //路由表初始化

rtable.print();//输出路由表中的默认项

hThread = CreateThread(NULL, NULL, receive, LPVOID(&rtable), 0, &dwThreadId);

while (1)

{

printf("请选择要进行的操作：\n");

printf("【1.添加路由表项】\t【2.删除路由表项】\t【3.查看路由表项】\n");

int num;

scanf("%d", &num);

if (num == 1)

{

RouteTableItem\* rtableitem = new RouteTableItem;

rtableitem->type = 1;//用户添加

char buf[INET\_ADDRSTRLEN];

printf("请输入子网掩码:\n");

scanf("%s", &buf);

rtableitem->netmask = inet\_addr(buf);

printf("输入目的网络:\n");

scanf("%s", &buf);

rtableitem->destnet = inet\_addr(buf);

printf("请输入下一跳IP地址:\n");

scanf("%s", &buf);

rtableitem->nextip = inet\_addr(buf);

rtable.add(rtableitem);

}

else if (num == 2)

{

printf("请输入删除的序号：");

int index;

scanf("%d", &index);

rtable.Delete(index - 1);//将链表序号与实际输入序号统一

}

else if (num == 3)

{

rtable.print();

}

else

{

printf("输入有误！请重新输入!\n");

}

}

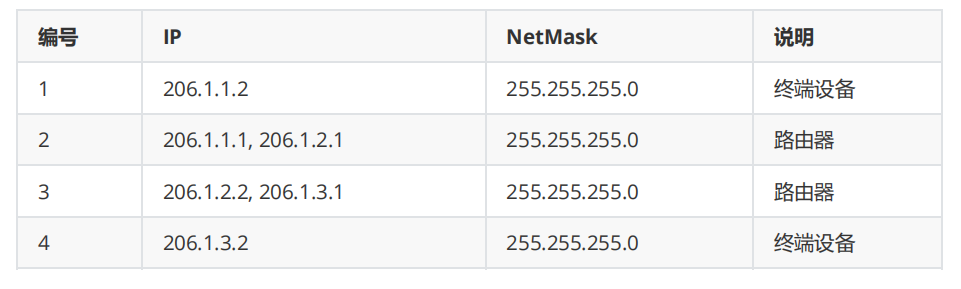
return 0;

}

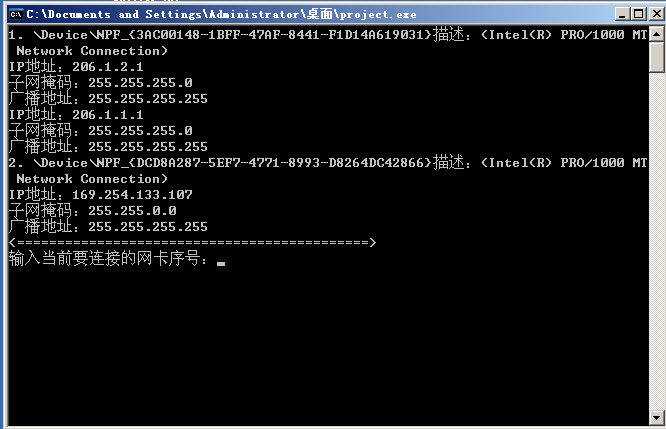
三、**实验结果**

打开四台虚拟机，在虚拟机3中设置路由表

route ADD 206.1.1.0 MASK 255.255.255.0 206.1.2.1



在虚拟机2运行自己的程序，初始页面如下



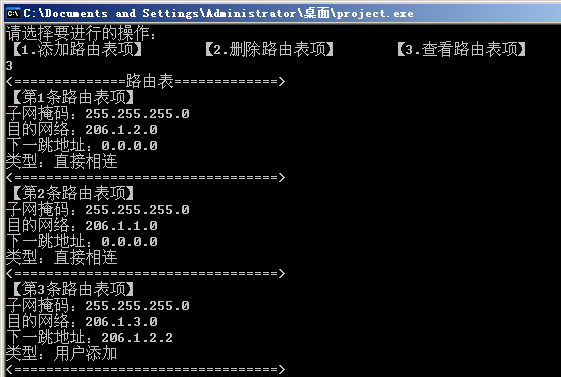
选择1号网卡设备打开并添加路由表子网：255.255.255.0 目的IP：206.1.3.0

下一跳IP：206.1.2.2

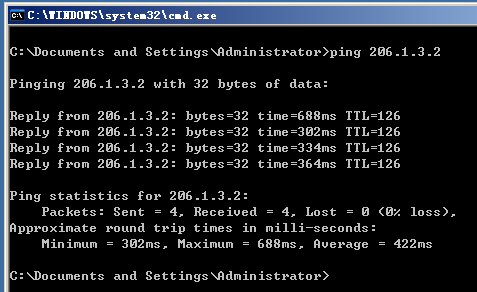




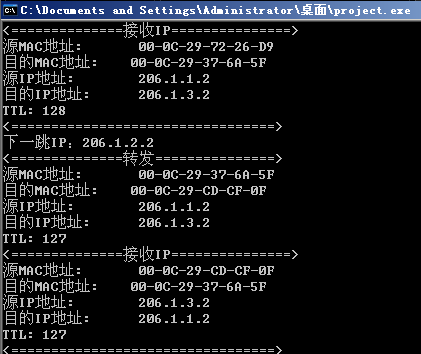
查看路由表发现里边有三条路由表，前两条是直接投递的初始化，第三条是手动添加的路由表。



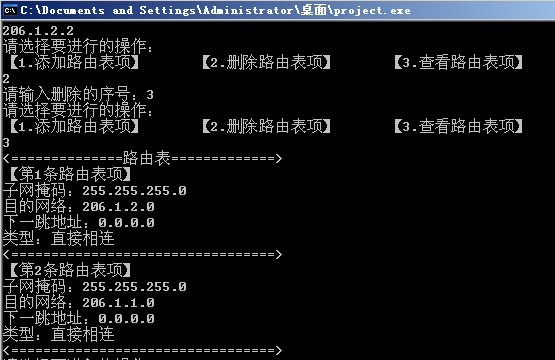
选择主机1ping主机4 ping 206.1.3.2,，发现可以ping通



路由器界面日志输出如下，可以输出数据包的内容以及转发的下一跳地址、接收转发等信息



删除路由表操作后查看路由表，发现第三条路由表已被删除



**四、实验总结**

对路由器的转发原理有了更深入的了解和认识，并对IP数据包和ARP数据包的构成有了清晰的认知。通过此次实验也对我的编程能力有了很大的锻炼，代码也还有很多不完善的地方，例如路由表使用的是链表结构，没有用哈希表等更方便高效的数据结构，ARP表也只是固定了一个大小为50的数组，没有对数组溢出进行有效的处置，有待提高。

实验过程也出现了生成的exe与Windows XP不兼容的情况出现，后来查询发现是inte\_ntop函数不被Windows Server2003的环境接受，更换成inte\_ntoa函数即可成功运行。

**五、附录**

GitHub链接：

https://github.com/Q-qiuqiu/Network-technology-and-Application/tree/main/lab5