

# 数据通信作业

---

姓名：刘浩文      学号：517021911065      日期：2020/5/15

## 数据通信作业

### 一、实验名称及内容

### 二、实验过程和结果

环境

程序设计

程序流程图

程序使用

程序主体说明

数据结构

函数

实验结果

### 三、问题与思考

## 一、实验名称及内容

---

名称：利用 *Winsock* 完成类似于系统自带的 ping 远程主机的功能

内容：利用 *Winsock* 完成基于 **ICMP** 协议的 **ping** 程序，该程序完成类似于系统自带的 ping 远程主机的功能，可以直接 ping **IP** 地址，也可以自动进行域名解析，并且可以指定 ping 次数和统计 ping 结果，基本包含了系统自带的 ping 命令的基本功能。并且，若一台远程主机有多个 **IP**，则该程序会自动依次 ping 该主机所有 **IP**。用户可以通过该程序 ping 远程主机来测试连接性。

## 二、实验过程和结果

---

### 环境

物理主机系统：*macOS Catalina 10.15.4*

虚拟机系统：*Windows 10 专业版 x64*

计算机名：*691B*

虚拟机软件：*Parallels Desktop 15 for Mac Pro Edition, version 15.1.4 (47270)*

编程环境(IDE)：*Visual Studio 2019*

### 程序设计

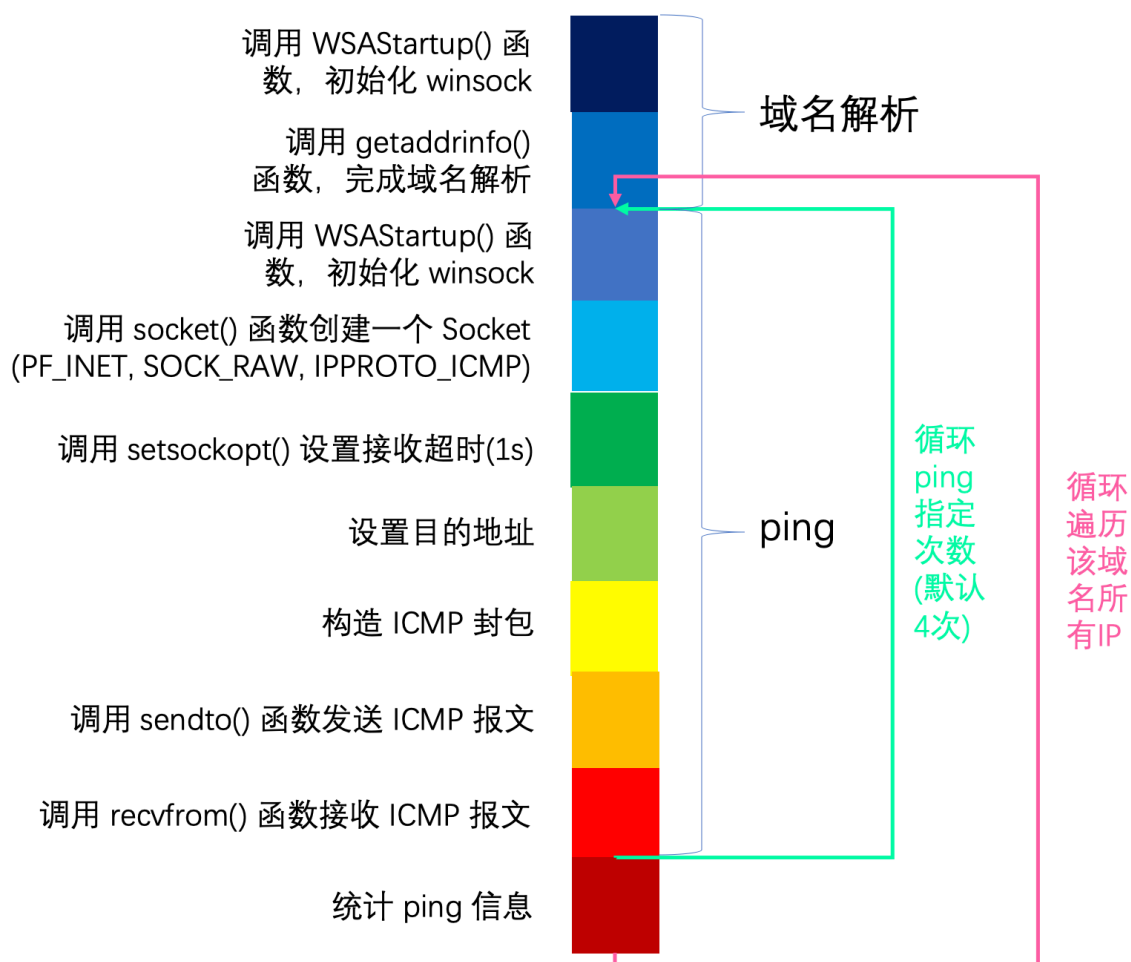
#### 1. 域名解析（获取远程主机名）

1. 调用 *WSAStartup()* 函数，初始化 *winsock*
2. 调用 *getaddrinfo()* 函数，获得指定 **IP** 或域名的主机信息，完成域名解析

## 2. ping 远程主机，循环指定次数，默认 4 次

1. 调用 `WSAStartup()` 函数，初始化 `winsock`
2. 调用 `socket()` 函数创建一个 **Socket (PF\_INET, SOCK\_RAW, IPPROTO\_ICMP)**
3. 调用 `setsockopt()` 设置接收超时(1s)
4. 设置目的地址
5. 构造 **ICMP** 封包
  1. 构造 **ICMP** 报头
  2. 在报头后填充数据，可以任意
  3. 计算校验和
6. 调用 `sendto()` 函数发送 **ICMP** 报文
7. 调用 `recvfrom()` 函数接收 **ICMP** 报文
8. 统计 ping 信息

## 程序流程图



## 程序使用

编译后在命令行运行可执行程序：

```
1 | myping.exe [IP/DN] ([times](default 4 times))
```

如:

```
1 | myping.exe baidu.com
```

```
1 | myping.exe baidu.com 10
```

```
1 | myping.exe 39.156.69.79 10
```

## 程序主体说明

### 数据结构

```
1 | #include <stdio.h>
2 | #include <time.h>
3 | #include <Winsock2.h>
4 | #include <ws2tcpip.h>
5 | #include <Windows.h>
6 | #include <sstream>
7 | #include <iostream>
8 | #include <string>
9 | using namespace std;
10 |
11 | #pragma comment (lib, "ws2_32.lib")
