实验3：通过编程获取IP地址与MAC地址的对应关系

实验要求

> 通过编程获取IP地址与MAC地址的对应关系实验，要求如下：
>
> （1）在IP数据报捕获与分析编程实验的基础上，学习NPcap的数据包发送方法。
>
> （2）通过NPcap编程，获取IP地址与MAC地址的映射关系。
>
> （3）程序要具有输入IP地址，显示输入IP地址与获取的MAC地址对应关系界面。界面可以是命令行界面，也可以是图形界面，但应以简单明了的方式在屏幕上显示。
>
> （4）编写的程序应结构清晰，具有较好的可读性。

实验环境

```
操作系统：Windows10
编程语言：C++
idea：visual studio2019
```

实验过程

> 实验的头文件获取与实验环境配置与实验二相同，在此处不再赘述，直接讲述实现思路。

- ARP协议

  地址解析协议，是根据IP地址获取物理地址的一个TCP/IP协议。

  主机发送信息时将包含目标IP地址的ARP请求广播到局域网络上的所有主机，并接收返回消息，以此确定目标的物理地址；收到返回消息后将该IP地址和物理地址存入本机ARP缓存中并保留一定时间，下次请求时直接查询ARP缓存以节约资源。

- ARP数据包结构（不包括以太网首部）

硬件类型		协议类型
硬件长度	协议长度	操作码（请求为1，响应为2）
源硬件地址		
源逻辑地址		
目的硬件地址		
目的逻辑地址		

（图3）ARP报文格式

ARP报文分为请求报文和响应报文两种：

- 请求报文：

  在请求报文中，以太网帧首部的目标MAC地址会被设置为全一，代表广播地址；源MAC地址为本机IP地址。而对应的ARP帧中的源硬件地址与以太网帧首部的源MAC地址相同，目的硬件地址为全零，因此时该地址还未知。

- 响应报文：

  响应报文格式中，以太网帧头部的硬件地址与ARP帧的硬件地址相同。

其余部分解析如下：

硬件类型：硬件地址的类型，硬件地址常见的有 MAC 物理或者以太网地址，对于以太网来说，此值为 1。

协议类型：对于 IPv4 地址，这个值是 0x0800。

硬件大小：硬件地址的字节数。对于以太网中使用 IPv4 的 ARP 请求或应答，该值为 6 。

协议大小：协议地址的字节数。对于以太网中使用 IPv4 的 ARP 请求或应答，该值为 4。

操作类型：值为1，表示进行ARP请求；值为2，表示进行ARP应答；值为3，表示进行RARP请求；值为4，表示进行RARP应答。

- ARP工作原理

  当主机需要找出这个网络中的另一个主机的物理地址时，它就可以发送一个ARP请求报文，这个报文包好了发送方的MAC地址和IP地址以及接收方的IP地址。因为发送方不知道接收方的物理地址，所以这个查询分组会在网络层中进行广播。

  局域网中的每一台主机都会接受并处理这个ARP请求报文，然后进行验证，查看接收方的IP地址是不是自己的地址，只有验证成功的主机才会返回一个ARP响应报文，这个响应报文包含接收方的IP地址和物理地址。这个报文利用收到的ARP请求报文中的请求方物理地址以单播的方式直接发送给ARP请求报文的请求方。

- 实验思路

  首先通过findalldevc()获取本机设备的全部网卡，通过opennetwork()函数选择需要捕获的网卡。接下来伪造ARP请求报文向该网卡发送ARP请求，其中伪造报文中的源MAC地址字段和源IP地址字段需要使用虚假的MAC地址和虚假的IP地址：如66-66-66-66-66-66作为源MAC地址，112.112.112.112作为源IP地址。在getip_mac()函数中，网卡一旦获取该ARP请求，就会做出响应，那么本机就进行对响应的ARP数据包进行捕获，从而获得对应网卡的MAC地址。

并且由于广播发送只能在相同网段的网卡内进行发送，而主机上各个网卡的IP可能并不在同一网段，所以本次实验必须使用相同的网卡发出和响应ARP报文。

- 代码解析

全局变量区：

```
pcap_if_t* alldevs;//指向设备列表首部的指针
char errbuf[PCAP_ERRBUF_SIZE];//错误信息缓冲区
pcap_addr_t* a;//用于遍历网络接口的地址信息
pcap_if_t* ptr;//用于遍历网络接口列表
pcap_t* pcap_handle;//存储指定网卡信息
struct pcap_pkthdr* pkt_header;//用于存储抓取到的数据包的头部信息
const u_char* pkt_data; //用于存储抓取到的数据包的头部数据
DWORD SendIP;//源IP地址
DWORD RevIP;//接收到的IP地址
ARPFrame_t ARPFrame;//用于存储ARP帧
ARPFrame_t* IPPacket;//用于存储IP数据包
```

