计算机网络实验6-实现ping命令实验报告

161910110万晔

一、实验目的及要求

1.1 实验目的

- （1）熟悉网络套接字编程（socket 编程技术）
- （2）了解网络的结构
- （3）了解网络传输底层协议（ICMP 协议）

1.2 实验要求

- （1）要求学生掌握利用 Socket 进行编程的技术
- （2）不能采用现有的工具，必须自己一步一步，根据协议进行操作
- （3）了解 ping 报文的格式和步骤，要求符合 ICMP 协议并组建报文
- （4）在一秒钟内，如果收到，则为成功，如果收不到，则失败（ping 功能）
- （5）必须采用图形界面，查看收到回应的结果
- （6）可以通过程序，查看子网中有哪些主机可以 ping 通（Find 功能）

二、实验思路

2.1 实验原理

（1） ping 命令的作用与原理

简单来说，「ping」是用来探测本机与网络中另一主机之间是否可达的命令，如果两台主机之间ping不通，则表明这两台主机不能建立起连接。ping是定位网络通不通的一个重要手段。

ping 命令是基于 ICMP 协议来工作的，「 ICMP 」全称为 Internet 控制报文协议（ Internet Control Message Protocol）。ping 命令会发送一份ICMP回显请求报文给目标主机，并等待目标主机返回ICMP回显应答。因为ICMP协议会要求目标主机在收到消息之后，必须返回ICMP应答消息给源主机，如果源主机在一定时间内收到了目标主机的应答，则表明两台主机之间网络是可达的。

举一个例子来描述「ping」命令的工作过程：

假设有两个主机，主机A（192.168.0.1）和主机B（192.168.0.2），现在我们要监测主机A和主机B之间网络是否可达，那么我们在主机A上输入命令：ping 192.168.0.2

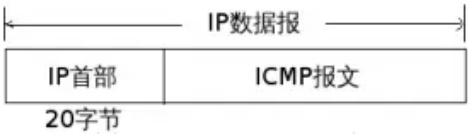
- 此时，ping命令会在主机A上构建一个 ICMP的请求数据包（数据包里的内容后面再详述），然后ICMP协议会将这个数据包以及目标IP（192.168.0.2）等信息一同交给IP层协议。
- IP层协议得到这些信息后，将源地址（即本机IP）、目标地址（即目标IP：192.168.0.2）、再加上一些其它的控制信息，构建成一个IP数据包。

- IP数据包构建完成后，还不够，还需要加上MAC地址，因此，还需要通过ARP映射表找出目标IP所对应的MAC地址。当拿到了目标主机的MAC地址和本机MAC后，一并交给数据链路层，组装成一个数据帧，依据以太网的介质访问规则，将它们传送出出去。
- 当主机B收到这个数据帧之后，会首先检查它的目标MAC地址是不是本机，如果是就接收下来处理，接收之后会检查这个数据帧，将数据帧中的IP数据包取出来，交给本机的IP层协议，然后IP层协议检查完之后，再将ICMP数据包取出来交给ICMP协议处理，当这一步也处理完成之后，就会构建一个ICMP应答数据包，回发给主机A
- 在一定的时间内，如果主机A收到了应答包，则说明它与主机B之间网络可达，如果没有收到，则说明网络不可达。除了监测是否可达以外，还可以利用应答时间和发起时间之间的差值，计算出数据包的延迟耗时。

通过ping的流程可以发现，ICMP协议是这个过程的基础，是非常重要的，因此下面就把ICMP协议再详细解释一下。

（2）ICMP 原理介绍

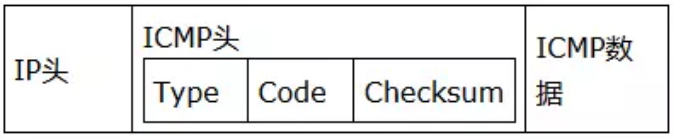
我们知道，ping命令是基于ICMP协议来实现的。那么我们再来看下图，就明白了ICMP协议又是通过IP协议来发送的，即ICMP报文是封装在IP包中。



