南开大学

网络空间安全学院学院

网络技术与应用课程报告

第 **5** 次实验报告

学号：2011428

姓名：王天行年级：

2020 级

专业：密码科学与技术

2022 年 12 月 24 日

第 1 节实验内容说明

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。

评分原则：前期准备 5，实验过程 35，程序及规范性 20，结果展示 20，实验报告 20，总分 100

第 2 节实验准备

初始化

- 选择并打开网口
- 获得本机 ip 地址和 mac 地址
- 初始化路由表并添加初始表项

报文转发时的组装

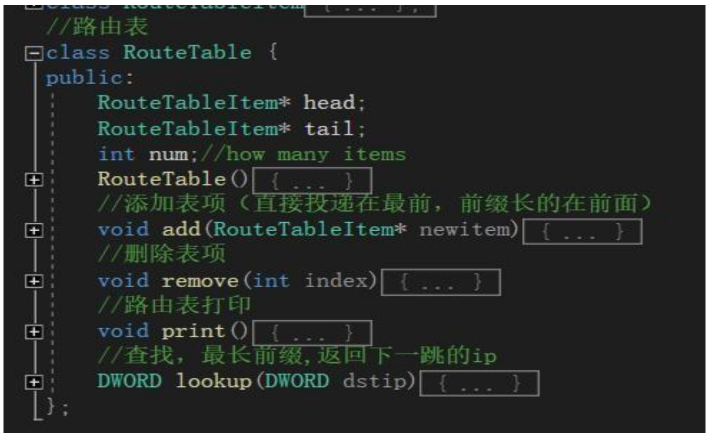
- 数据帧首部
  - SrcMac=路由器自己的 MAC
  - DesMac=下一跳的 MAC
- IP 报首部
  - 若 TTL=0，则返回 ICMP 报文，不转发
  - 若 TTL>0，则 TTL-1，重新计算校验和

转发流程

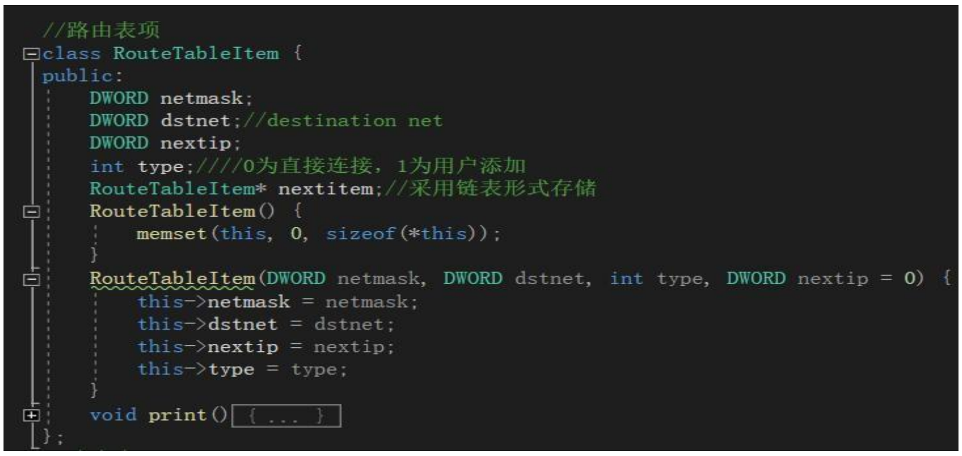
- 收到数据包，判断该数据包为 ip 数据包，且消息目的 MAC 地址为自己的 MAC
- 查看目的 IP，如果指向自己则接收，不是则转发
- 查找路由表对应的下一跳 ip，若没找到则丢弃
- 查找 ARP 表中是否能找到对应的 MAC 地址，若没有则发送 ARP
- 请求。

使用到的数据结构

路由表

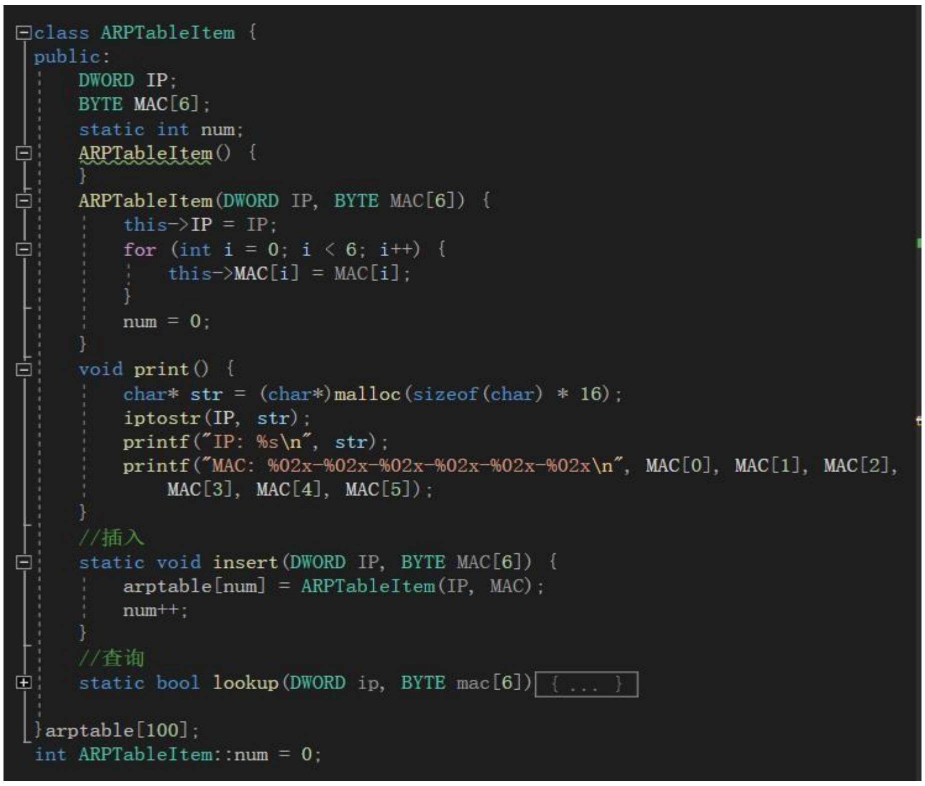


```
//路由表
class RouteTable {
public:
    RouteTableItem* head;
    RouteTableItem* tail;
    int num;//how many items
    RouteTable() { ... }
    //添加表项（直接投递在最前，前缀长的在前面）
    void add(RouteTableItem* newitem) { ... }
    //删除表项
    void remove(int index) { ... }
    //路由表打印
    void print() { ... }
    //查找，最长前缀,返回下一跳的ip
    DWORD lookup(DWORD dstip) { ... }
};
```

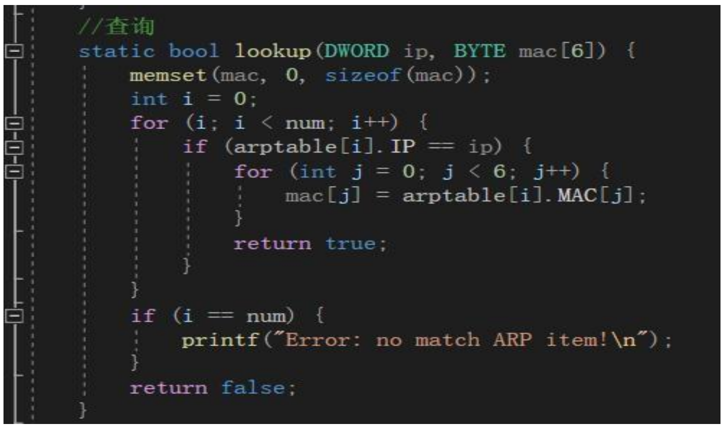


```
//路由表项
class RouteTableItem {
public:
    DWORD netmask;
    DWORD dstnet;//destination net
    DWORD nextip;
    int type;////0为直接连接, 1为用户添加
    RouteTableItem* nextitem;//采用链表形式存储
    RouteTableItem() {
        memset(this, 0, sizeof(*this));
    }
    RouteTableItem(DWORD netmask, DWORD dstnet, int type, DWORD nextip = 0) {
        this->netmask = netmask;
        this->dstnet = dstnet;
        this->nextip = nextip;
        this->type = type;
    }
    void print(){ ... }
};
```

ARP 缓存表



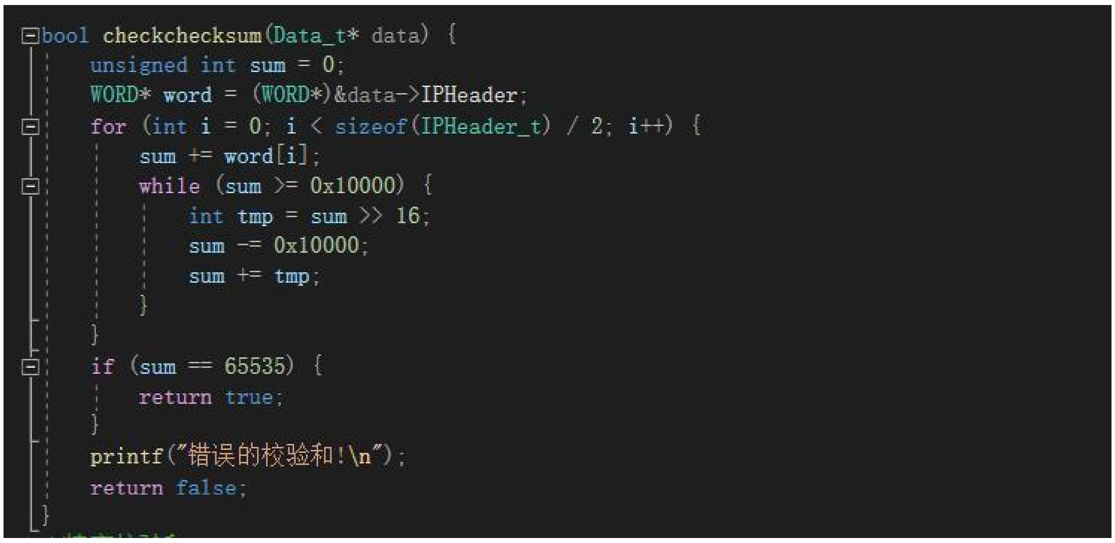
```
class ARPTableItem {
public:
    DWORD IP;
    BYTE MAC[6];
    static int num;
    ARPTableItem() {
    }
    ARPTableItem(DWORD IP, BYTE MAC[6]) {
        this->IP = IP;
        for (int i = 0; i < 6; i++) {
            this->MAC[i] = MAC[i];
        }
        num = 0;
    }
    void print() {
        char* str = (char*)malloc(sizeof(char) * 16);
        iptostr(IP, str);
        printf("IP: %s\n", str);
        printf("MAC: %02x-%02x-%02x-%02x-%02x-%02x\n", MAC[0], MAC[1], MAC[2],
            MAC[3], MAC[4], MAC[5]);
    }
    //插入
    static void insert(DWORD IP, BYTE MAC[6]) {
        arptable[num] = ARPTableItem(IP, MAC);
        num++;
    }
    //查询
    static bool lookup(DWORD ip, BYTE mac[6]){ ... }

}arptable[100];
int ARPTableItem::num = 0;
```



```
    //查询
    static bool lookup(DWORD ip, BYTE mac[6]) {
        memset(mac, 0, sizeof(mac));
        int i = 0;
        for (i; i < num; i++) {
            if (arptable[i].IP == ip) {
                for (int j = 0; j < 6; j++) {
                    mac[j] = arptable[i].MAC[j];
                }
                return true;
            }
        }
        if (i == num) {
            printf("Error: no match ARP item!\n");
        }
        return false;
    }
```

