数据通信作业-5

姓名：<u>费扬</u> 学号：<u>519021910917</u> 日期：<u>2022.04.27</u>

一、 实验名称及内容

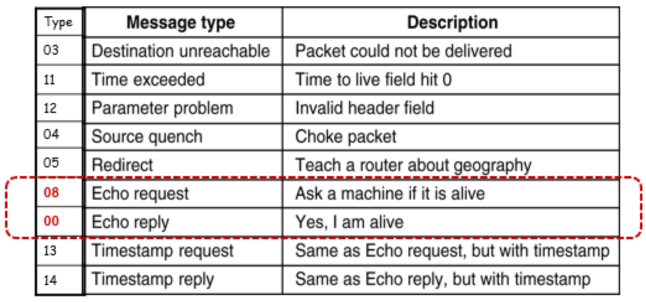
Assignment4：*<u>Ping: icmp</u>*

使用 winsock 编程，主要目标任务为：

Task: Write a program to test the reachability of an Internet interface identified by an IP address or name. (The basic function of "ping" command)
Hints: Send an ICMP "echo request" to the destination, an ICMP "echo reply" will be sent back if the destination is reachable.

即编写一个程序来测试由IP地址或名称标识的Internet接口的可访问性（ping命令的基本功能）。提示：向目的地发送ICMP"echo request"，如果可以到达目的地，则返回ICMP"echo reply"。

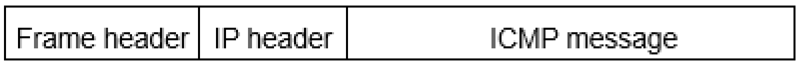
二、 实验过程和结果



Type	Message type	Description
03	Destination unreachable	Packet could not be delivered
11	Time exceeded	Time to live field hit 0
12	Parameter problem	Invalid header field
04	Source quench	Choke packet
05	Redirect	Teach a router about geography
08	Echo request	Ask a machine if it is alive
00	Echo reply	Yes, I am alive
13	Timestamp request	Same as Echo request, but with timestamp
14	Timestamp reply	Same as Echo reply, but with timestamp

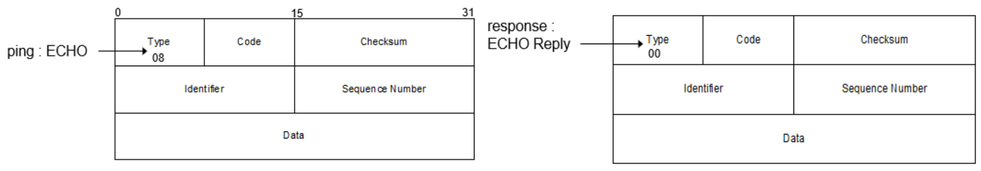
ICMP 报文的类型

这里使用了 08 和 00 两种，即 Echo request 和 Echo reply，确认连接的机器是处于活跃状态。



ICMP 被认为是 IP 的一部分，但也是 IP 用户，ICMP 的报文被封装在 IP 报文内部。

因此，本次通信任务的报文处理如下：



基本操作步骤：

Myping.cpp：

1. Create a raw socket：socktype=SOCK_RAW, protocol=IPPROTO_ICMP;
2. Construct an ICMP message;
3. Use “sendto” to send the ICMP message to the remote machine;
4. Use “recvfrom” to receive any response.
5. Wireshark

代码见打包文件中 myping.cpp，这里只介绍思路：

首先定义 icmp_hdr 的结构体，包括变量如下：

```
typedef struct icmp_hdr {
    unsigned char icmp_type;
    unsigned char icmp_code;
    unsigned short    icmp_checksum;
    unsigned short    icmp_id;
    unsigned short    icmp_sequence;
} ICMP_HDR;
```