IP协议是一种无连接的，不可靠的数据包协议，它并不能保证数据一定被送达，那么我们要保证数据送到就需要通过其它模块来协助实现，这里就引入的是ICMP协议。

当传送的IP数据包发送异常的时候，ICMP就会将异常信息封装在包内，然后回传给源主机。

将上图再细拆一下可见：



继续将ICMP协议模块细拆:

由图可知，ICMP数据包由8bit的类型字段和8bit的代码字段以及16bit的校验字段再加上选项数据组成。

类型 TYPE	编码 CODE	检验和 CHECKSUM
标志符 ID	顺序号 SEQ	

代码结构为：

```
class ICMPHeader
{
    public:
        u_char type;             // 类型
        u_char code;             // 代码
        u_short check_sum;       // 校验和
        u_short id;              // 标示符 标识本进程
        u_short seq;             // 序列号
};
```

各字段说明：

- **类型**：占一字节，标识ICMP报文的类型，目前已定义了14种，Ping操作中ICMP报文的回显请求报文类型字段值为8和回显应答报文类型字段值为0；
- **代码**：占一字节，标识对应ICMP报文的代码。它与类型字段一起共同标识了ICMP报文的详细类型。Ping操作中ICMP报文的回显请求报文（类型，代码）字段值为（8，0）和回显应答报文类型字段值为（0，0）；
- **校验和**：这是对包括ICMP报文数据部分在内的整个ICMP数据报的校验和，以检验报文在传输过程中是否出现了差错。
- **标识符**：占两字节，用于标识本ICMP进程，当同时与多个目的通信时，通过本字段来区分，但仅适用于回显请求和应答ICMP报文，对于目标不可达ICMP报文和超时ICMP报文等，该字段的值为0。
- **序列号**：占两字节，标识本地到目的的数据包序号，一般从序号1开始。

ICMP协议大致可分为两类：

- 查询报文类型
- 差错报文类型

ICMP报文类型	类型的值	ICMP的报文类型
差错报文类型	3	终点不可达
	4	源点抑制
	11	时间超过
	12	参数问题
	5	改变路由
询问报文	8	回送请求
	0	回送回答
	13	时间戳请求
	14	时间戳会带

- 查询报文类型：
  - 查询报文主要应用于：ping查询、子网掩码查询、时间戳查询等等
- 差错报文类型：

- 差错报文主要产生于当数据传送发送错误的时候

Ping 命令只使用众多 ICMP 报文中的两种：回显请求(ICMP_ECHO)和回显应答(ICMP_ECHOREPLY)。

```
//回显请求 ICMP_ECHO
struct EchoRequest
{
    ICMPHeader icmp_header;          //ICMP头部
    unsigned long long time;        //记录ping时间
};

//回显应答 ICMP_ECHOREPLY
struct EchoResponse
{
    IPHeader ip_header;
    EchoRequest echo_request;
};
```

当传送IP数据包发生错误的时候（例如 主机不可达），ICMP协议就会把错误信息封包，然后传送回源主机，那么源主机就知道该怎么处理了。

（3）IP 数据报格式

如图，为IP数据报格式：

版本号 VER	IP 报头长度 IHL	服务类型 TOS	数据包长度 TL
报文标志 ID	报文标志 F		分段偏移 FO
生存时间 TTL	协议号 PORT		报头校验和
源地址			
目标地址			
任选项和填充位			

IP 数据报文由首部和数据两部分组成。首部的前一部分是固定长度，共 20 字节，是所有 IP 数据报必须具有的。在首部的固定部分的后面是一些可选字段，其长度是可变的。

每个 IP 数据报都以一个 IP 报头开始。源计算机构造这个 IP 报头，而目的计算机利用 IP 报头中封装的信息处理数据。

代码片段

```
class IPHeader
{
    public:
        //u_char 占1个字节
        //u_short 占两个字节
        //u_char version;         // 版本
        //u_char head_len;        // 首部长度
        u_char ver_headlen;       // 版本+首部长度
        u_char service;           // 服务类型
        u_short total_len;        // 总长度
        u_short id;               // 标识符
```

```
        u_short flag;           // 标记+片偏移
        u_char ttl;             // 存活时间
        u_char protocol;        // 协议
        u_short check_sum;      // 首部校验和
        u_int src_IP;           // 源IP地址
        u_int dst_IP;           // 目的IP地址
};
```

