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。

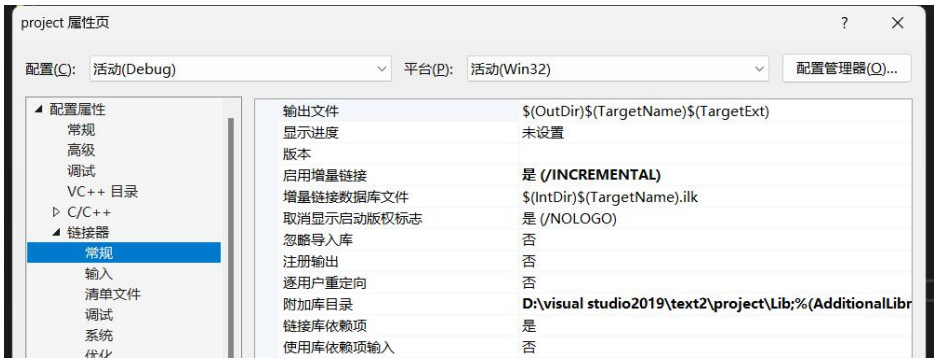
（5）完成的程序须通过现场测试，并在班（或小组）中展示和报告自己的设计思路、开发和实现过程、测试方法和过程。

二、实验过程

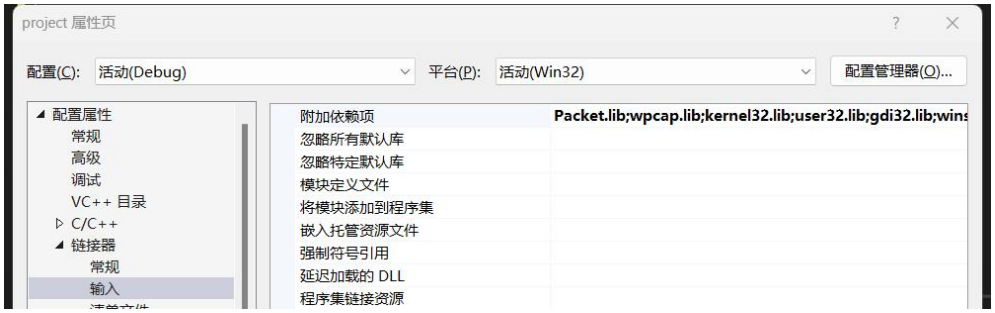
1.环境配置

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



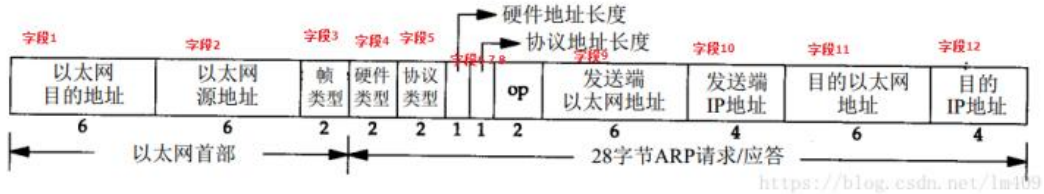
项目→属性→链接器→输入→附加依赖项：添加 Packet.lib;wpcap.lib;



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
```

4. 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);
                    }
                }
            }
        }
    }
}
```

（4）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;
}
```

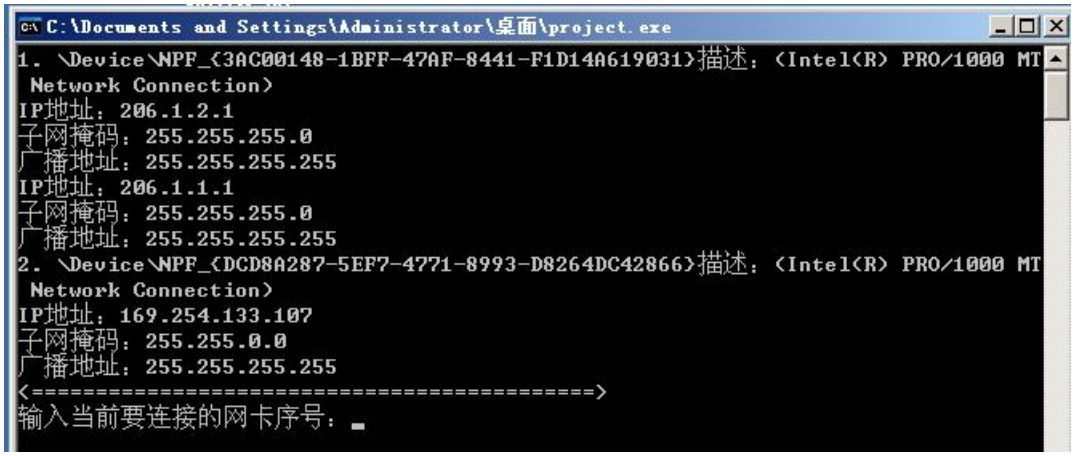
三、实验结果

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
```

编号	IP	NetMask	说明
1	206.1.1.2	255.255.255.0	终端设备
2	206.1.1.1, 206.1.2.1	255.255.255.0	路由器
3	206.1.2.2, 206.1.3.1	255.255.255.0	路由器
4	206.1.3.2	255.255.255.0	终端设备