12 |
13 | // 2字节 对齐 sizeof(icmp_header) == 8
14 | // 这是ping 在wireshark抓包中的数据结构
15 | typedef struct icmp_header // ICMP报头
16 | {
17 |     unsigned char icmp_type; // 消息类型
18 |     unsigned char icmp_code; // 代码
19 |     unsigned short icmp_checksum; // 校验和
20 |     unsigned short icmp_id; // 用来惟一标识此请求的ID号，通常设置为进程ID
21 |     unsigned short icmp_sequence; // 序列号
22 | } icmp_header;
```

### 函数

```

1  u_short ss2n(string s)    // 将字符串转为数字，用来将argv指向的字符串类型的指定ping
    次数转为短整型
2  {
3      stringstream ss;
4      u_short u;
5      ss << s;
6      ss >> u;
7      return u;
8  }

```

```

1  // 计算校验和
2  unsigned short chsum(struct icmp_header* picmp, int len)
3  {
4      long sum = 0;
5      unsigned short* pusicmp = (unsigned short*)picmp;
6      while (len > 1)
7      {
8          sum += *(pusicmp++);
9          if (sum & 0x80000000)
10             sum = (sum & 0xffff) + (sum >> 16);
11         len -= 2;
12     }
13     if (len)
14         sum += (unsigned short)*(unsigned char*)pusicmp;
15     while (sum >> 16)
16         sum = (sum & 0xffff) + (sum >> 16);
17     return (unsigned short)~sum;
18 }

```

```

1  // ping 函数
2  static int respNum = 0;
3  static int minTime = 65535, maxTime = 0, sumTime = 0;
4  int ping(char* szDestIp)
5  {
6      //printf("destIp = %s\n",szDestIp);
7      int bRet = 1;
8
9      WSADATA wsaData;
10     int nTimeOut = 1000;//1s
11     char szBuff[ICMP_HEADER_SIZE + 32] = { 0 };
12     icmp_header* pIcmp = (icmp_header*)szBuff;
13     char icmp_data[32] = { 0 };
14
15     WSStartup(MAKEWORD(2, 2), &wsaData);
16     // 创建原始套接字
17     SOCKET s = socket(PF_INET, SOCK_RAW, IPPROTO_ICMP);
18
19     // 设置接收超时