数据报结构定义：

```
struct FrameHeader_t //帧首部
{
    BYTE DesMAC[6];  //目的地址
    BYTE SrcMAC[6];  //源地址
    WORD FrameType;  //帧类型
};

struct ARPFrame_t                //ARP帧
{
    FrameHeader_t FrameHeader;
    WORD HardwareType;//16位,运行ARP的网络类型
    WORD ProtocolType;//16位，使用的协议
    BYTE HLen;//8位，MAC地址的长度
    BYTE PLen;//8位，IP地址的长度
    WORD Operation;//16位，报文的类型
    BYTE SendHa[6];//48位，源MAC地址
    DWORD SendIP;//32位，源IP地址
    BYTE RecvHa[6];//48位，目的MAC地址
    DWORD RecvIP;//32位，目的IP地址
};
```

MAC/IP地址输出：

```
void printMAC(BYTE MAC[6])//以两位十六进制数的格式打印
{
    for (int i = 0; i < 6; i++)
    {
        if (i < 5)
            printf("%02x:", MAC[i]);
        else
            printf("%02x", MAC[i]);
    }

};
```

```
void printIP(DWORD IP)
{
    BYTE* p = (BYTE*)&IP;//将IP的地址强制转换为BYTE类型指针
    //将32位的IPv4地址拆分成4个8位的字节，以便逐个打印
    for (int i = 0; i < 3; i++)
    {
        cout << dec << (int)*p << ".";//dec用于指定十进制格式
        p++;
    }
    cout << dec << (int)*p;
};
```

findalldevc():

```
void findalldevc() {
    int index = 0;//用于跟踪网络接口的索引
    //获得本机的设备列表
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf) ==
-1)
    {
        cout << "获取网络接口时发生错误:" << errbuf << endl;
        return;
    }
    //显示接口列表
    for (ptr = alldevs; ptr != NULL; ptr = ptr->next)
    {
        cout << "编号" << index + 1 << "\t" << ptr->description << endl;
        for (a = ptr->addresses; a != NULL; a = a->next)
        {
            //筛选出IPv4地址
            if (a->addr->sa_family == AF_INET)
            {
                //inet_ntoa用于将IPv4地址（32位）转换为点分十进制字符串表示法
                cout << "IP地址: " << inet_ntoa(((struct sockaddr_in*)(a-
>addr))->sin_addr) << endl;
            }
        }
        index++;
    }
}
```

该函数通过pcap_findalldevs_ex函数，获取本机设备列表并存储到alldevs中，该部分与实验二相同，本实验需要打印对应网卡的IP地址，因此增加循环，通过遍历对应网卡的地址信息，筛选出IPv4地址后使用inet_ntoa函数打印对应的IP地址。

opennetwork():

```
void opennetwork() {
    int num;
    cout << "请选要打开的网卡号: ";
    cin >> num;
    ptr = alldevs;
    for (int i = 1; i < num; i++)
    {
        ptr = ptr->next;
```

```
    }
    pcap_handle = pcap_open(ptr->name, 1024, PCAP_OPENFLAG_PROMISCUOUS, 1000,
NULL, errbuf);//打开网卡
    if (pcap_handle == NULL)
    {
        cout << "打开网卡时发生错误: " << errbuf << endl;
        return ;
    }
    else
    {
        cout << "成功打开该网卡" << endl;
    }
}
```

在该函数中，通过循环遍历到指定的网卡，并使用pcap_open函数将对应的网卡信息传递给pcap_handle。

filter():

```
void filter() {
    //编译过滤器，只捕获ARP包
    u_int netmask;//用于存储子网掩码
    netmask = ((sockaddr_in*)(ptr->addresses->netmask))-
>sin_addr.S_un.S_addr;
    bpf_program fcode;//用于存储编译后的过滤器规则
    char packet_filter[] = "ether proto \\arp";//捕获以太网数据链路层上的ARP协议数
据包
    //pcap_compile 函数将规则字符串 packet_filter 编译成一个 bpf_program 结构,用于
编译过滤器规则
    if (pcap_compile(pcap_handle, &fcode, packet_filter, 1, netmask) < 0)
    {
        cout << "无法编译数据包过滤器。检查语法";
        pcap_freealldevs(alldevs);
        return ;
    }
    //设置过滤器
    if (pcap_setfilter(pcap_handle, &fcode) < 0)
    {
        cout << "过滤器设置错误";
        pcap_freealldevs(alldevs);
        return ;
    }
    cout << "ARP过滤器设置成功" << endl;
}
```