然后在 Main 函数中：
WSAStartup 并创建 socket；

```
s = socket(remote->ai_family, SOCK_RAW, IPPROTO_ICMP);
if (s == INVALID_SOCKET) {
        cout << "socket() failed with " << WSAGetLastError() << endl;
        freeaddrinfo(remote);
        freeaddrinfo(local);
        WSACleanup();
        return -1;
    }
```

进行 setsockopt、分配空间的初始化，并对参数如下定义；

```
    icmp_hdr = (ICMP_HDR*)icmpbuf;
    icmp_hdr->icmp_type = 8;
    icmp_hdr->icmp_code = 0;
    icmp_hdr->icmp_id = (unsigned short)GetCurrentProcessId();
    icmp_hdr->icmp_sequence = 0;
    icmp_hdr->icmp_checksum = 0;

    datapart = icmpbuf + sizeof(ICMP_HDR);
    memset(datapart, 'Q', DEFAULT_SIZE);
```

进行 Bind；

```
iResult = bind(s, local->ai_addr, (int)local->ai_addrlen);
    if (iResult == SOCKET_ERROR) {
        cout << "bind failed with " << WSAGetLastError() << endl;
        freeaddrinfo(remote);
        freeaddrinfo(local);
        closesocket(s);
        free(icmpbuf);
        WSACleanup();
        return -1;
    }
```

Receive 并打印输出；

```
      RecvFrom(s, recvbuf, recvbuflen, (SOCKADDR*)&from, &fromlen, &recvol);
      cout << "Pinging: " << destHost;
          PrintAddress(remote->ai_addr, remote->ai_addrlen);
          cout << " with " << DEFAULT_SIZE << " bytes of data." << endl;
```

(☆)循环四次传输，使用 sendto 和 WaitForSingleObject 进行收发；

- **首先调用函数确定ICMP格式，并进行校验和计算：**

```
 SetIcmpSequence(icmpbuf);
ComputeIcmpchecksum(icmpbuf, packetlen);
```

- **Sendto和SingleObject捕获：**

```
 iResult = sendto(s, icmpbuf, packetlen, 0, remote->ai_addr,
(int)remote->ai_addrlen);
if (iResult == SOCKET_ERROR) {
    cout << "sendto failed with %" << WSAGetLastError() << endl;
    freeaddrinfo(remote);
    freeaddrinfo(local);
    closesocket(s);
    free(icmpbuf);
    WSACloseEvent(recvol.hEvent);
    WSACleanup();
    return -1;
}

iResult = WaitForSingleObject((HANDLE)recvol.hEvent, DEFAULT_RECV_TIMEOUT);
if (iResult == WAIT_FAILED) {
    cout << "WaitForSigleObject failed with " << WSAGetLastError() << endl;
    freeaddrinfo(remote);
    freeaddrinfo(local);
    closesocket(s);
    free(icmpbuf);
```

```
            WSACloseEvent(recvol.hEvent);
            WSACleanup();
            return -1;
        }
        else if (iResult == WAIT_TIMEOUT) {
            cout << "Request Time Out." << endl;
        }
        else {
            time = (ULONG)GetTickCount64() - time;
            WSAResetEvent(recvol.hEvent);
            RecvPack += 1;

            cout << "Reply From";
            PrintAddress((SOCKADDR*)&from, fromlen);
            if (time == 0) {
                printf(": bytes = %d time < 1 ms TTL = %d\n", DEFAULT_SIZE, TTL);
            }
            else {
                printf(": bytes = %d time = %d ms TTL = %d\n", DEFAULT_SIZE, time,
TTL);
            }
            if (i < 3) {
                fromlen = sizeof(SOCKADDR_STORAGE);
                RecvFrom(s, recvbuf, recvbuflen, (SOCKADDR*)&from, &fromlen, &recvol);
            }
        }
       Sleep(1000);
```