```

```

20     setsockopt(s, SOL_SOCKET, SO_RCVTIMEO, (char const*)&nTimeOut,
sizeof(nTimeOut));
21
22     // 设置目的地址
23     sockaddr_in dest_addr;
24     dest_addr.sin_family = AF_INET;
25     inet_pton(AF_INET, szDestIp, &dest_addr.sin_addr);
26     dest_addr.sin_port = htons(0);
27
28     // 构造ICMP封包
29     pIcmp->icmp_type = ICMP_ECHO_REQUEST;
30     pIcmp->icmp_code = 0;
31     pIcmp->icmp_id = (USHORT)::GetCurrentProcessId();
32     pIcmp->icmp_sequence = 0;
33     pIcmp->icmp_checksum = 0;
34
35     // 填充数据, 可以任意
36     memcpy((szBuff + ICMP_HEADER_SIZE),
"abcdelmnopqrstuvwxyzwiamekakuaactor", 32);
37
38     // 计算校验和
39     pIcmp->icmp_checksum = chsum((struct icmp_header*)szBuff,
sizeof(szBuff));
40
41     sockaddr_in from_addr;
42     char szRecvBuff[1024];
43     int nLen = sizeof(from_addr);
44     int ret, flag = 0;
45
46     DWORD start = GetTickCount();
47     ret = sendto(s, szBuff, sizeof(szBuff), 0, (SOCKADDR*)&dest_addr,
sizeof(SOCKADDR));
48     //printf("ret = %d ,errorCode:%d\n",ret ,WSAGetLastError() );
49
50     int i = 0;
51     //这里一定要用while循环, 因为recvfrom 会接受到很多报文, 包括 发送出去的报文也会
被收到! 不信你可以用 wireshark 抓包查看, 这个问题纠结来了一晚上 才猜想出来!
52     while (1) {
53         if (i++ > 5) { // icmp报文 如果到不了目标主机, 是不会返回报文, 多尝试几次
接受数据, 如果都没收到 即请求失败
54             flag = 1;
55             break;
56         }
57         memset(szRecvBuff, 0, 1024);
58         //printf("errorCode1:%d\n",WSAGetLastError() );
59         int ret = recvfrom(s, szRecvBuff, MAXBYTE, 0,
(SOCKADDR*)&from_addr, &nLen);
60         //printf("errorCode2:%d\n",WSAGetLastError() );
61         //printf("ret=%d,%s\n",ret,inet_ntoa(from_addr.sin_addr)) ;

```

```

62         //接受到 目标ip的 报文
63         char str[INET_ADDRSTRLEN];
64         char* ptr = (char*)inet_ntop(AF_INET, &from_addr.sin_addr, str,
sizeof(str));
65         if (strcmp(ptr, szDestIp) == 0) {
66             respNum++;
67             break;
68         }
69     }
70
71     DWORD end = GetTickCount();
72     DWORD time = end - start;
73
74     if (flag) {
75         printf("请求超时。 \n");
76         return bRet;
77     }
78     sumTime += time;
79     if (minTime > time) {
80         minTime = time;
81     }
82     if (maxTime < time) {
83         maxTime = time;
84     }
85
86
87     // Windows的原始套接字开发，系统没有去掉IP协议头，需要程序自己处理。
88     // ip头部的第一个字节（只有1个字节不涉及大小端问题），前4位表示ip协议版本号，后4
位表示IP头部长度(单位为4字节)
89     char ipInfo = szRecvBuff[0];
90     // ipv4头部的第9个字节为TTL的值
91     unsigned char ttl = szRecvBuff[8];
92     //printf("ipInfo = %x\n",ipInfo);
93
94
95     int ipVer = ipInfo >> 4;
96     int ipHeadLen = ((char)(ipInfo << 4) >> 4) * 4;
97     if (ipVer == 4) {
98         //ipv4
99         //printf("ipv4 len = %d\n",ipHeadLen);
100         // 跨过ip协议头，得到ICMP协议头的位置，不过是网络字节序。
101         // 网络字节序 是大端模式 低地址 高位字节 高地址 低位字节。-> 转换为 本地字
节序 小端模式 高地址高字节 低地址低字节
102         icmp_header* icmp_rep = (icmp_header*)(szRecvBuff + ipHeadLen);
103         //由于校验和是 2个字节 涉及大小端问题，需要转换字节序
104         unsigned short checksum_host = ntohs(icmp_rep->icmp_checksum);//
转主机字节序
105