由于本实验我们只需要捕获ARP数据包，因此将过滤器条件设置为"ether proto \arp"，再使用pcap_compile 函数将规则字符串 packet_filter 编译成一个 bpf_program 结构，用于编译过滤器规则，通过设置过滤器，过滤出ARP数据包。

ARPconsist():

```
void ARPconsist() {
    //组装报文
    for (int i = 0; i < 6; i++)
    {
```

```
        ARPFrame.FrameHeader.DesMAC[i] = 0xFF;//设置为本机广播地址
255.255.255.255.255.255
        ARPFrame.FrameHeader.SrcMAC[i] = 0x66;//设置为虚拟的MAC地址66-66-66-66-
66-66-66
        ARPFrame.RecvHa[i] = 0;//设置为0
        ARPFrame.SendHa[i] = 0x66;
    }
    ARPFrame.FrameHeader.FrameType = htons(0x0806);//帧类型为ARP
    ARPFrame.HardwareType = htons(0x0001);//硬件类型为以太网
    ARPFrame.ProtocolType = htons(0x0800);//协议类型为IP
    ARPFrame.HLen = 6;//硬件地址长度为6
    ARPFrame.PLen = 4; // 协议地址长为4
    ARPFrame.Operation = htons(0x0001);//操作为ARP请求
    SendIP = ARPFrame.SendIP = htonl(0x70707070);//源IP地址设置为虚拟的IP地址
112.112.112.112
    cout << "报文组装成功" << endl;
}
```

getip_mac():

```
void getip_mac(DWORD SendIP, DWORD RevIP) {
    while (true)
    {
        //使用 pcap_next_ex 函数来尝试捕获一个数据包
        int rtn = pcap_next_ex(pcap_handle, &pkt_header, &pkt_data);
        if (rtn == -1)
        {
            cout << "捕获数据包时发生错误：" << errbuf << endl;
            return ;
        }
        else
        {
            if (rtn == 0)
            {
                cout << "没有捕获到数据报" << endl;
            }
            else
            {
                IPPacket = (ARPFrame_t*)pkt_data;//将捕获到的数据包内容转换为
ARPFrame_t 类型的数据
                //假定捕获到的数据包是 ARP 请求或响应数据包
                if (IPPacket->RecvIP == SendIP && IPPacket->SendIP ==
RevIP)//判断是不是一开始发的包
                //用来验证是否捕获到了与之前发送的ARP请求相匹配的ARP响应
                {
                    cout << "捕获到回复的数据报,请求IP与其MAC地址对应关系如下：" <<
endl;
                    printIP(IPPacket->SendIP);
                    cout << "   -----   ";
                    printMAC(IPPacket->SendHa);
                    cout << endl;
                    break;
                }
            }
        }
```

```
    }
    return;
}
```

该函数使用pcap_next_ex捕获pcap_handle网卡的数据包，并将捕获到的数据头部保存在pkt_header中，数据包保存在pkt_data中。通过判断rtn的值判断是否捕捉成功。

若捕捉成功则将数据包部分转化为ARPFrame_t 类型，并判断发送端IP与接收端IP是否能够对应上，若能够对应上，则输出对应的IP地址与MAC地址。

senddata():

```
void senddata() {
    //向网络发送数据包
    cout << "\n请输入请求的IP地址:";
    char str[15];
    cin >> str;
    RevIP = ARPFrame.RecvIP = inet_addr(str);
    SendIP = ARPFrame.SendIP = IPPacket->SendIP;//将本机IP赋值给数据报的源IP
    for (int i = 0; i < 6; i++)
    {
        ARPFrame.SendHa[i] = ARPFrame.FrameHeader.SrcMAC[i] = IPPacket-
>SendHa[i];
    }

    if (pcap_sendpacket(pcap_handle, (u_char*)&ARPFrame, sizeof(ARPFrame_t))
!= 0)
    {
        cout << "ARP请求发送失败" << endl;
    }
    else
    {
        cout << "ARP请求发送成功" << endl;
        getip_mac(SendIP, RevIP);
    }
}
```

通过控制台输入字符串，并将字符串通过函数inet_addr转化为IP地址格式，通过将对应IP和MAC填入报文，重新发送ARP请求，使用pcap_sendpacket函数判断请求报文是否发送成功，最后再通过getip_mac函数获取响应报文中IP地址与mac地址的对应关系。

main:

