数据包捕获与分析编程实验

姓名：杨惠媛 学号：2010575

要求如下：（1）了解NPcap的架构。（2）学习NPcap的设备列表获取方法、网卡设备打开方法，以及数据包捕获方法。（3）通过NPcap编程，实现本机的数据包捕获，显示捕获数据帧的源MAC地址和目的MAC地址，以及类型/长度字段的值。（4）捕获的数据报不要求硬盘存储，但应以简单明了的方式在屏幕上显示。必显字段包括源MAC地址、目的MAC地址和类型/长度字段的值。（5）编写的程序应结构清晰，具有较好的可读性

一、环境配置

首先进行环境配置，配置过程参考了Win10安装Npcap、npcap-sdk，并在Visual Studio 2017配置使用最新npcap-CSDN博客

本实验环境为**win32**+VS2022

二、代码过程

1.包含头文件和宏定义

```
#include <winsock2.h>
#include "pcap.h"
#include <iostream>
#include <iomanip>
#include <cstdio>

//将 Windows 套接字库（ws2_32.lib）链接到生成的可执行文件中。
#pragma comment(lib,"ws2_32.lib")

//禁止特定的编译器警告
#pragma warning(disable:4996)
#pragma warning(disable:6011)
using namespace std;
```

2.结构体设计

mac结构体的设计

```
//存储MAC地址的结构体
struct MAC
{
    uint8_t source[6];//源MAC地址
    uint8_t destination[6];//目的MAC地址
    //表示以太网帧的类型。这个字段通常标识了以太网帧中携带的协议类型，如IPv4、IPv6、ARP等。
```

```
    uint16_t ether_type;
};
```

IP信息结构体的设计

```
//存储IP地址和校验和的结构体
struct IPinfo
{
    uint8_t header_length : 4;// IP头部长度
    uint8_t version : 4;
    uint8_t tos;// 服务类型
    uint16_t totallength;
    uint16_t checksum;//校验和字段
    struct in_addr  source_ip;//源地址
    struct in_addr  destination_ip;//目的地址
};
```

对于结构体强制转换类型的探讨：

在后面的packet_handle函数中，获取到的数据包是放在pkt_data中的，这是原始的二进制数据，而我们使用了强制转换类型`mac_save = (struct MAC*)pkt_data;`将pkt_data转换为我们设计的MAC结构体，由此数据包被存在了MAC结构体中

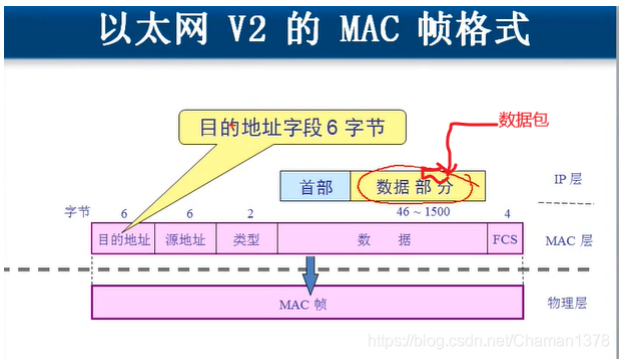
```
void packet_handler(u_char* param, const struct pcap_pkt hdr* header, const u_char*
pkt_data)
struct MAC* mac_save; /*以太网协议变量*/
struct IPinfo* ip_save;/*ip协议变量*/
mac_save = (struct MAC*)pkt_data;
ip_save = (struct IPinfo*)(pkt_data + 14);
```

强制类型转换`(struct MAC*)pkt_data`告诉编译器将`pkt_data`视为一个`struct MAC*`类型的指针，这使得程序员可以访问`pkt_data`中的数据，并将其解释为`struct MAC`结构体中的字段。

这就涉及到如何将`pkt_data`中的字节映射到`struct MAC`结构体的字段：

`pkt_data`是一个`u_char*`类型的指针，指向原始捕获的网络数据包的内容。数据是按照以太网帧的格式存储的。因为以太网帧是一个二进制协议，所以每个字段都是按照字节的形式存储的。

以太网帧的结构如下：



存储MAC地址信息的结构体设计如下：

```
//存储MAC地址的结构体
struct MAC
{
    uint8_t source[6];//源MAC地址
    uint8_t destination[6];//目的MAC地址
    //表示以太网帧的类型。这个字段通常标识了以太网帧中携带的协议类型，如IPv4、IPv6、ARP等。
    uint16_t ether_type;
};
```