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

```
<================================>
请选择要进行的操作：
【1.添加路由表项】        【2.删除路由表项】        【3.查看路由表项】
1
请输入子网掩码：
255.255.255.0
输入目的网络：
206.1.3.0
请输入下一跳IP地址：
206.1.2.2
请选择要进行的操作：
【1.添加路由表项】        【2.删除路由表项】        【3.查看路由表项】
```

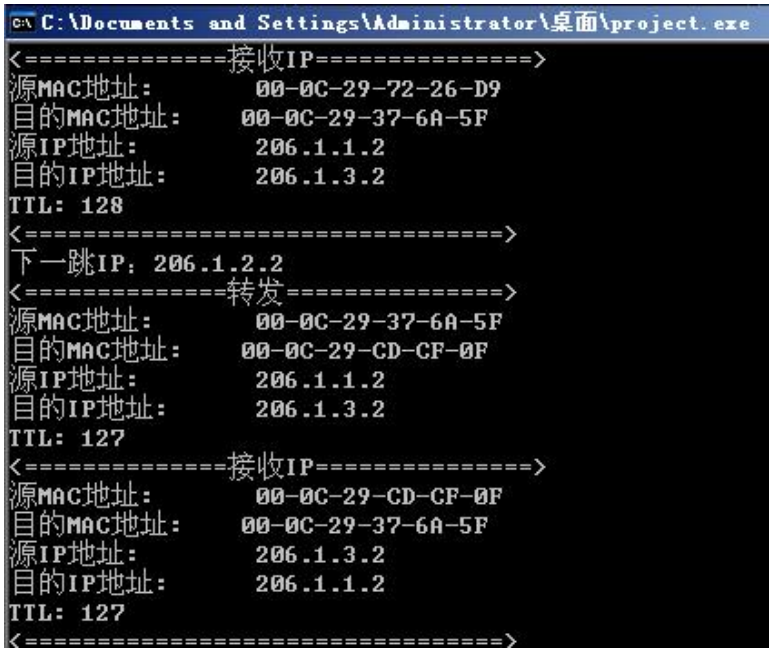
查看路由表发现里边有三条路由表，前两条是直接投递的初始化，第三条是手动添加的路由表。

```
C:\Documents and Settings\Administrator\桌面\project.exe
请选择要进行的操作：
【1.添加路由表项】        【2.删除路由表项】        【3.查看路由表项】
3
<==============路由表==============>
【第1条路由表项】
子网掩码：255.255.255.0
目的网络：206.1.2.0
下一跳地址：0.0.0.0
类型：直接相连
<================================>
【第2条路由表项】
子网掩码：255.255.255.0
目的网络：206.1.1.0
下一跳地址：0.0.0.0
类型：直接相连
<================================>
【第3条路由表项】
子网掩码：255.255.255.0
目的网络：206.1.3.0
下一跳地址：206.1.2.2
类型：用户添加
<================================>
```

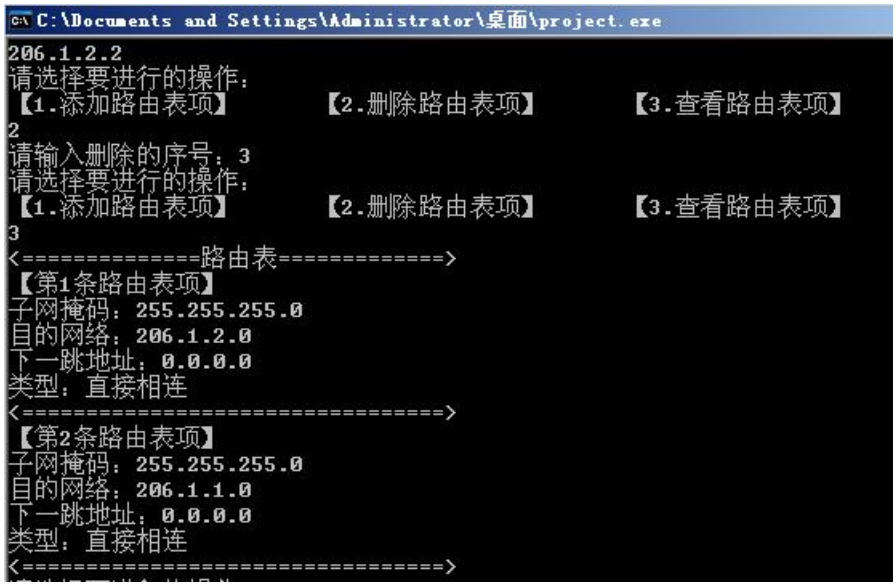
选择主机 1ping 主机 4 ping 206.1.3.2,，发现可以 ping 通

```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Administrator>ping 206.1.3.2

Pinging 206.1.3.2 with 32 bytes of data:

Reply from 206.1.3.2: bytes=32 time=688ms TTL=126
Reply from 206.1.3.2: bytes=32 time=302ms TTL=126
Reply from 206.1.3.2: bytes=32 time=334ms TTL=126
Reply from 206.1.3.2: bytes=32 time=364ms TTL=126

Ping statistics for 206.1.3.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 302ms, Maximum = 688ms, Average = 422ms

C:\Documents and Settings\Administrator>
```

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