- **操作输出**
- **Checksum函数定义如下：**

```
 USHORT checksum(USHORT* buffer, int size) {
unsigned long cksum = 0;
 while (size > 1) {
     cksum += *buffer++;
     size -= sizeof(USHORT);
 }
 if (size) {
     cksum += *(UCHAR*)buffer;
 }
 cksum = (cksum >> 16) + (cksum & 0xffff);
 cksum += (cksum >> 16);
 return (USHORT)(~cksum);
}
```

- **ComputeIcmpchecksum函数定义如下：**

```
void ComputeIcmpchecksum(char* buf, int packetlen) {
  ICMP_HDR* icmpv4 = NULL;
  icmpv4 = (ICMP_HDR*)buf;
  icmpv4->icmp_checksum = 0;
 icmpv4->icmp_checksum = checksum((USHORT*)buf, packetlen);
}
```

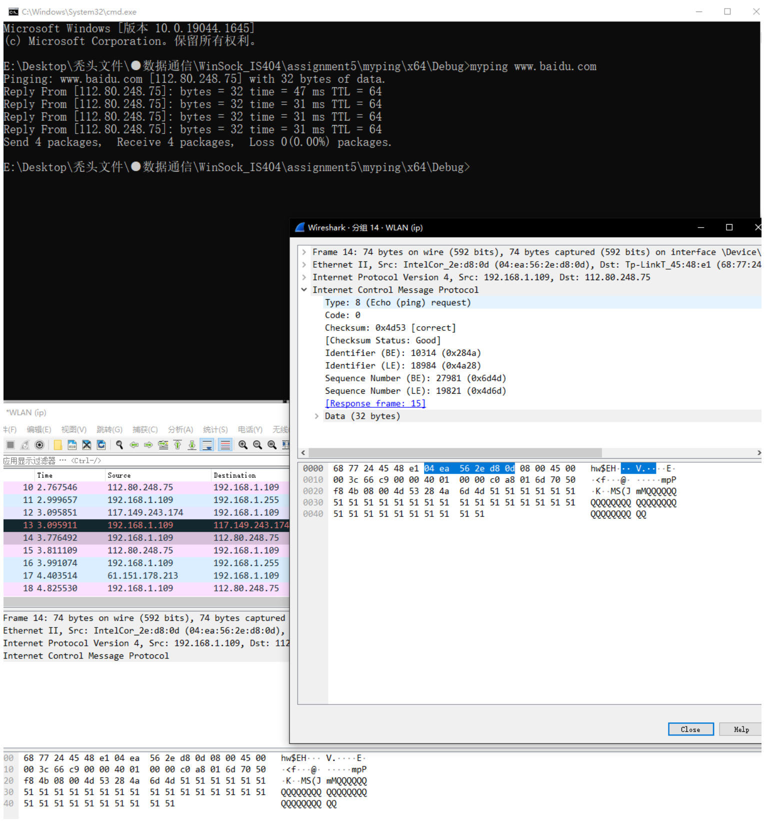
- **SetIcmpSequence函数定义如下：**

```
void SetIcmpSequence(char* buf) {
    ULONG sequence = 0;
    sequence = (ULONG)GetTickCount64();
    ICMP_HDR* icmpv4 = NULL;
    icmpv4 = (ICMP_HDR*)buf;
    icmpv4->icmp_sequence = (USHORT)sequence;
    }
```

- **RecvFrom 函数定义如下：**

```
int RecvFrom(SOCKET s, char* buf, int buflen, SOCKADDR* from, int* fromlen,
WSAOVERLAPPED* ol) {
    WSABUF wbuf;
    DWORD flags, bytes;
    int iResult;
    wbuf.buf = buf;
    wbuf.len = buflen;
    flags = NULL;

    iResult = WSARecvFrom(s, &wbuf, 1, &bytes, &flags, from, fromlen, ol,
NULL);
    if (iResult == SOCKET_ERROR) {
        if (WSAGetLastError() != WSA_IO_PENDING) {
            printf("WSARecvfrom failed: %d\n", WSAGetLastError());
            return SOCKET_ERROR;
        }
    }
    return NO_ERROR;
    }
```