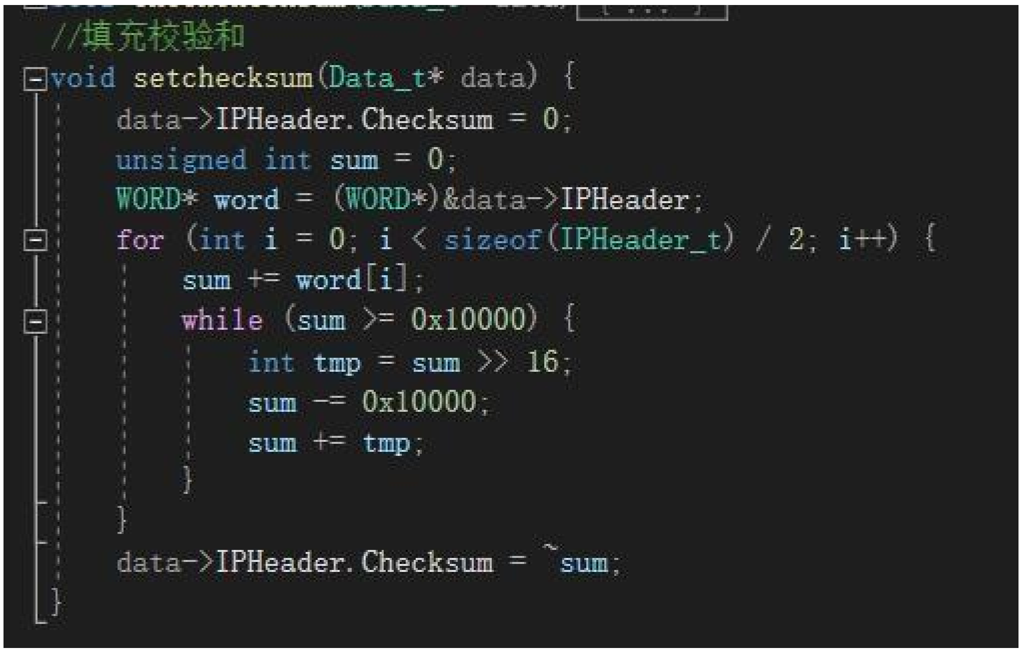
校验和

检查校验和



```
bool checkchecksum(Data_t* data) {
    unsigned int sum = 0;
    WORD* word = (WORD*)&data->IPHeader;
    for (int i = 0; i < sizeof(IPHeader_t) / 2; i++) {
        sum += word[i];
        while (sum >= 0x10000) {
            int tmp = sum >> 16;
            sum -= 0x10000;
            sum += tmp;
        }
    }
    if (sum == 65535) {
        return true;
    }
    printf("错误的校验和!\n");
    return false;
}
```

填充校验和

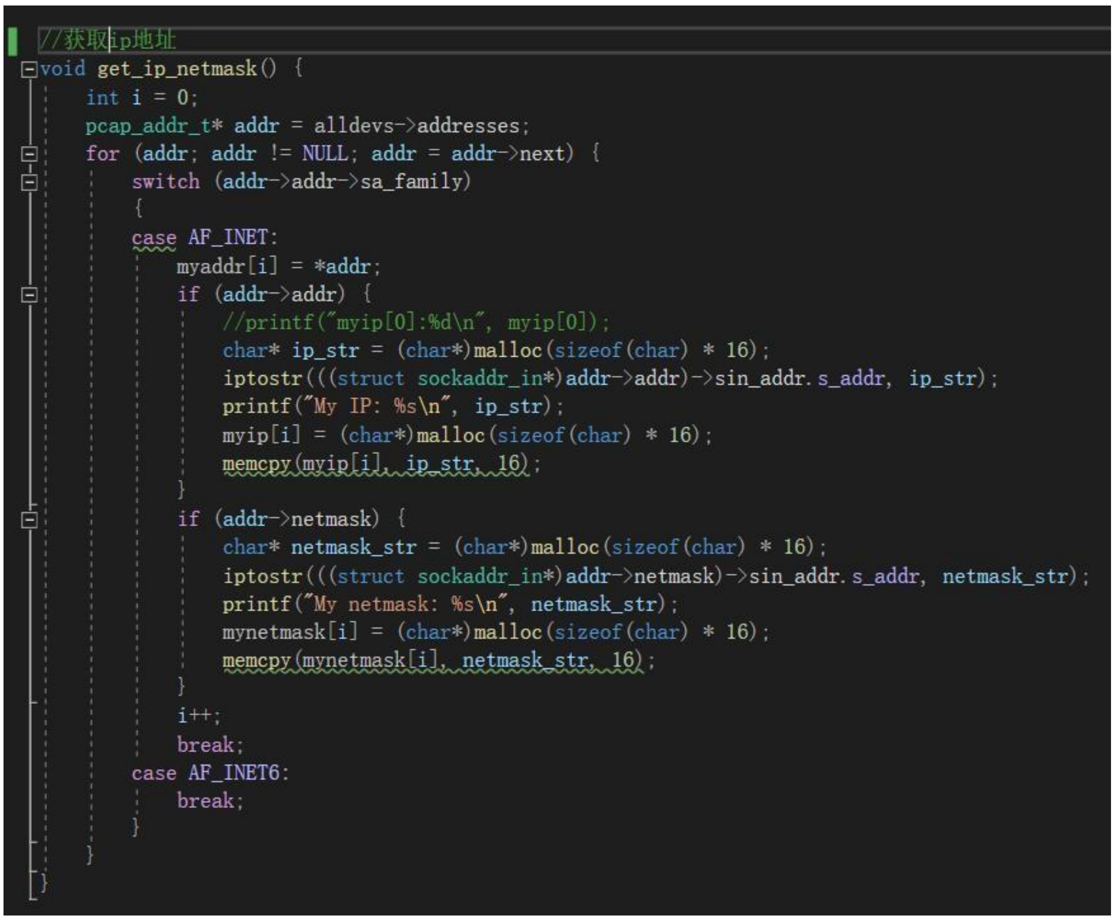


```
 //填充校验和
void setchecksum(Data_t* data) {
    data->IPHeader.Checksum = 0;
    unsigned int sum = 0;
    WORD* word = (WORD*)&data->IPHeader;
    for (int i = 0; i < sizeof(IPHeader_t) / 2; i++) {
        sum += word[i];
        while (sum >= 0x10000) {
            int tmp = sum >> 16;
            sum -= 0x10000;
            sum += tmp;
        }
    }
    data->IPHeader.Checksum = ~sum;
}
```