```

```

106         //printf("type = %d , checksum_host =
107         %x\n",icmp_rep,checksum_host);
108
109         if (icmp_rep->icmp_type == 0) { //回显应答报文
110             printf("来自 %s 的回复: 字节=32 时间=%2dms TTL=%d checksum=0x%x
111             \n", szDestIp, time, ttl, checksum_host);
112         }
113         else {
114             bRet = 0;
115             printf("请求超时。type = %d\n", icmp_rep->icmp_type);
116         }
117         else {
118             // ipv6 icmpv6 和 icmpv4 不一样, 要做对应的处理
119             //printf("ipv6 len = %d\n",ipLen);
120         }
121         return bRet;
122     }

```

```

1 // 主函数, 包括完成域名解析功能
2 int main(int argc, char** argv)
3 {
4     if (argc < 2) {
5         printf("please input:myting ipaddr!\n");
6         return 0;
7     }
8     u_short times;
9     if (argv[2])
10         times = ss2n(argv[2]);
11     else times = 4;
12
13     struct addrinfo* result = NULL;
14     struct addrinfo* hostEntry = NULL;
15     struct addrinfo hints;
16     struct sockaddr_in addr;
17     ZeroMemory(&hints, sizeof(hints));
18     hints.ai_family = AF_INET; /* Allow IPv4 */
19     hints.ai_flags = AI_PASSIVE; /* For wildcard IP address */
20     hints.ai_protocol = 0; /* Any protocol */
21     hints.ai_socktype = SOCK_STREAM;
22
23     char** ppAlias = NULL; // 主机别名
24     char** ppAddr = NULL; // 点分十进制ip地址
25     WORD sockVersion = MAKEWORD(2, 2);
26     WSADATA wsaData;
27     if (WSAStartup(sockVersion, &wsaData) != 0) {
28         return false;
29     }
30

```

```

31     DWORD dwRetVal;
32     dwRetVal = getaddrinfo(argv[1], NULL, &hints, &result); // 域名解析
33
34     if (result == NULL) {
35         cout << "无法解析域名。\\n";
36         return 0;
37     }
38
39     for (hostEntry = result; hostEntry != NULL; hostEntry = hostEntry-
>ai_next)
40     {
41         addr = *(struct sockaddr_in*)hostEntry->ai_addr;
42         char str[INET_ADDRSTRLEN];
43         char* ptr = (char*)inet_ntop(AF_INET, &addr.sin_addr, str,
sizeof(str));
44         printf("\\n正在 Ping %s 具有 32 字节的数据:\\n", ptr);
45         int i = 0;
46
47         while (i < times)
48         {
49             int result = ping(ptr);
50             Sleep(500);
51             i++;
52         }
53
54         printf("\\n%s 的 Ping 统计信息:\\n", argv[1]);
55         printf("    数据包: 已发送 = %d, 已接收 = %d, 丢失 = %d (%d%% 丢失),
\\n", i, respNum, i - respNum, (i - respNum) * 100 / i);
56         if (i - respNum >= 4) {
57             return 0;
58         }
59         printf("往返行程的估计时间(以毫秒为单位):\\n");
60         printf("    最短 = %dms, 最长 = %dms, 平均 = %dms\\n", minTime,
maxTime, sumTime / respNum);
61         minTime = 65535, maxTime = 0, sumTime = 0;
62         respNum = 0;
63     }
64     return 0;
65 }