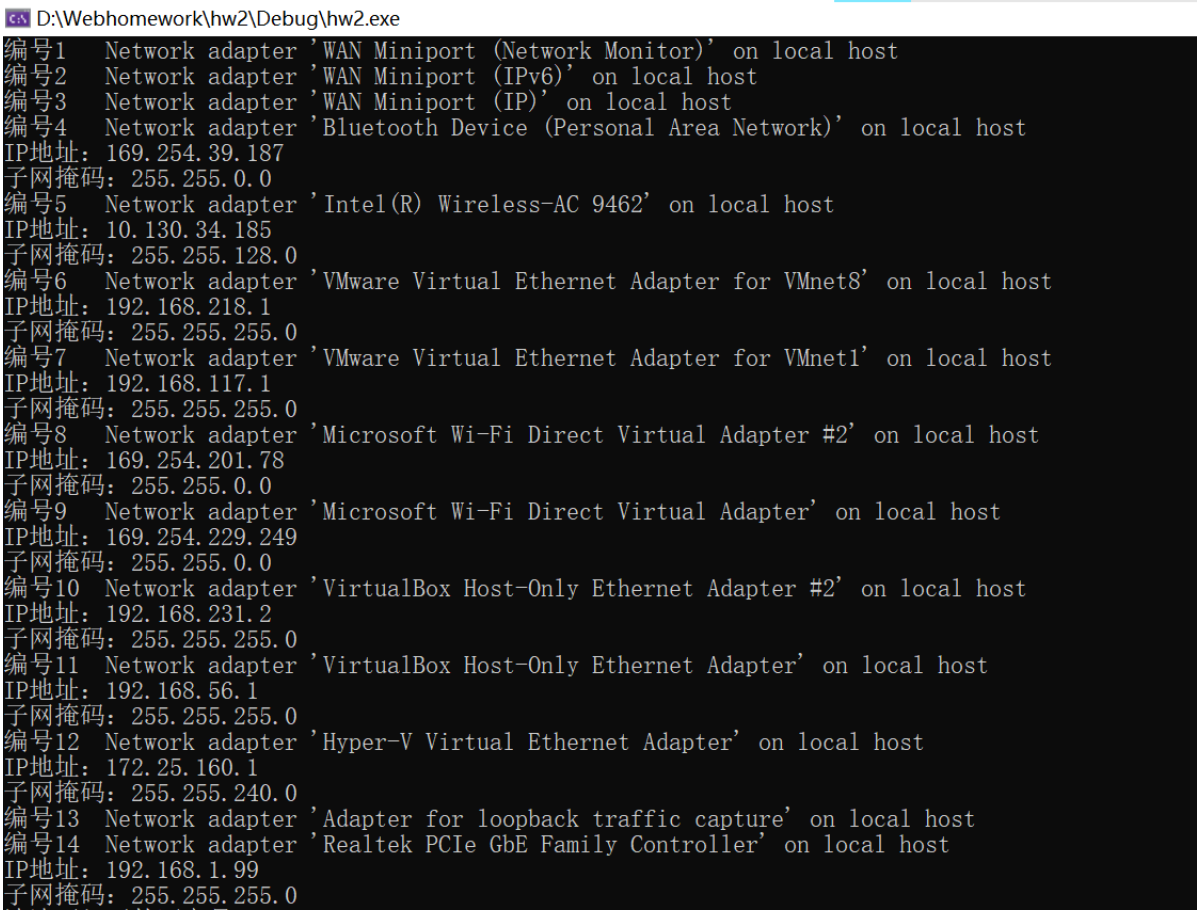
```
int main()
{
    findalldevc();
    opennetwork();
    filter();
    ARPconsist();
    //将所选择的网卡的IP设置为请求的IP地址
    for (a = ptr->addresses; a != NULL; a = a->next)
    {
        if (a->addr->sa_family == AF_INET)
        {
            RevIP = ARPFrame.RecvIP = inet_addr(inet_ntoa(((struct
sockaddr_in*)(a->addr))->sin_addr));
```

```
        }
    }
    //使用 pcap_sendpacket 函数发送一个构建好的ARP请求报文
    pcap_sendpacket(pcap_handle, (u_char*)&ARPFrame, sizeof(ARPFrame_t));
    cout << "ARP请求发送成功" << endl;
    getip_mac(SendIP, RevIP);
    senddata();
}
```