第 **3** 节实验过程

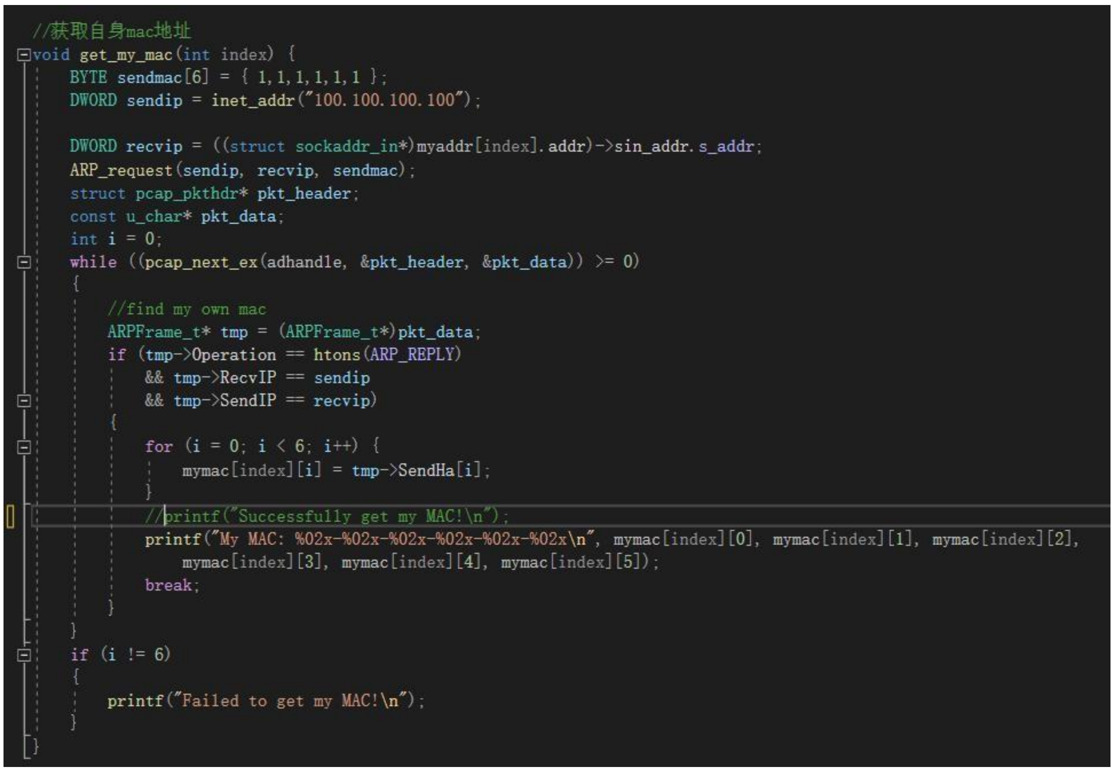
实验代码

打开网卡获取双 **IP**



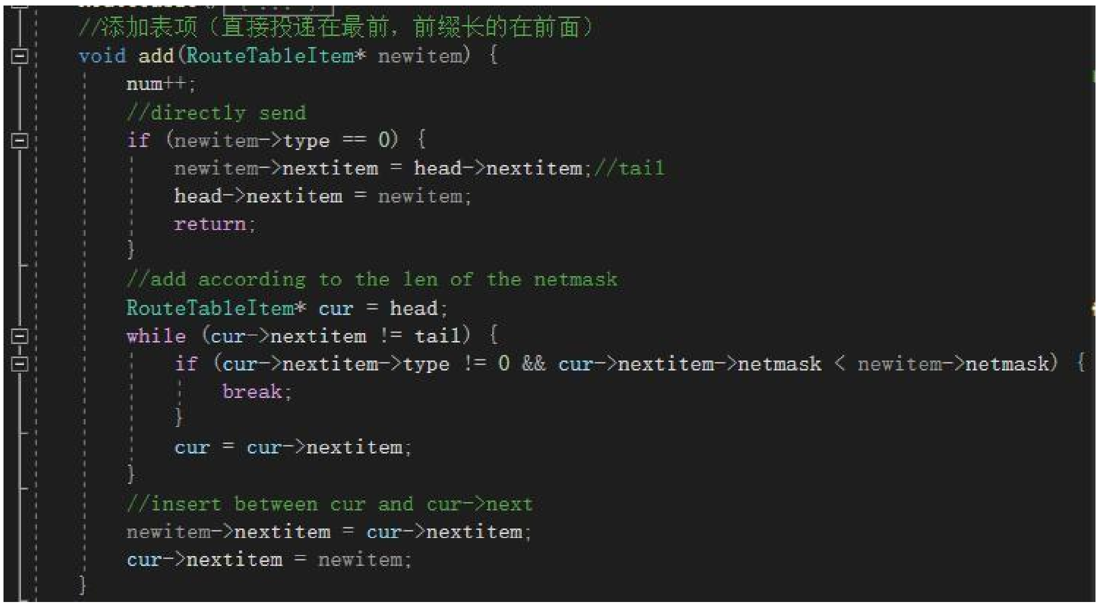
```c
//获取ip地址
void get_ip_netmask() {
    int i = 0;
    pcap_addr_t* addr = alldevs->addresses;
    for (addr; addr != NULL; addr = addr->next) {
        switch (addr->addr->sa_family)
        {
        case AF_INET:
            myaddr[i] = *addr;
            if (addr->addr) {
                //printf("myip[0]:%d\n", myip[0]);
                char* ip_str = (char*)malloc(sizeof(char) * 16);
                iptostr(((struct sockaddr_in*)addr->addr)->sin_addr.s_addr, ip_str);
                printf("My IP: %s\n", ip_str);
                myip[i] = (char*)malloc(sizeof(char) * 16);
                memcpy(myip[i], ip_str, 16);
            }
            if (addr->netmask) {
                char* netmask_str = (char*)malloc(sizeof(char) * 16);
                iptostr(((struct sockaddr_in*)addr->netmask)->sin_addr.s_addr, netmask_str);
                printf("My netmask: %s\n", netmask_str);
                mynetmask[i] = (char*)malloc(sizeof(char) * 16);
                memcpy(mynetmask[i], netmask_str, 16);
            }
            i++;
            break;
        case AF_INET6:
            break;
        }
    }
}
```

伪造 **ARP** 报文获取本机 **MAC**

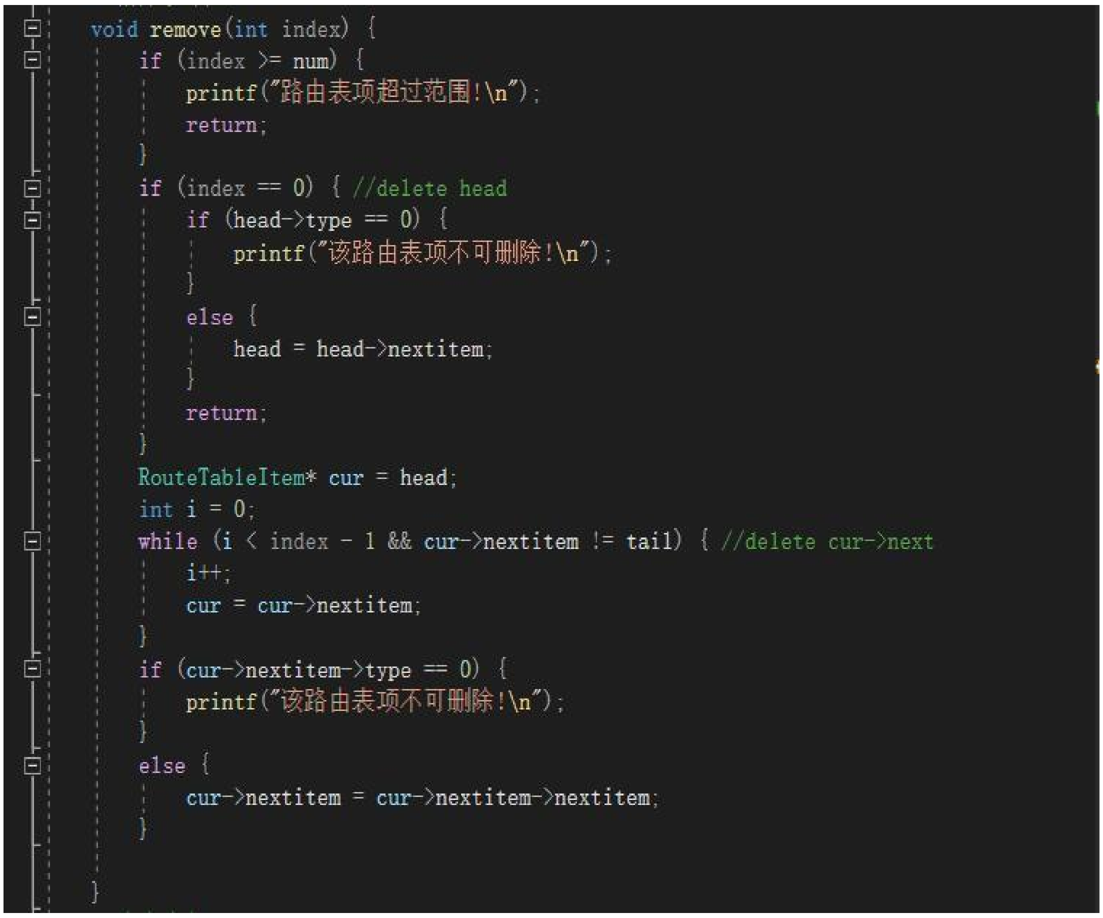


```c
//获取自身mac地址
void get_my_mac(int index) {
    BYTE sendmac[6] = { 1,1,1,1,1,1 };
    DWORD sendip = inet_addr("100.100.100.100");

    DWORD recvip = ((struct sockaddr_in*)myaddr[index].addr)->sin_addr.s_addr;
    ARP_request(sendip, recvip, sendmac);
    struct pcap_pkthdr* pkt_header;
    const u_char* pkt_data;
    int i = 0;
    while ((pcap_next_ex(adhandle, &pkt_header, &pkt_data)) >= 0)
    {
        //find my own mac
        ARPFrame_t* tmp = (ARPFrame_t*)pkt_data;
        if (tmp->Operation == htons(ARP_REPLY)
            && tmp->RecvIP == sendip
            && tmp->SendIP == recvip)
        {
            for (i = 0; i < 6; i++) {
                mymac[index][i] = tmp->SendHa[i];
            }
            //printf("Successfully get my MAC!\n");
            printf("My MAC: %02x-%02x-%02x-%02x-%02x-%02x\n", mymac[index][0], mymac[index][1], mymac[index][2],
                mymac[index][3], mymac[index][4], mymac[index][5]);
            break;
        }
    }
    if (i != 6)
    {
        printf("Failed to get my MAC!\n");
    }
}
```