之后在命令行操作输出，同时打开 wireshark 进行监测：

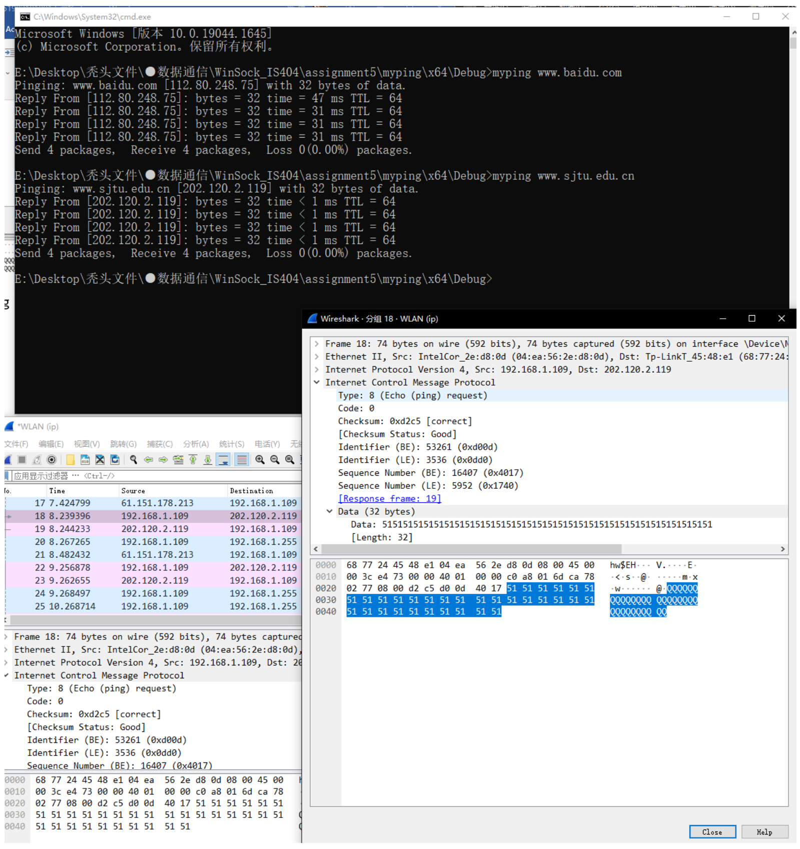


```
Microsoft Windows [版本 10.0.19044.1645]
(c) Microsoft Corporation。保留所有权利。

E:\Desktop\秃头文件\●数据通信\WinSock_IS404\assignment5\myping\x64\Debug>myping www.baidu.com
Pinging: www.baidu.com [112.80.248.75] with 32 bytes of data.
Reply From [112.80.248.75]: bytes = 32 time = 47 ms TTL = 64
Reply From [112.80.248.75]: bytes = 32 time = 31 ms TTL = 64
Reply From [112.80.248.75]: bytes = 32 time = 31 ms TTL = 64
Reply From [112.80.248.75]: bytes = 32 time = 31 ms TTL = 64
Send 4 packages,  Receive 4 packages,  Loss 0(0.00%) packages.

E:\Desktop\秃头文件\●数据通信\WinSock_IS404\assignment5\myping\x64\Debug>
```

```
      Time        Source            Destination
   10 2.767546    112.80.248.75     192.168.1.109
   11 2.999657    192.168.1.109     192.168.1.255
   12 3.095851    117.149.243.174   192.168.1.109
   13 3.095911    192.168.1.109     117.149.243.174
   14 3.776492    192.168.1.109     112.80.248.75
   15 3.811109    112.80.248.75     192.168.1.109
   16 3.991074    192.168.1.109     192.168.1.255
   17 4.403514    61.151.178.213    192.168.1.109
   18 4.825530    192.168.1.109     112.80.248.75
```

```
Wireshark · 分组 14 · WLAN (ip)
> Frame 14: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\
> Ethernet II, Src: IntelCor_2e:d8:0d (04:ea:56:2e:d8:0d), Dst: Tp-LinkT_45:48:e1 (68:77:24
> Internet Protocol Version 4, Src: 192.168.1.109, Dst: 112.80.248.75
v Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x4d53 [correct]
    [Checksum Status: Good]
    Identifier (BE): 10314 (0x284a)
    Identifier (LE): 18984 (0x4a28)
    Sequence Number (BE): 27981 (0x6d4d)
    Sequence Number (LE): 19821 (0x4d6d)
    [Response frame: 15]
  > Data (32 bytes)

0000  68 77 24 45 48 e1 04 ea  56 2e d8 0d 08 00 45 00   hw$EH·····V.···E·
0010  00 3c 66 c9 00 00 40 01  00 00 c0 a8 01 6d 70 50   ·<f···@· ·····mpP
0020  f8 4b 08 00 4d 53 28 4a  6d 4d 51 51 51 51 51 51   ·K··MS(J mMQQQQQQ
0030  51 51 51 51 51 51 51 51  51 51 51 51 51 51 51 51   QQQQQQQQ QQQQQQQQ
0040  51 51 51 51 51 51 51 51  51 51                     QQQQQQQQ QQ
```