在主函数中，依次调用上述函数，首先获取本机全部网卡，在选定网卡选择一局域网，选定好后设置过滤器、组装ARP报文，使用pcap_sendpacket函数发送一个构建好的ARP请求报文，随后使用getip_mac函数输出对应关系。在senddata函数中实现输入IP地址，显示输入IP地址与获取的MAC地址对应关系界面的功能。

实验结果

显示网卡：



显示的网卡中，编号1、2、3、13没有对应的IP地址，原因如下：

- 网卡1、2、3中，网卡类型为WAN Miniport，该网卡不是一个特定的物理网卡，而是一种虚拟网络适配器或网络接口，用于在Windows操作系统中处理各种广域网（WAN）连接。这些虚拟适配器用于实现不同类型的远程网络连接，例如虚拟专用网络（VPN）连接、拨号连接、宽带连接等。WAN Miniport 本身通常不具有 IP 地址。它是一个虚拟网络适配器，主要用于创建和管理各种广域网（WAN）连接，例如 VPN 连接、拨号连接等。这些虚拟适配器不直接分配或持有 IP 地址，而是在创建特定的网络连接时，它们会与网络协议栈一起工作，以获取和分配 IP 地址。
- 网卡13为Adapter for loopback traffic capture，该网卡为回环网卡，该接口不与任何物理网络适配器或网络电缆连接。它主要用于发送和接收本地的网络流量，通常用于测试和排错。

我们选择捕获网卡5：

```
子网掩码: 255.255.255.0
请选要打开的网卡号: 5
成功打开该网卡
ARP过滤器设置成功
报文组装成功
ARP请求发送成功
捕获到回复的数据报,请求IP与其MAC地址对应关系如下:
10.130.34.185    -----    f0:77:c3:eb:88:bd
```

可以看到我们成功获取IP与MAC的对应关系。为验证正确性，在命令行输入ipconfig命令，查看本机网络配置：

```
无线局域网适配器 WLAN:

   连接特定的 DNS 后缀 . . . . . :
   描述. . . . . . . . . . . . . . : Intel(R) Wireless-AC 9462
   物理地址. . . . . . . . . . . . : F0-77-C3-EB-88-BD
   DHCP 已启用 . . . . . . . . . . : 是
   自动配置已启用. . . . . . . . . : 是
   IPv6 地址 . . . . . . . . . . . : 2001:250:401:6559:645:81d3:affb:5cc9(首选)
   临时 IPv6 地址. . . . . . . . . : 2001:250:401:6559:dd36:9d80:a63e:c060(首选)
   本地链接 IPv6 地址. . . . . . . : fe80::250e:8458:e86a:f0bc%3(首选)
   IPv4 地址 . . . . . . . . . . . : 10.130.34.185(首选)
   子网掩码 . . . . . . . . . . . . : 255.255.128.0
   获得租约的时间  . . . . . . . . : 2023年11月8日 21:32:18
   租约过期的时间  . . . . . . . . : 2023年11月9日 3:32:18
   默认网关. . . . . . . . . . . . : fe80::865b:12ff:fe5e:360f%3
                                     10.130.0.1
   DHCP 服务器 . . . . . . . . . . : 10.130.0.1
   DHCPv6 IAID . . . . . . . . . . : 66090947
   DHCPv6 客户端 DUID  . . . . . . : 00-01-00-01-28-A8-22-CA-60-18-95-3F-05-0C
   DNS 服务器  . . . . . . . . . . : 222.30.45.41
                                     202.113.16.41
   TCPIP 上的 NetBIOS  . . . . . . : 已启用
```

结果相同，获取成功。

我们得知，网卡5的子网掩码为255.255.128.0，可知IP地址的后15位为主机号，因此，对于该IP地址（10.130.34.185）在同一局域网的IP地址为10.130.0.1至10.130.127.254（10.130.0.0为网络号，10.130.127.254为广播号，因此不能使用）

我们任意选择该范围内的一IP地址进行数据包发送：

```
请输入请求的IP地址:10.130.0.1
ARP请求发送成功
捕获到回复的数据报,请求IP与其MAC地址对应关系如下:
10.130.0.1       -----    00:00:5e:00:01:0d
```

arp -a命令验证结果如下：

```
C:\Users\花花>arp -a 10.130.0.1

接口: 10.130.34.185 --- 0x3
  Internet 地址         物理地址              类型
  10.130.0.1            00-00-5e-00-01-0d     动态
```

验证结果正确。

实验心得

通过本实验了解IP地址和MAC地址的基本概念和作用，熟悉ARP协议与报文的内容，加深对网络通信的理解，数据传输机制。