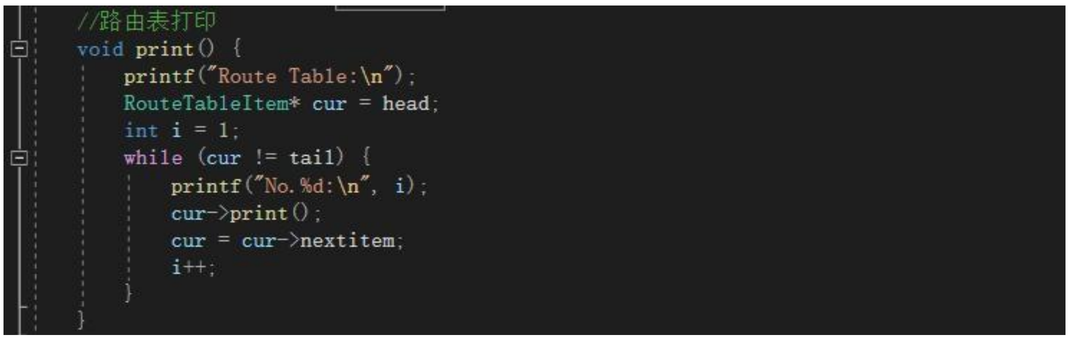
自动添加默认路由表项，手动添加&删除路由表项，显示路由表



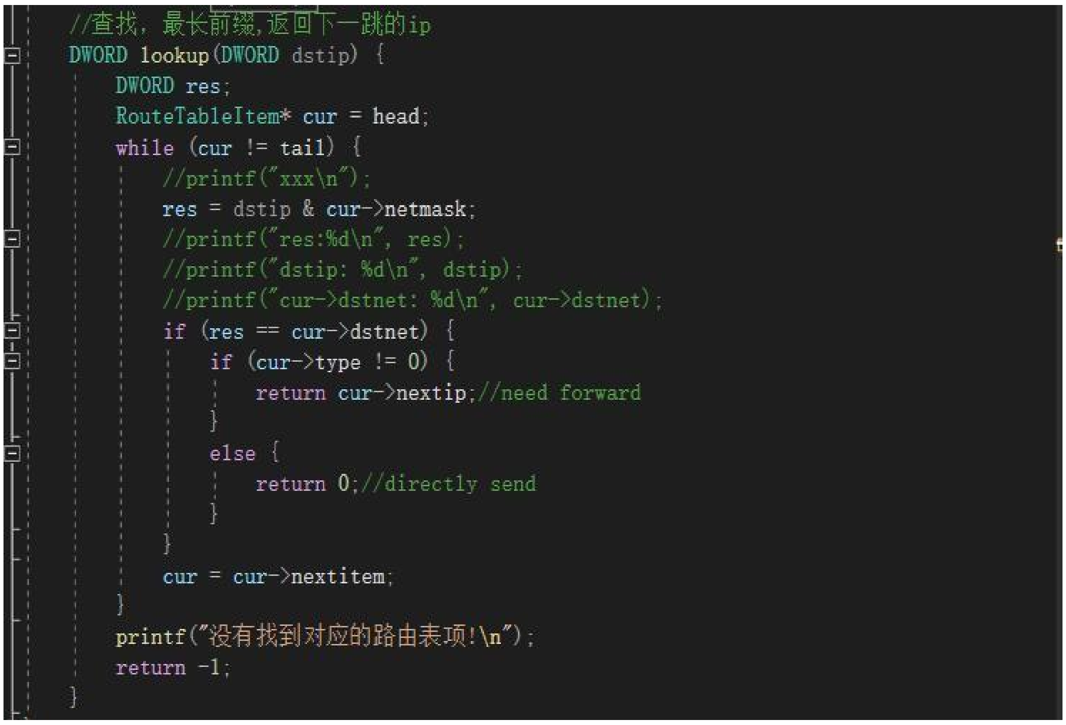
```
    //添加表项（直接投递在最前，前缀长的在前面）
    void add(RouteTableItem* newitem) {
        num++;
        //directly send
        if (newitem->type == 0) {
            newitem->nextitem = head->nextitem;//tail
            head->nextitem = newitem;
            return;
        }
        //add according to the len of the netmask
        RouteTableItem* cur = head;
        while (cur->nextitem != tail) {
            if (cur->nextitem->type != 0 && cur->nextitem->netmask < newitem->netmask) {
                break;
            }
            cur = cur->nextitem;
        }
        //insert between cur and cur->next
        newitem->nextitem = cur->nextitem;
        cur->nextitem = newitem;
    }
```



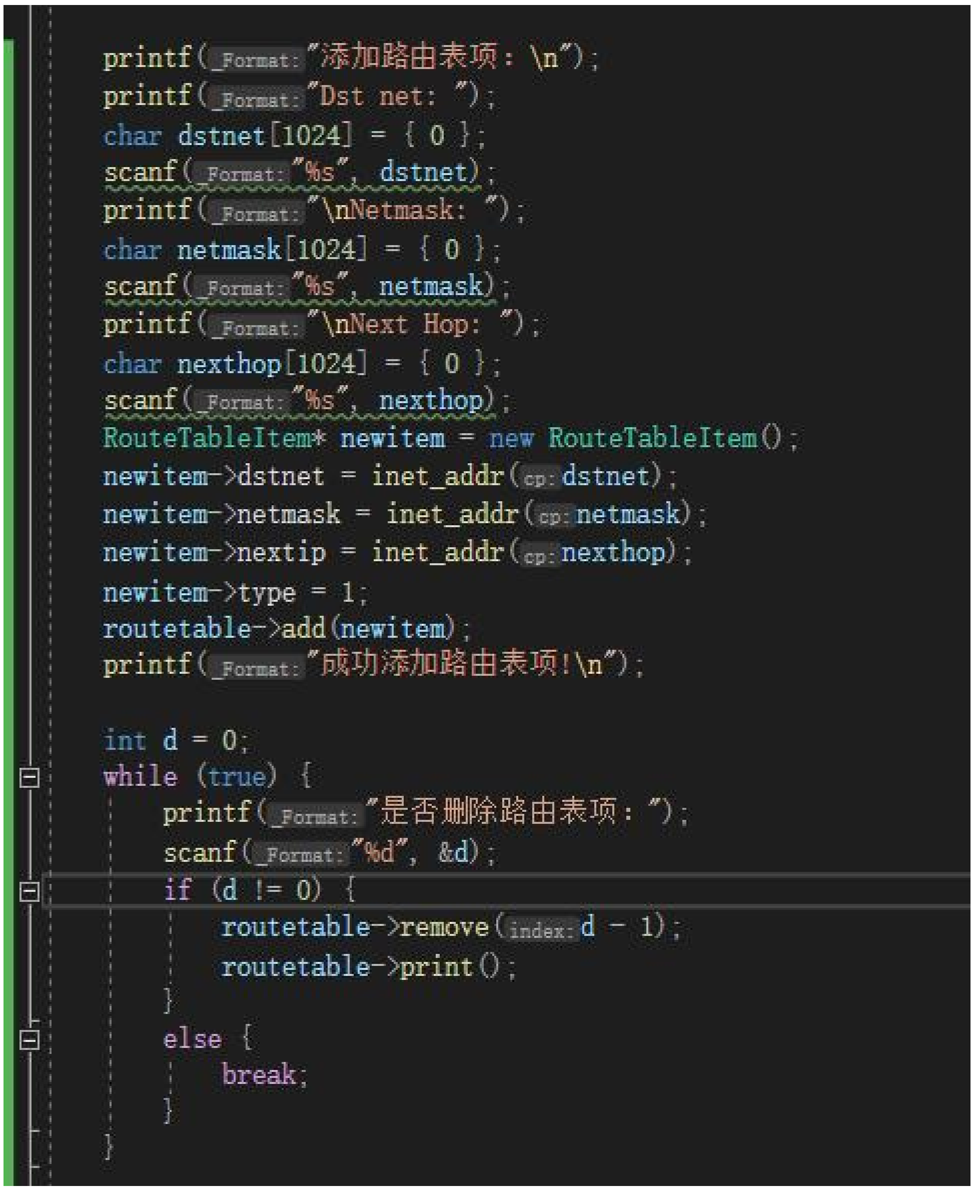
```
    void remove(int index) {
        if (index >= num) {
            printf("路由表项超过范围!\n");
            return;
        }
        if (index == 0) { //delete head
            if (head->type == 0) {
                printf("该路由表项不可删除!\n");
            }
            else {
                head = head->nextitem;
            }
            return;
        }
        RouteTableItem* cur = head;
        int i = 0;
        while (i < index - 1 && cur->nextitem != tail) { //delete cur->next
            i++;
            cur = cur->nextitem;
        }
        if (cur->nextitem->type == 0) {
            printf("该路由表项不可删除!\n");
        }
        else {
            cur->nextitem = cur->nextitem->nextitem;
        }

    }
```



```
//路由表打印
void print() {
    printf("Route Table:\n");
    RouteTableItem* cur = head;
    int i = 1;
    while (cur != tail) {
        printf("No.%d:\n", i);
        cur->print();
        cur = cur->nextitem;
        i++;
    }
}
```

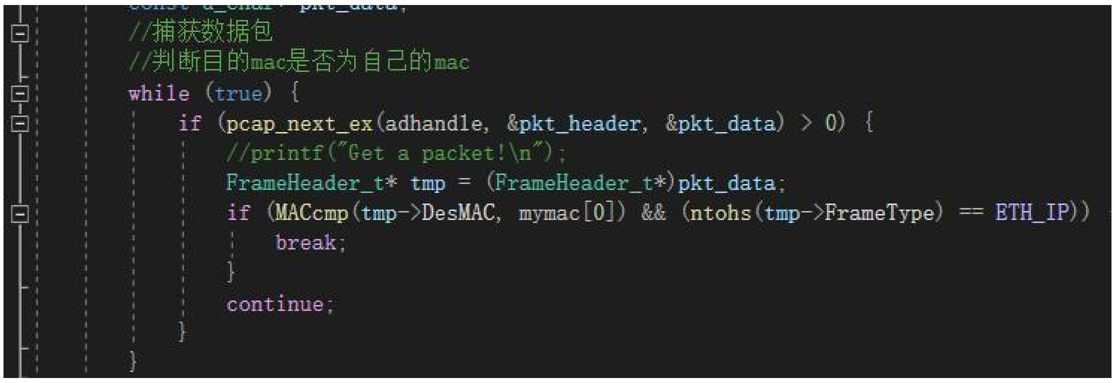


```
//查找，最长前缀,返回下一跳的ip
DWORD lookup(DWORD dstip) {
    DWORD res;
    RouteTableItem* cur = head;
    while (cur != tail) {
        //printf("xxx\n");
        res = dstip & cur->netmask;
        //printf("res:%d\n", res);
        //printf("dstip: %d\n", dstip);
        //printf("cur->dstnet: %d\n", cur->dstnet);
        if (res == cur->dstnet) {
            if (cur->type != 0) {
                return cur->nextip;//need forward
            }
            else {
                return 0;//directly send
            }
        }
        cur = cur->nextitem;
    }
    printf("没有找到对应的路由表项!\n");
    return -1;
}
```