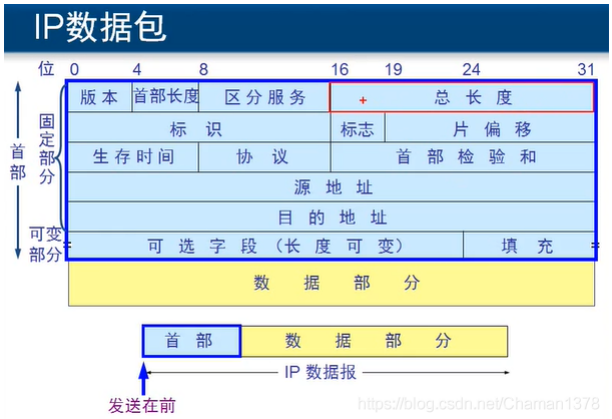
所以映射关系如下：

1. 前6个字节（0到5）对应于 `uint8_t source[6]`，即源MAC地址。
2. 接下来的6个字节（6到11）对应于 `uint8_t destination[6]`，即目的MAC地址。
3. 最后的2个字节（12到13）对应于 `uint16_t ether_type`，即以太网帧类型字段。

换句话说就是，设计结构体的时候，**结构体的成员顺序决定了内存中存储数据的布局**，所以结构体的成员顺序设计是很重要的设计考虑因素，这对于处理二进制数据、解析协议或进行低级数据操作非常重要。

其中IP数据包结构体的设计也是同理，此处不再赘述。

其中IP数据包如下：



IP数据包结构体设计如下：

```
//存储IP地址和校验和的结构体
struct IPinfo
{
    uint8_t header_length : 4;// IP头部长度
    uint8_t version : 4;
    uint8_t tos;// 服务类型
    uint16_t totallength;
    uint16_t checksum;//校验和字段
    struct in_addr  source_ip;//源地址
    struct in_addr  destination_ip;//目的地址
};
```

其中对于`uint8_t header_length : 4;`和`uint8_t version : 4;`和IP数据包的4位版本和4位首部长度看似不匹配

实际上是因为，在C/C++中，没有直接支持4位整数类型的内置数据类型。因此，通常使用8位的`uint8_t`或`unsigned char`来表示4位字段，然后在程序中使用位运算或其他方法来提取或设置4位字段的值。

```
uint8_t header_length : 4;// IP头部长度
uint8_t version : 4;
```

这两个字段，`header_length`和`version`，**共同占用了8位**（1字节）的空间。`: 4`的语法用于指定每个字段在字节内占用的位数。`: 4`表示`header_length`使用这个字节的低4位，而`version`使用这个字节的高4位。

所以和IP数据的映射关系是正确的

3.main函数的设计

首先对端口列表进行扫描，并将结果存储在Header中；如果出错则将错误信息存储在errinfo当中，并且函数值会返回-1

```
pcap_if_t* Header;
char errinfo[PCAP_ERRBUF_SIZE];
if(pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &Header, errinfo) == -1)
    {
        cout << "Error: 获取端口失败! ";
        return -1;
    }
```

当扫描成功时，对header链表进行扫描，直到链尾，把链表中每个节点的信息，也即每个端口的信息输出出来

```
int i = 0;
    pcap_if_t* tool = Header;
    for (tool; tool != NULL; tool = tool->next)
    {
        cout << ++i << "端口名称: " << tool->name << endl;
        cout << tool->description << endl;
        cout<<endl;

    }
```

其中再设置一个错误情况的处理

```
if (i == 0)
    {
        cout << "Error:没有找到端口,请检查。" << endl;
        return -1;
    }
```

当错误情况没有发生时，进行下一步操作

即数据包的获取

首先获取客户想要进入的端口号

```
//开始获取数据包
    int num;
    cout << "请输入想要进入的端口号:" << endl;
    cin >> num;
    if (num<1 || num>i)
    {
        cout << "端口号越界" << endl;
        pcap_freealldevs(Header);
        return -1;
    }
```

进入该端口号

```
/* 跳转到选中的适配器 */
    tool = Header;
    for (i = 0; i < num; i++) {
        tool = tool->next;
    }
```

将该端口传给myport，并检查该端口是否成功打开

```
if ((myport = pcap_open(tool->name, 65535, PCAP_OPENFLAG_PROMISCUOUS, 1000, NULL,
errinfo)) == NULL)
    {
        cout << "Eerror:打开端口失败!"<<endl;
        pcap_freealldevs(Header);
        return -1;
    }
    //打开成功
    cout << "开始监听: " << tool->description << endl;
    pcap_freealldevs(Header);
```