```

## 实验结果



```
C:\Windows\system32\cmd.exe
C:\Users\hongxing\source\repos\myping\x64\Debug>myping.exe baidu.com

正在 Ping 39.156.69.79 具有 32 字节的数据:
来自 39.156.69.79 的回复: 字节=32 时间=47ms TTL=128 checksum=0x7d86
来自 39.156.69.79 的回复: 字节=32 时间=47ms TTL=128 checksum=0x7d86
来自 39.156.69.79 的回复: 字节=32 时间=31ms TTL=128 checksum=0x7d86
来自 39.156.69.79 的回复: 字节=32 时间=47ms TTL=128 checksum=0x7d86

baidu.com 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 31ms, 最长 = 47ms, 平均 = 43ms

正在 Ping 220.181.38.148 具有 32 字节的数据:
来自 220.181.38.148 的回复: 字节=32 时间=62ms TTL=128 checksum=0x7d86
来自 220.181.38.148 的回复: 字节=32 时间=63ms TTL=128 checksum=0x7d86
来自 220.181.38.148 的回复: 字节=32 时间=62ms TTL=128 checksum=0x7d86
来自 220.181.38.148 的回复: 字节=32 时间=63ms TTL=128 checksum=0x7d86

baidu.com 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 62ms, 最长 = 63ms, 平均 = 62ms

C:\Users\hongxing\source\repos\myping\x64\Debug>myping.exe baidu.com 2

正在 Ping 39.156.69.79 具有 32 字节的数据:
来自 39.156.69.79 的回复: 字节=32 时间=31ms TTL=128 checksum=0x699a
来自 39.156.69.79 的回复: 字节=32 时间=47ms TTL=128 checksum=0x699a

baidu.com 的 Ping 统计信息:
    数据包: 已发送 = 2, 已接收 = 2, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 31ms, 最长 = 47ms, 平均 = 39ms

正在 Ping 220.181.38.148 具有 32 字节的数据:
来自 220.181.38.148 的回复: 字节=32 时间=63ms TTL=128 checksum=0x699a
来自 220.181.38.148 的回复: 字节=32 时间=62ms TTL=128 checksum=0x699a

baidu.com 的 Ping 统计信息:
    数据包: 已发送 = 2, 已接收 = 2, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 62ms, 最长 = 63ms, 平均 = 62ms

C:\Users\hongxing\source\repos\myping\x64\Debug>

C:\Windows\system32\cmd.exe
C:\Users\hongxing\source\repos\myping\x64\Debug>myping.exe 39.156.69.79

正在 Ping 39.156.69.79 具有 32 字节的数据:
来自 39.156.69.79 的回复: 字节=32 时间=32ms TTL=128 checksum=0x119d
来自 39.156.69.79 的回复: 字节=32 时间=31ms TTL=128 checksum=0x119d
来自 39.156.69.79 的回复: 字节=32 时间=47ms TTL=128 checksum=0x119d
来自 39.156.69.79 的回复: 字节=32 时间=32ms TTL=128 checksum=0x119d

39.156.69.79 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 31ms, 最长 = 47ms, 平均 = 35ms

C:\Users\hongxing\source\repos\myping\x64\Debug>myping.exe 39.156.69.79 10

正在 Ping 39.156.69.79 具有 32 字节的数据:
来自 39.156.69.79 的回复: 字节=32 时间=219ms TTL=128 checksum=0x918f
来自 39.156.69.79 的回复: 字节=32 时间=47ms TTL=128 checksum=0x918f
来自 39.156.69.79 的回复: 字节=32 时间=47ms TTL=128 checksum=0x918f
来自 39.156.69.79 的回复: 字节=32 时间=46ms TTL=128 checksum=0x918f
来自 39.156.69.79 的回复: 字节=32 时间=32ms TTL=128 checksum=0x918f
来自 39.156.69.79 的回复: 字节=32 时间=47ms TTL=128 checksum=0x918f
来自 39.156.69.79 的回复: 字节=32 时间=93ms TTL=128 checksum=0x918f
来自 39.156.69.79 的回复: 字节=32 时间=47ms TTL=128 checksum=0x918f
来自 39.156.69.79 的回复: 字节=32 时间=47ms TTL=128 checksum=0x918f

39.156.69.79 的 Ping 统计信息:
    数据包: 已发送 = 10, 已接收 = 10, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 32ms, 最长 = 219ms, 平均 = 65ms

C:\Users\hongxing\source\repos\myping\x64\Debug>myping.exe google.com

正在 Ping 172.217.160.78 具有 32 字节的数据:
请求超时。
请求超时。
请求超时。
请求超时。

google.com 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 0, 丢失 = 4 (100% 丢失),

C:\Users\hongxing\source\repos\myping\x64\Debug>
```