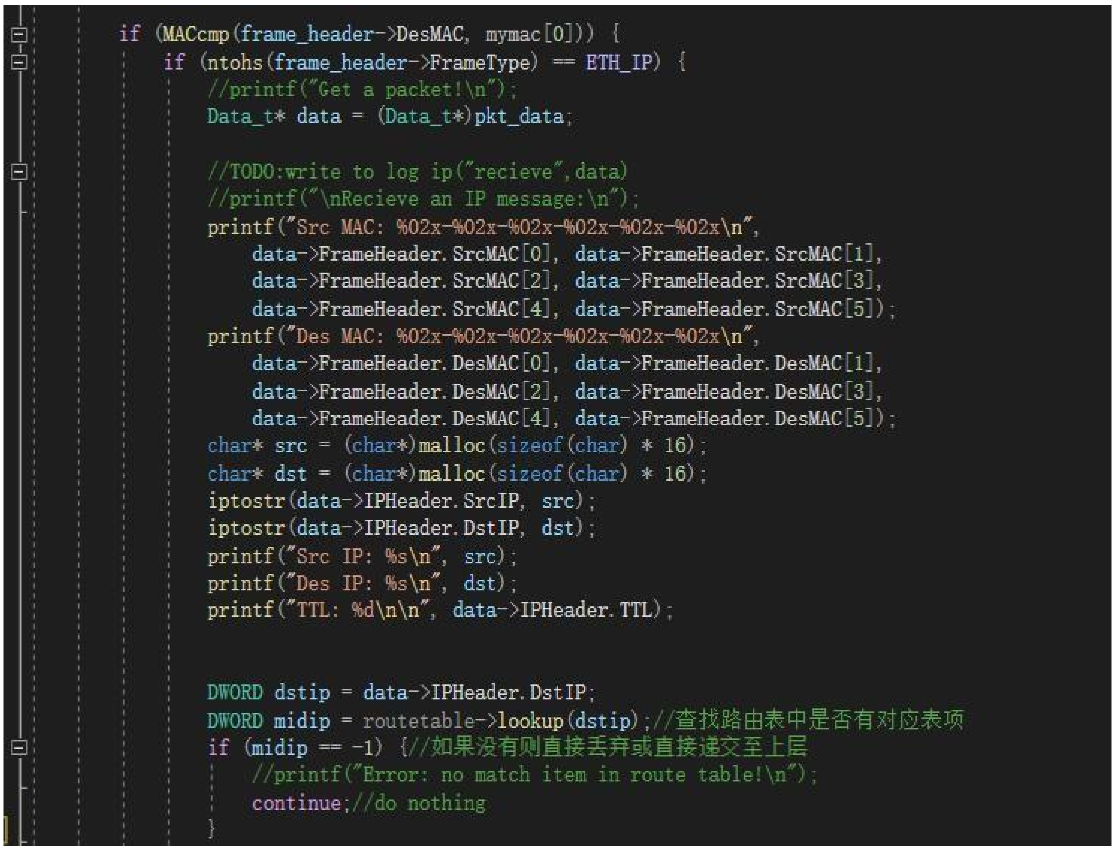
```
    printf(_Format:"添加路由表项: \n");
    printf(_Format:"Dst net: ");
    char dstnet[1024] = { 0 };
    scanf(_Format:"%s", dstnet);
    printf(_Format:"\nNetmask: ");
    char netmask[1024] = { 0 };
    scanf(_Format:"%s", netmask);
    printf(_Format:"\nNext Hop: ");
    char nexthop[1024] = { 0 };
    scanf(_Format:"%s", nexthop);
    RouteTableItem* newitem = new RouteTableItem();
    newitem->dstnet = inet_addr(cp:dstnet);
    newitem->netmask = inet_addr(cp:netmask);
    newitem->nextip = inet_addr(cp:nexthop);
    newitem->type = 1;
    routetable->add(newitem);
    printf(_Format:"成功添加路由表项!\n");

    int d = 0;
    while (true) {
        printf(_Format:"是否删除路由表项: ");
        scanf(_Format:"%d", &d);
        if (d != 0) {
            routetable->remove(index:d - 1);
            routetable->print();
        }
        else {
            break;
        }
    }
```

捕获报文的过滤条件（ARP & IP）

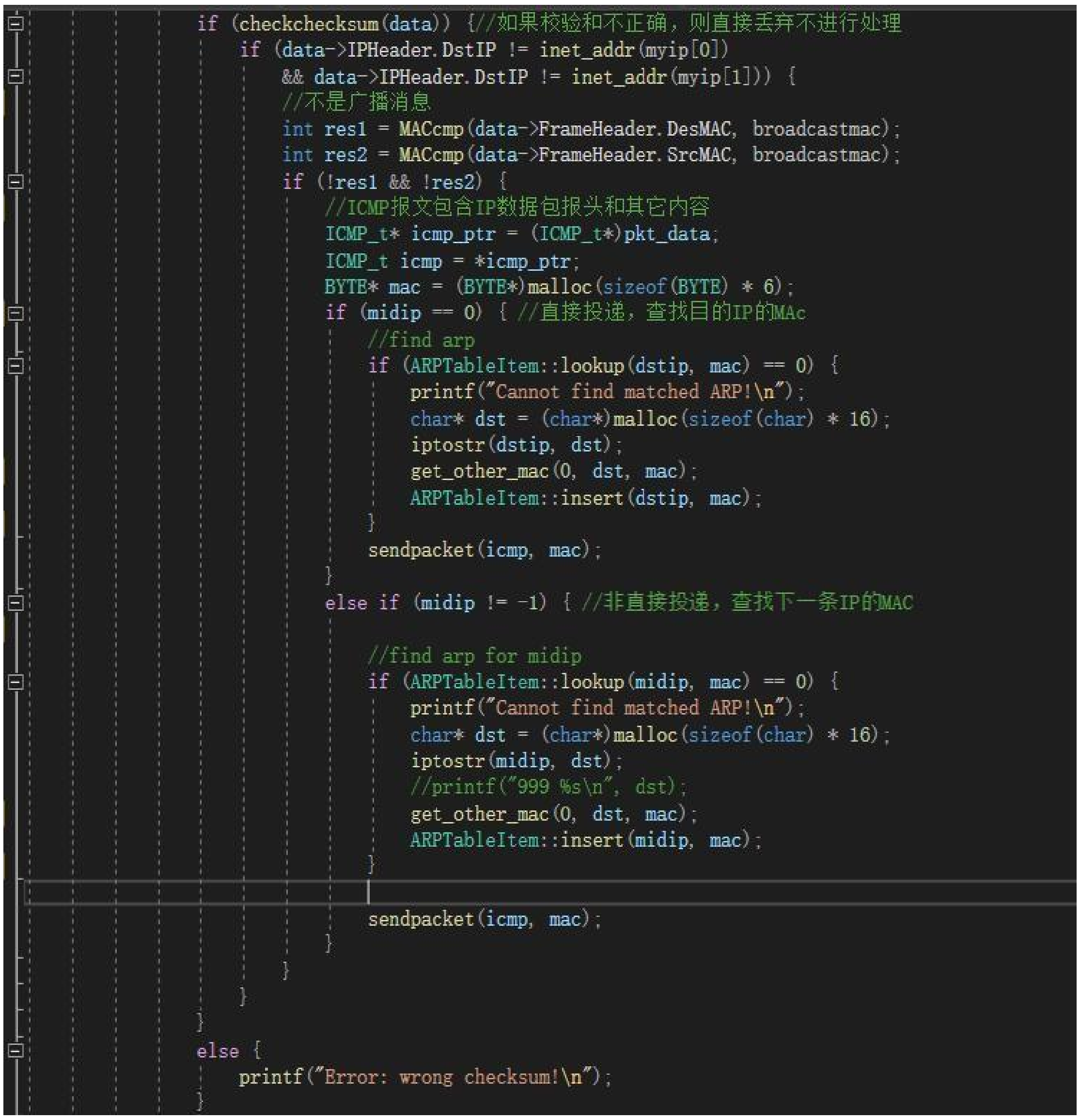


```
        //捕获数据包
        //判断目的mac是否为自己的mac
        while (true) {
            if (pcap_next_ex(adhandle, &pkt_header, &pkt_data) > 0) {
                //printf("Get a packet!\n");
                FrameHeader_t* tmp = (FrameHeader_t*)pkt_data;
                if (MACcmp(tmp->DesMAC, mymac[0]) && (ntohs(tmp->FrameType) == ETH_IP))
                    break;
                }
                continue;
            }
        }
```

捕获 IP 报文的处理



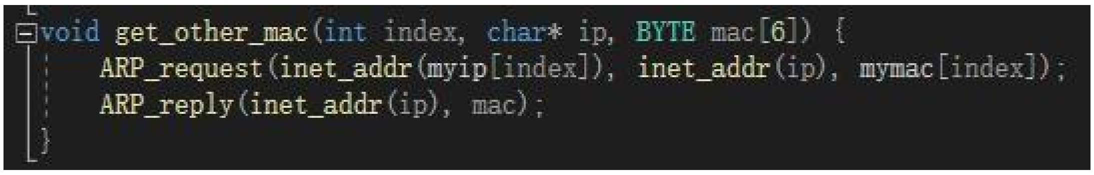
```
        if (MACcmp(frame_header->DesMAC, mymac[0])) {
            if (ntohs(frame_header->FrameType) == ETH_IP) {
                //printf("Get a packet!\n");
                Data_t* data = (Data_t*)pkt_data;

                //TODO:write to log ip("recieve",data)
                //printf("\nRecieve an IP message:\n");
                printf("Src MAC: %02x-%02x-%02x-%02x-%02x-%02x\n",
                    data->FrameHeader.SrcMAC[0], data->FrameHeader.SrcMAC[1],
                    data->FrameHeader.SrcMAC[2], data->FrameHeader.SrcMAC[3],
                    data->FrameHeader.SrcMAC[4], data->FrameHeader.SrcMAC[5]);
                printf("Des MAC: %02x-%02x-%02x-%02x-%02x-%02x\n",
                    data->FrameHeader.DesMAC[0], data->FrameHeader.DesMAC[1],
                    data->FrameHeader.DesMAC[2], data->FrameHeader.DesMAC[3],
                    data->FrameHeader.DesMAC[4], data->FrameHeader.DesMAC[5]);
                char* src = (char*)malloc(sizeof(char) * 16);
                char* dst = (char*)malloc(sizeof(char) * 16);
                iptostr(data->IPHeader.SrcIP, src);
                iptostr(data->IPHeader.DstIP, dst);
                printf("Src IP: %s\n", src);
                printf("Des IP: %s\n", dst);
                printf("TTL: %d\n\n", data->IPHeader.TTL);


                DWORD dstip = data->IPHeader.DstIP;
                DWORD midip = routetable->lookup(dstip);//查找路由表中是否有对应表项
                if (midip == -1) {//如果没有则直接丢弃或直接递交至上层
                    //printf("Error: no match item in route table!\n");
                    continue;//do nothing
                }
```