客户输入想要捕获的数据包数量,pcap_loop()函数由此会回调packet_handler函数

```
int numofdata = 0;
    cout << " 请输入你想要捕获的数据包数量:" << endl;
    cin >> numofdata;

    //用于捕获网络数据包并将它们传递给一个回调函数进行处理。
    pcap_loop(myport, numofdata, packet_handler, NULL);

    cout << " 数据报捕获结束！退出程序！" << endl;
```

4.packet_handler()函数的设计

```
void packet_handler(u_char* param, const struct pcap_pkt hdr* header, const u_char*
pkt_data)
{
     // header包含有关捕获数据包的元数据，如时间戳和数据包长度。
    /* 将时间戳转换成可识别的格式 */
    cout << "数据包长度:  " << header->len << "字节" << endl;

    struct MAC* mac_save; /*以太网协议变量*/
    struct IPinfo* ip_save;/*ip协议变量*/

    //指向捕获到的数据包内容的指针
    //以太网帧
    mac_save = (struct MAC*)pkt_data;
    /*源MAC地址*/
    mac_tool = mac_save->source;
    cout << "源MAC地址:   ";
    //mac_tool有16字节
    printf("%02x:%02x:%02x:%02x:%02x:%02x", *mac_tool, *(mac_tool + 1), *(mac_tool +
2), *(mac_tool + 3), *(mac_tool + 4), *(mac_tool + 5));
    cout << endl;

    /*目的MAC地址*/
    mac_tool = mac_save->destination;
    cout << "目的地MAC地址: ";
    printf("%02x:%02x:%02x:%02x:%02x:%02x", *mac_tool, *(mac_tool + 1), *(mac_tool +
2), *(mac_tool + 3), *(mac_tool + 4), *(mac_tool + 5));
    cout << endl;

    /*源ip地址*/
    cout << "源IP地址:" << inet_ntoa(ip_save->source_ip) << endl;

    ip_save = (struct IPinfo*)(pkt_data + 14);
    /*源ip地址*/
    cout << "源IP地址:" << inet_ntoa(ip_save->source_ip) << endl;

    /*目的ip地址*/
    cout << "目的地IP地址:" << inet_ntoa(ip_save->destination_ip) << endl;

    /*校验和字段*/
    cout << "校验和字段:  " << ip_save->checksum << endl;
}
```

三、代码运行结果

```
Network adapter 'VMware Virtual Ethernet Adapter for VMnet8' on local host

7端口名称: rpcap://\Device\NPF_{41CD2907-A1C4-48E7-B0D4-FC71A4BD67DD}
Network adapter 'VMware Virtual Ethernet Adapter for VMnet1' on local host

8端口名称: rpcap://\Device\NPF_{56E7872B-0BDA-4E62-898E-6E46B745D8D5}
Network adapter 'Microsoft Wi-Fi Direct Virtual Adapter #2' on local host

9端口名称: rpcap://\Device\NPF_{2B4A8A8E-BC5E-4DAF-B06E-31EDA16274A5}
Network adapter 'Microsoft Wi-Fi Direct Virtual Adapter' on local host

10端口名称: rpcap://\Device\NPF_Loopback
Network adapter 'Adapter for loopback traffic capture' on local host

11端口名称: rpcap://\Device\NPF_{3BD79D60-191E-4D68-AE37-88AEF6D33DB8}
Network adapter 'Sangfor SSL VPN CS Support System VNIC' on local host

12端口名称: rpcap://\Device\NPF_{45D84681-311B-4EE1-8C2E-998293902EE5}
Network adapter 'Kaspersky VPN' on local host

13端口名称: rpcap://\Device\NPF_{62A9E5FF-381F-439C-87B7-08673490F8A0}
Network adapter 'Intel(R) Ethernet Connection (6) I219-V' on local host

请输入想要进入的端口号:
4
开始监听: Network adapter 'Intel(R) Wi-Fi 6 AX201 160MHz' on local host
 请输入你想要捕获的数据包数量:
1
捕获到数据包!
数据包长度:  217字节
源MAC地址:   01:00:5e:7f:ff:fa
目的地MAC地址: cc:f9:e4:f2:f2:74
以太网帧类型/长度字段: 0x800 (2048)
源IP地址:1.17.0.0
目的地IP地址:10.130.108.186
校验和字段:  31338


 数据报捕获结束! 退出程序!
```