在 cmd 中实现 ping 百度，wireshark 抓包 icmp

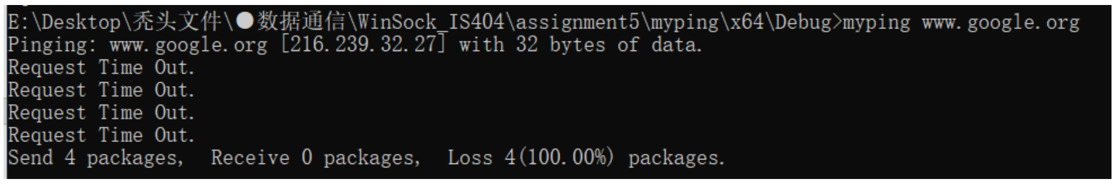


```
Microsoft Windows [版本 10.0.19044.1645]
(c) Microsoft Corporation。保留所有权利。

E:\Desktop\秃头文件\●数据通信\WinSock_IS404\assignment5\myping\x64\Debug>myping www.baidu.com
Pinging: www.baidu.com [112.80.248.75] with 32 bytes of data.
Reply From [112.80.248.75]: bytes = 32 time = 47 ms TTL = 64
Reply From [112.80.248.75]: bytes = 32 time = 31 ms TTL = 64
Reply From [112.80.248.75]: bytes = 32 time = 31 ms TTL = 64
Reply From [112.80.248.75]: bytes = 32 time = 31 ms TTL = 64
Send 4 packages,  Receive 4 packages,  Loss 0(0.00%) packages.

E:\Desktop\秃头文件\●数据通信\WinSock_IS404\assignment5\myping\x64\Debug>myping www.sjtu.edu.cn
Pinging: www.sjtu.edu.cn [202.120.2.119] with 32 bytes of data.
Reply From [202.120.2.119]: bytes = 32 time < 1 ms TTL = 64
Reply From [202.120.2.119]: bytes = 32 time < 1 ms TTL = 64
Reply From [202.120.2.119]: bytes = 32 time < 1 ms TTL = 64
Reply From [202.120.2.119]: bytes = 32 time < 1 ms TTL = 64
Send 4 packages,  Receive 4 packages,  Loss 0(0.00%) packages.

E:\Desktop\秃头文件\●数据通信\WinSock_IS404\assignment5\myping\x64\Debug>
```

```
No.   Time         Source            Destination
   17 7.424799     61.151.178.213    192.168.1.109
   18 8.239396     192.168.1.109     202.120.2.119
   19 8.244233     202.120.2.119     192.168.1.109
   20 8.267265     192.168.1.109     192.168.1.255
   21 8.482432     61.151.178.213    192.168.1.109
   22 9.256878     192.168.1.109     202.120.2.119
   23 9.262655     202.120.2.119     192.168.1.109
   24 9.268497     192.168.1.109     192.168.1.255
   25 10.268714    192.168.1.109     192.168.1.255
```

```
Wireshark · 分组 18 · WLAN (ip)
> Frame 18: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\N
> Ethernet II, Src: IntelCor_2e:d8:0d (04:ea:56:2e:d8:0d), Dst: Tp-LinkT_45:48:e1 (68:77:24:
> Internet Protocol Version 4, Src: 192.168.1.109, Dst: 202.120.2.119
v Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xd2c5 [correct]
    [Checksum Status: Good]
    Identifier (BE): 53261 (0xd00d)
    Identifier (LE): 3536 (0x0dd0)
    Sequence Number (BE): 16407 (0x4017)
    Sequence Number (LE): 5952 (0x1740)
    [Response frame: 19]
  v Data (32 bytes)
      Data: 5151515151515151515151515151515151515151515151515151515151515151
      [Length: 32]

0000  68 77 24 45 48 e1 04 ea  56 2e d8 0d 08 00 45 00   hw$EH··· V.····E·
0010  00 3c e4 73 00 00 40 01  00 00 c0 a8 01 6d ca 78   ·<·s··@· ·····m·x
0020  02 77 08 00 d2 c5 d0 0d  40 17 51 51 51 51 51 51   ·w······ @·QQQQQQ
0030  51 51 51 51 51 51 51 51  51 51 51 51 51 51 51 51   QQQQQQQQ QQQQQQQQ
0040  51 51 51 51 51 51 51 51  51 51                     QQQQQQQQ QQ
```