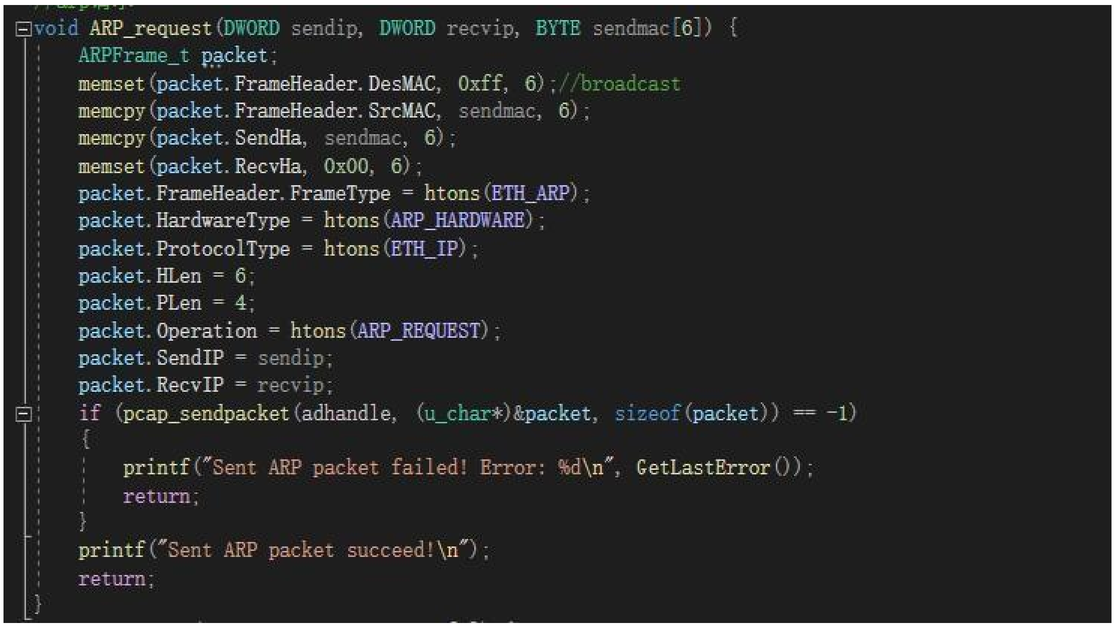
```
if (checkchecksum(data)) {//如果校验和不正确，则直接丢弃不进行处理
    if (data->IPHeader.DstIP != inet_addr(myip[0])
        && data->IPHeader.DstIP != inet_addr(myip[1])) {
        //不是广播消息
        int res1 = MACcmp(data->FrameHeader.DesMAC, broadcastmac);
        int res2 = MACcmp(data->FrameHeader.SrcMAC, broadcastmac);
        if (!res1 && !res2) {
            //ICMP报文包含IP数据包报头和其它内容
            ICMP_t* icmp_ptr = (ICMP_t*)pkt_data;
            ICMP_t icmp = *icmp_ptr;
            BYTE* mac = (BYTE*)malloc(sizeof(BYTE) * 6);
            if (midip == 0) { //直接投递，查找目的IP的MAc
                //find arp
                if (ARPTableItem::lookup(dstip, mac) == 0) {
                    printf("Cannot find matched ARP!\n");
                    char* dst = (char*)malloc(sizeof(char) * 16);
                    iptostr(dstip, dst);
                    get_other_mac(0, dst, mac);
                    ARPTableItem::insert(dstip, mac);
                }
                sendpacket(icmp, mac);
            }
            else if (midip != -1) { //非直接投递，查找下一条IP的MAC

                //find arp for midip
                if (ARPTableItem::lookup(midip, mac) == 0) {
                    printf("Cannot find matched ARP!\n");
                    char* dst = (char*)malloc(sizeof(char) * 16);
                    iptostr(midip, dst);
                    //printf("999 %s\n", dst);
                    get_other_mac(0, dst, mac);
                    ARPTableItem::insert(midip, mac);
                }

                sendpacket(icmp, mac);
            }
        }
    }
}
else {
    printf("Error: wrong checksum!\n");
}
```

捕获 ARP 报文的处理（ARP 请求）

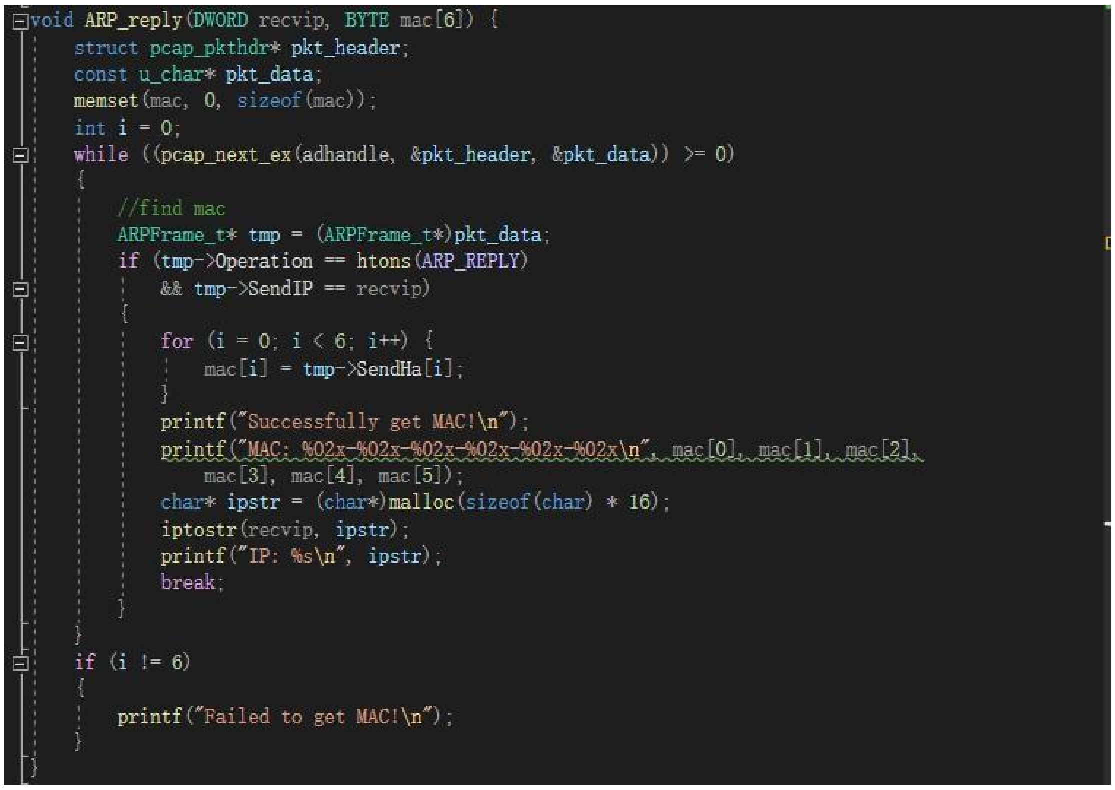
对于不是需要的 ARP 报文，进行忽视，只在进行 ARP 请求是获取 ARP 报文。



```
void get_other_mac(int index, char* ip, BYTE mac[6]) {
    ARP_request(inet_addr(myip[index]), inet_addr(ip), mymac[index]);
    ARP_reply(inet_addr(ip), mac);
}
```



```
void ARP_request(DWORD sendip, DWORD recvip, BYTE sendmac[6]) {
    ARPFrame_t packet;
    memset(packet.FrameHeader.DesMAC, 0xff, 6);//broadcast
    memcpy(packet.FrameHeader.SrcMAC, sendmac, 6);
    memcpy(packet.SendHa, sendmac, 6);
    memset(packet.RecvHa, 0x00, 6);
    packet.FrameHeader.FrameType = htons(ETH_ARP);
    packet.HardwareType = htons(ARP_HARDWARE);
    packet.ProtocolType = htons(ETH_IP);
    packet.HLen = 6;
    packet.PLen = 4;
    packet.Operation = htons(ARP_REQUEST);
    packet.SendIP = sendip;
    packet.RecvIP = recvip;
    if (pcap_sendpacket(adhandle, (u_char*)&packet, sizeof(packet)) == -1)
    {
        printf("Sent ARP packet failed! Error: %d\n", GetLastError());
        return;
    }
    printf("Sent ARP packet succeed!\n");
    return;
}
```

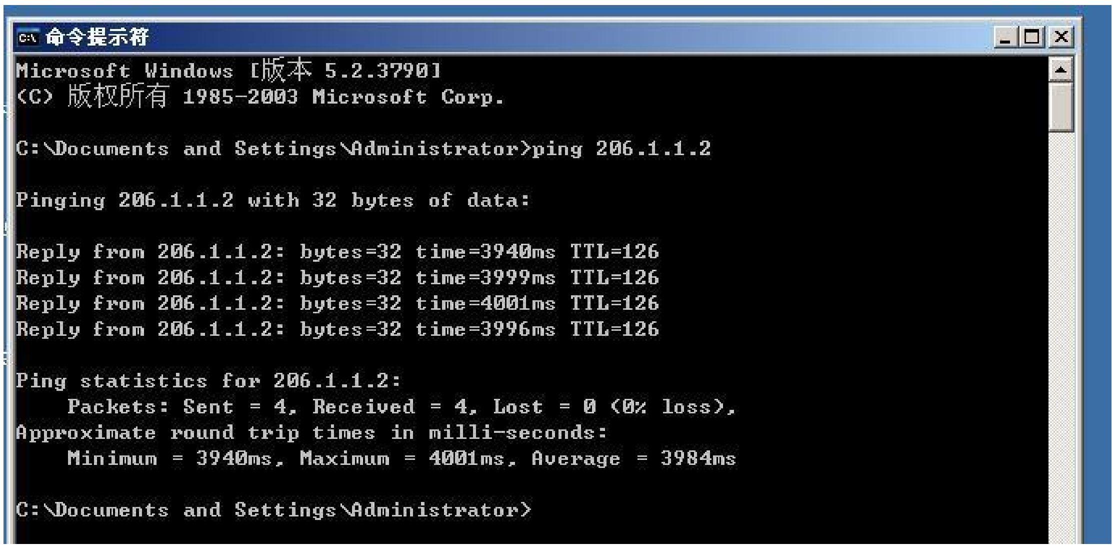


```
void ARP_reply(DWORD recvip, BYTE mac[6]) {
    struct pcap_pkthdr* pkt_header;
    const u_char* pkt_data;
    memset(mac, 0, sizeof(mac));
    int i = 0;
    while ((pcap_next_ex(adhandle, &pkt_header, &pkt_data)) >= 0)
    {
        //find mac
        ARPFrame_t* tmp = (ARPFrame_t*)pkt_data;
        if (tmp->Operation == htons(ARP_REPLY)
            && tmp->SendIP == recvip)
        {
            for (i = 0; i < 6; i++) {
                mac[i] = tmp->SendHa[i];
            }
            printf("Successfully get MAC!\n");
            printf("MAC: %02x-%02x-%02x-%02x-%02x-%02x\n", mac[0], mac[1], mac[2],
                mac[3], mac[4], mac[5]);
            char* ipstr = (char*)malloc(sizeof(char) * 16);
            iptostr(recvip, ipstr);
            printf("IP: %s\n", ipstr);
            break;
        }
    }
    if (i != 6)
    {
        printf("Failed to get MAC!\n");
    }
}
```