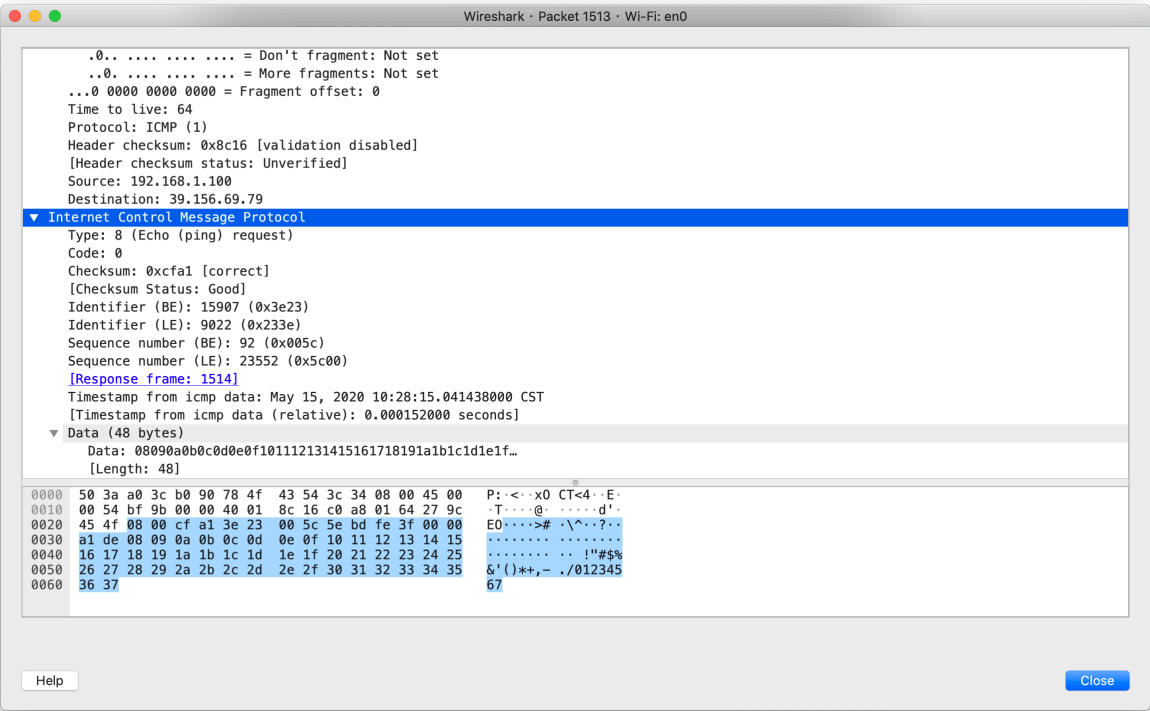
上图一共成功进行了五个实验。第一个实验是 ping 一个域名，选择 **baidu.com**，不设置 ping 次数，结果显示由于该域名解析出两个 IP，于是每个 IP 均成功 ping 了四次，四次均可达，连通性良好，最后对每个 IP 的 ping 结果都进行了正确统计。第二个实验仍 ping **baidu.com**，设置 ping 次数为 2，结果显示每个 IP 均成功 ping 了两次，两次均可达，连通性良好，最后对每个 IP 的 ping 结果都进行了正确统计。第三个和第四个实验是 ping **baidu.com** 其中一个域名 39.156.69.79，前者不设置 ping 次数，后者设置 ping 10 次，均成功并符合程序设计的逻辑。最后一个实验是 ping **google.com**，由于该域名在中国内地被封禁，所以理应无法连通，实验结果也证明了这一点。最后用 macOS 的终端的 ping 命令来测试一下上述结果进行对照实验，结果一模一样，所以本设计的程序能够完成系统自带 ping 命令的基本功能。

```
hongxing — -bash — 80x24

declare -x XPC_SERVICE_NAME=""

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) liuhaowendeMacBook-Pro:~ hongxing$ ping baidu.com
PING baidu.com (39.156.69.79): 56 data bytes
64 bytes from 39.156.69.79: icmp_seq=0 ttl=53 time=28.620 ms
64 bytes from 39.156.69.79: icmp_seq=1 ttl=53 time=33.035 ms
64 bytes from 39.156.69.79: icmp_seq=2 ttl=53 time=31.909 ms
^C
--- baidu.com ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 28.620/31.188/33.035/1.873 ms
(base) liuhaowendeMacBook-Pro:~ hongxing$ ping google.com
PING google.com (93.46.8.90): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2
Request timeout for icmp_seq 3
^C
--- google.com ping statistics ---
5 packets transmitted, 0 packets received, 100.0% packet loss
(base) liuhaowendeMacBook-Pro:~ hongxing$
```

### 三、问题与思考



该实验最大的难点是 **ICMP** 请求报文的构造。虽然在《计算机通信网络》课程上已经详细介绍过 **ICMP** 报文的结构，但自己用 C++ 构造一个还是有难度的。为了能够正确地完成 **ICMP** 请求报文的构造，我使用 *Wireshark* 抓取了一些使用系统 ping 命令发出的 **ICMP** 请求报文，依次查看各项的含义、类型、大小，以及在报文中的位置，再结合网上的资料，成功构造了 **ICMP** 数据包。第二个困难是域名解析，但由于之前已经做过类似实验，所以也顺利解决了。