在 cmd 中实现 ping 交大官网，wireshark 抓包 icmp



```
E:\Desktop\秃头文件\●数据通信\WinSock_IS404\assignment5\myping\x64\Debug>myping www.google.org
Pinging: www.google.org [216.239.32.27] with 32 bytes of data.
Request Time Out.
Request Time Out.
Request Time Out.
Request Time Out.
Send 4 packages,  Receive 0 packages,  Loss 4(100.00%) packages.
```

Ping 谷歌会被墙，丢包

三、问题与思考

1. 对比 IP 和 ICMP 传输：

ICMP 的全称是 Internet Control Message Protocol(互联网控制协议)，它是一种互联网套件，它用于 IP 协议中发送控制消息。也就是说，ICMP 是依靠 IP 协议来完成信息发送的，它是 IP 的主要部分，但是从体系结构上来讲，它位于 IP 之上，因为 ICMP 报文是承载在 IP 分组中的，就和 TCP 与 UDP 报文段作为 IP 有效载荷被承载那样。这也就是说，当主机收到一个指明上层协议为 ICMP 的 IP 数据报时，它会分解出该数据报的内容给 ICMP，就像分解数据报的内容给 TCP 和 UDP 一样。

ICMP 协议和 TCP、UDP 等协议不同，它不用于传输数据，只是用来发送消息。因为 IP 协议现在有两类版本：IPv4 和 IPv6 ，所以 ICMP 也有两个版本：ICMPv4 和 ICMPv6。

2. 几种错误：

11004：网路连接错误，可能是网址不对；

Timeout：比如访问外网时 ping 不通，需要挂 VPN；

Time<1ms：访问非常快，TTL 正常则无误，有可能是访问了局域网，比如访问 127.0.0.1 就是 time<1ms；

Ref:
Getting started with Winsock
https://msdn.microsoft.com/en-us/library/ms738545(v=vs.85).aspx
Winsock reference
https://msdn.microsoft.com/en-us/library/ms741416(v=vs.85).aspx