实验结果

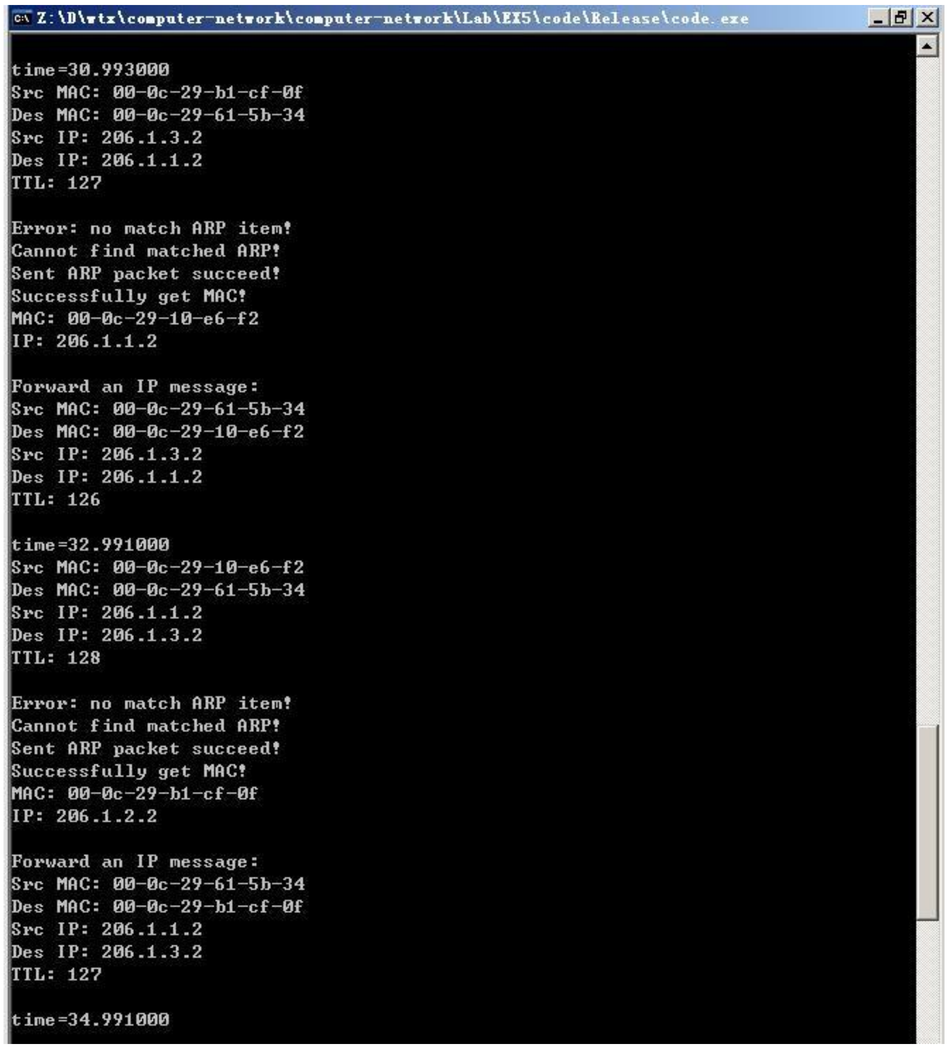
主机 B 可以 ping 通主机 A：

路机 A 可以 ping 通主机 B：



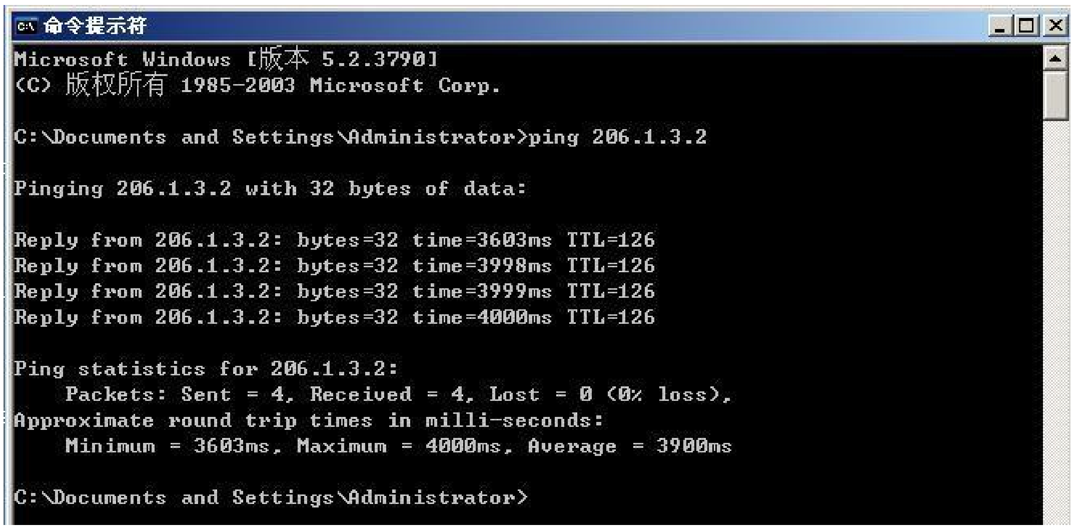
```
命令提示符
Microsoft Windows [版本 5.2.3790]
(C) 版权所有 1985-2003 Microsoft Corp.

C:\Documents and Settings\Administrator>ping 206.1.1.2

Pinging 206.1.1.2 with 32 bytes of data:

Reply from 206.1.1.2: bytes=32 time=3940ms TTL=126
Reply from 206.1.1.2: bytes=32 time=3999ms TTL=126
Reply from 206.1.1.2: bytes=32 time=4001ms TTL=126
Reply from 206.1.1.2: bytes=32 time=3996ms TTL=126

Ping statistics for 206.1.1.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 3940ms, Maximum = 4001ms, Average = 3984ms

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

路由程序日志显示：

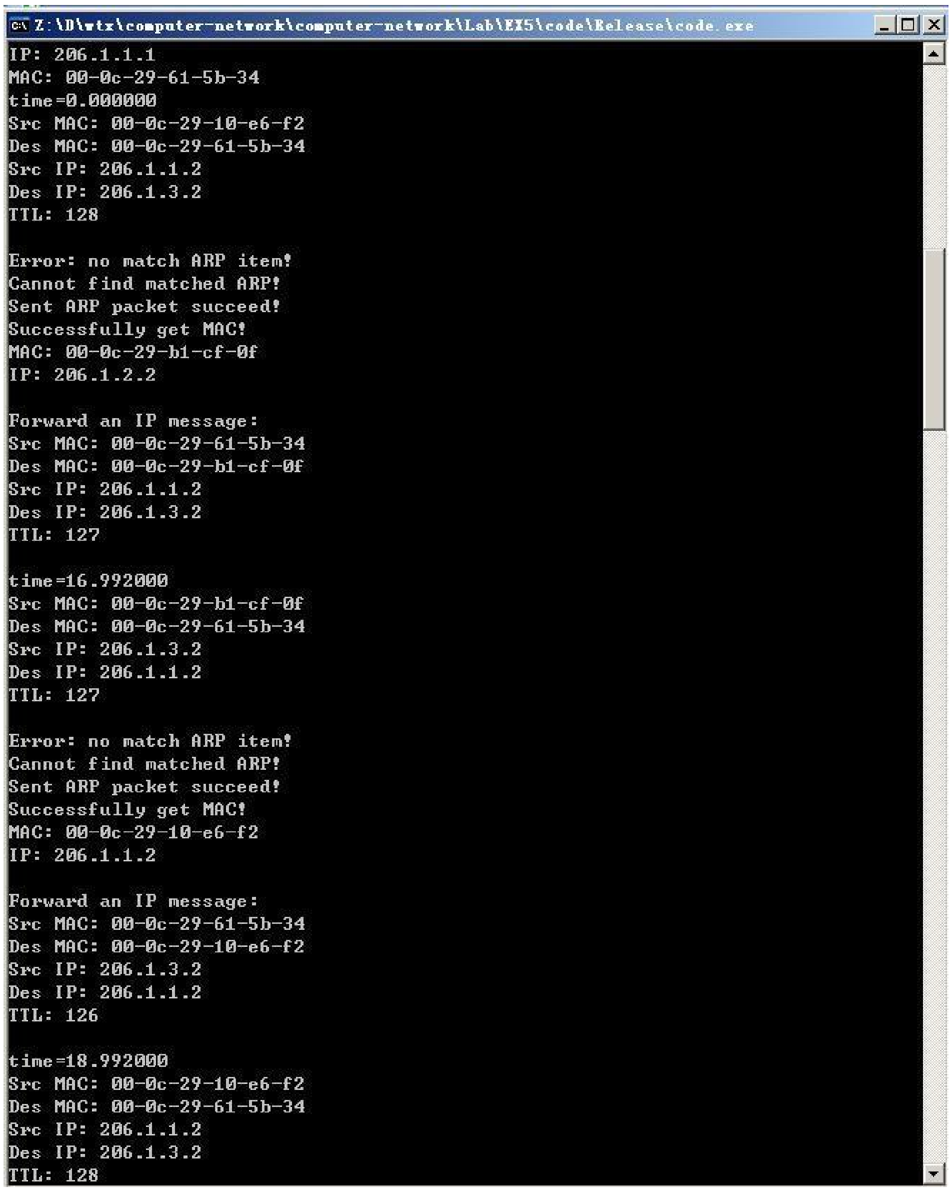


```
Z:\D\wtx\computer-network\computer-network\Lab\EX5\code\Release\code.exe

time=30.993000
Src MAC: 00-0c-29-b1-cf-0f
Des MAC: 00-0c-29-61-5b-34
Src IP: 206.1.3.2
Des IP: 206.1.1.2
TTL: 127

Error: no match ARP item!
Cannot find matched ARP!
Sent ARP packet succeed!
Successfully get MAC!
MAC: 00-0c-29-10-e6-f2
IP: 206.1.1.2

Forward an IP message:
Src MAC: 00-0c-29-61-5b-34
Des MAC: 00-0c-29-10-e6-f2
Src IP: 206.1.3.2
Des IP: 206.1.1.2
TTL: 126

time=32.991000
Src MAC: 00-0c-29-10-e6-f2
Des MAC: 00-0c-29-61-5b-34
Src IP: 206.1.1.2
Des IP: 206.1.3.2
TTL: 128

Error: no match ARP item!
Cannot find matched ARP!
Sent ARP packet succeed!
Successfully get MAC!
MAC: 00-0c-29-b1-cf-0f
IP: 206.1.2.2

Forward an IP message:
Src MAC: 00-0c-29-61-5b-34
Des MAC: 00-0c-29-b1-cf-0f
Src IP: 206.1.1.2
Des IP: 206.1.3.2
TTL: 127

time=34.991000
```

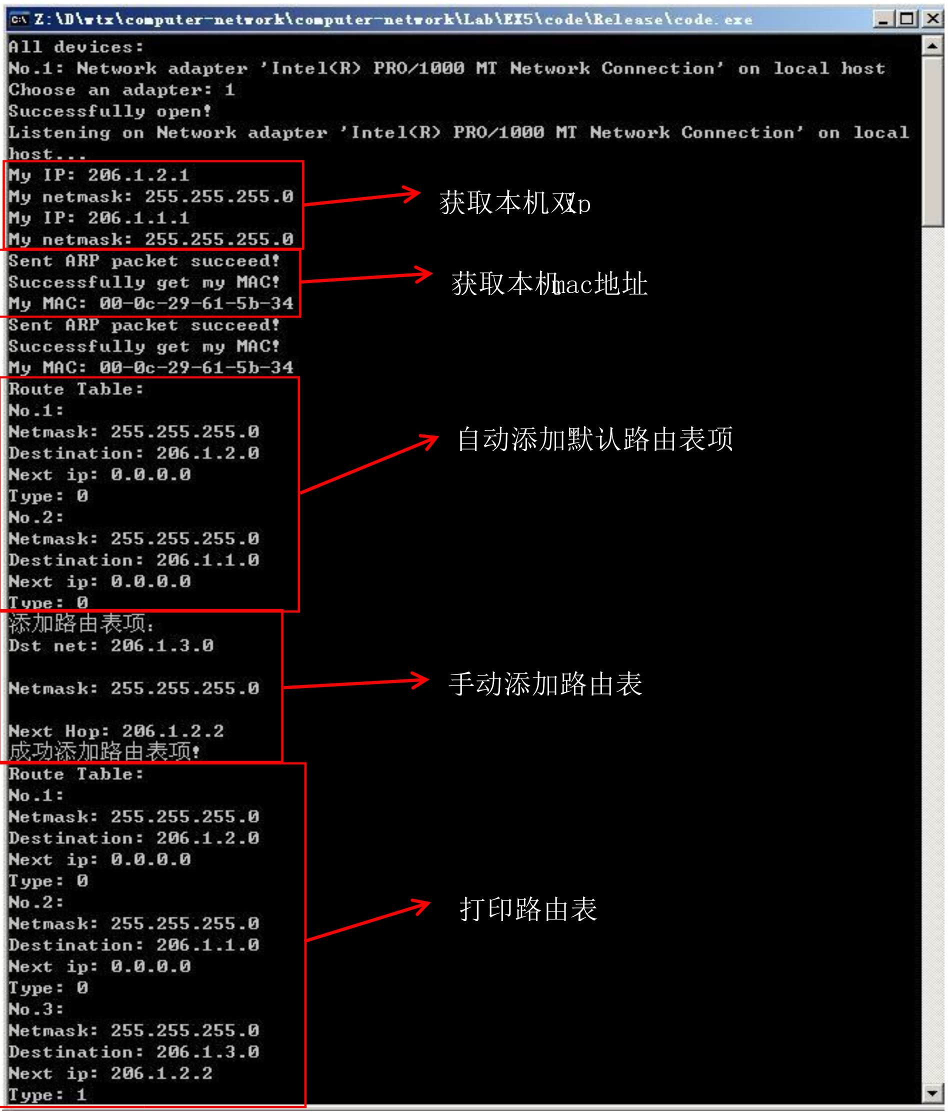
主机 A 可以 ping 通主机 B：


命令提示符
```
Microsoft Windows [版本 5.2.3790]
(C) 版权所有 1985-2003 Microsoft Corp.

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=3603ms TTL=126
Reply from 206.1.3.2: bytes=32 time=3998ms TTL=126
Reply from 206.1.3.2: bytes=32 time=3999ms TTL=126
Reply from 206.1.3.2: bytes=32 time=4000ms 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 = 3603ms, Maximum = 4000ms, Average = 3900ms

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

路由程序日志输出：


Z:\D\wtx\computer-network\computer-network\Lab\EX5\code\Release\code.exe
```
IP: 206.1.1.1
MAC: 00-0c-29-61-5b-34
time=0.000000
Src MAC: 00-0c-29-10-e6-f2
Des MAC: 00-0c-29-61-5b-34
Src IP: 206.1.1.2
Des IP: 206.1.3.2
TTL: 128

Error: no match ARP item!
Cannot find matched ARP!
Sent ARP packet succeed!
Successfully get MAC!
MAC: 00-0c-29-b1-cf-0f
IP: 206.1.2.2

Forward an IP message:
Src MAC: 00-0c-29-61-5b-34
Des MAC: 00-0c-29-b1-cf-0f
Src IP: 206.1.1.2
Des IP: 206.1.3.2
TTL: 127

time=16.992000
Src MAC: 00-0c-29-b1-cf-0f
Des MAC: 00-0c-29-61-5b-34
Src IP: 206.1.3.2
Des IP: 206.1.1.2
TTL: 127

Error: no match ARP item!
Cannot find matched ARP!
Sent ARP packet succeed!
Successfully get MAC!
MAC: 00-0c-29-10-e6-f2
IP: 206.1.1.2

Forward an IP message:
Src MAC: 00-0c-29-61-5b-34
Des MAC: 00-0c-29-10-e6-f2
Src IP: 206.1.3.2
Des IP: 206.1.1.2
TTL: 126

time=18.992000
Src MAC: 00-0c-29-10-e6-f2
Des MAC: 00-0c-29-61-5b-34
Src IP: 206.1.1.2
Des IP: 206.1.3.2
TTL: 128
```

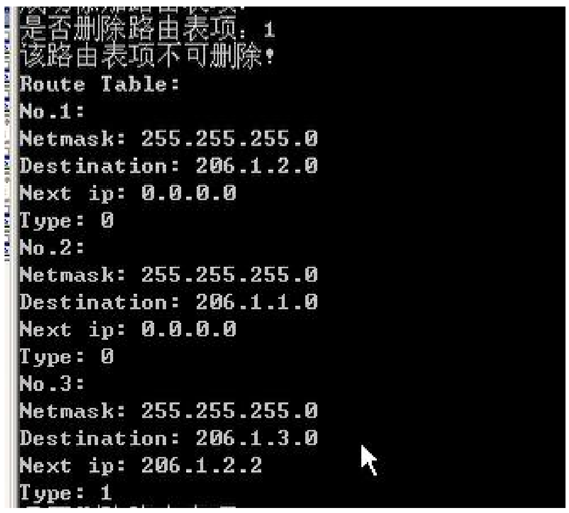
路由表其余信息输出：


Z:\D\wtx\computer-network\computer-network\Lab\EX5\code\Release\code.exe
获取本机双ip
获取本机mac地址
自动添加默认路由表项
手动添加路由表
打印路由表

```
All devices:
No.1: Network adapter 'Intel(R) PRO/1000 MT Network Connection' on local host
Choose an adapter: 1
Successfully open!
Listening on Network adapter 'Intel(R) PRO/1000 MT Network Connection' on local
host...
My IP: 206.1.2.1
My netmask: 255.255.255.0
My IP: 206.1.1.1
My netmask: 255.255.255.0
Sent ARP packet succeed!
Successfully get my MAC!
My MAC: 00-0c-29-61-5b-34
Sent ARP packet succeed!
Successfully get my MAC!
My MAC: 00-0c-29-61-5b-34
Route Table:
No.1:
Netmask: 255.255.255.0
Destination: 206.1.2.0
Next ip: 0.0.0.0
Type: 0
No.2:
Netmask: 255.255.255.0
Destination: 206.1.1.0
Next ip: 0.0.0.0
Type: 0
添加路由表项：
Dst net: 206.1.3.0

Netmask: 255.255.255.0

Next Hop: 206.1.2.2
成功添加路由表项!
Route Table:
No.1:
Netmask: 255.255.255.0
Destination: 206.1.2.0
Next ip: 0.0.0.0
Type: 0
No.2:
Netmask: 255.255.255.0
Destination: 206.1.1.0
Next ip: 0.0.0.0
Type: 0
No.3:
Netmask: 255.255.255.0
Destination: 206.1.3.0
Next ip: 206.1.2.2
Type: 1
```

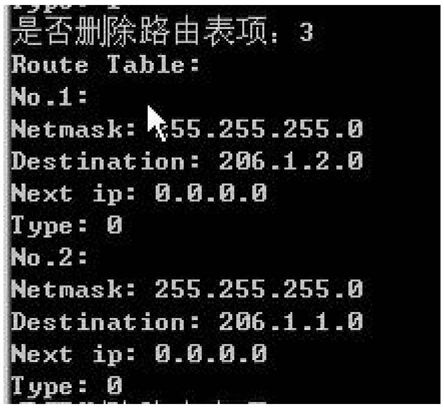
删除路由表项：

默认路由不可删除



```
是否删除路由表项：1
该路由表项不可删除!
Route Table:
No.1:
Netmask: 255.255.255.0
Destination: 206.1.2.0
Next ip: 0.0.0.0
Type: 0
No.2:
Netmask: 255.255.255.0
Destination: 206.1.1.0
Next ip: 0.0.0.0
Type: 0
No.3:
Netmask: 255.255.255.0
Destination: 206.1.3.0
Next ip: 206.1.2.2
Type: 1
```

手动添加的可以删除



```
是否删除路由表项: 3
Route Table:
No.1:
Netmask: 255.255.255.0
Destination: 206.1.2.0
Next ip: 0.0.0.0
Type: 0
No.2:
Netmask: 255.255.255.0
Destination: 206.1.1.0
Next ip: 0.0.0.0
Type: 0
```