各字段说明：

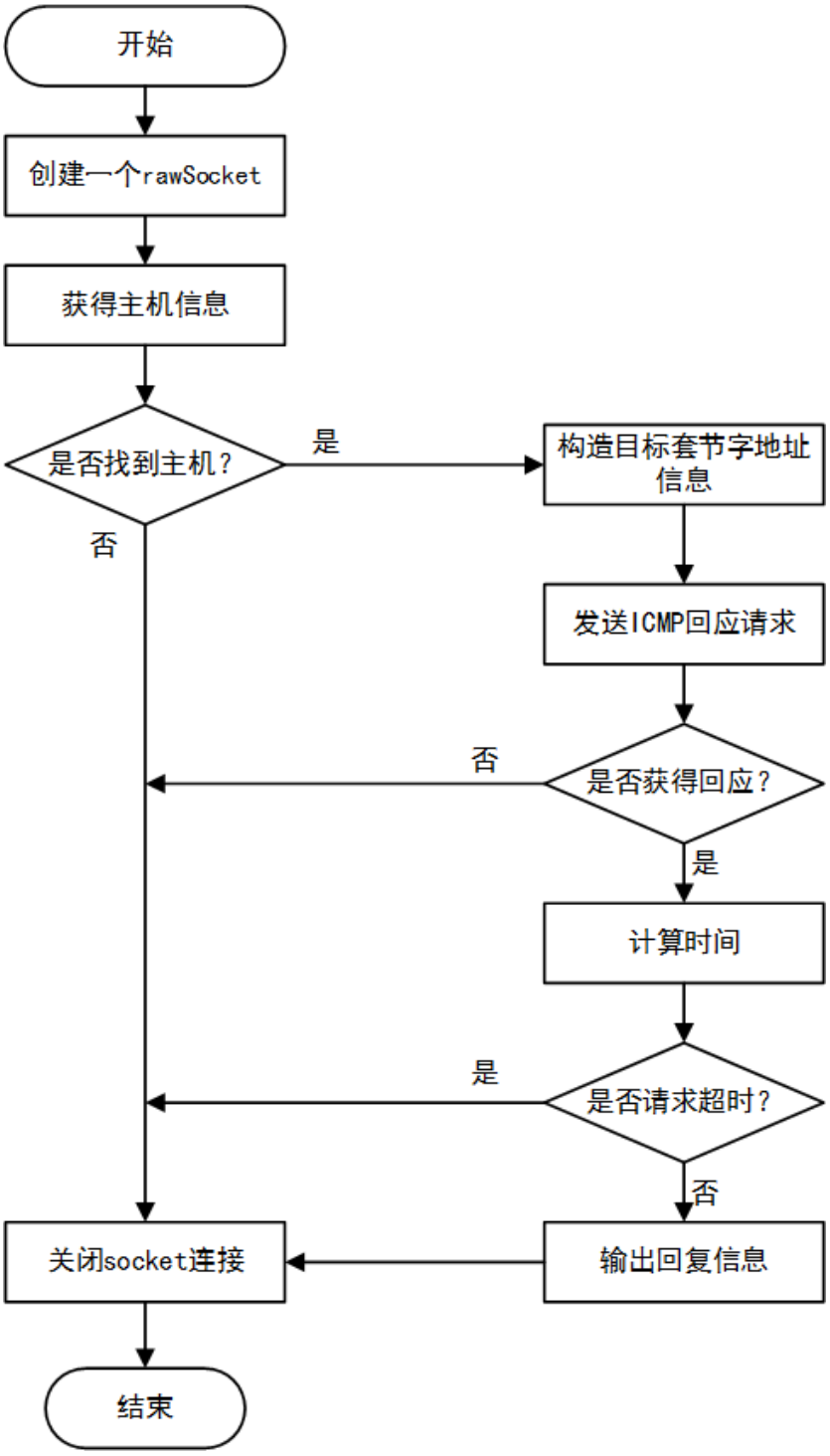
- **版本**：占 4 位，表示 IP 协议的版本。通信双方使用的 IP 协议版本必须一致。目前广泛使用的IP协议版本号为 4，即 IPv4。
- **首部长度**：最常用的首部长度就是 20 字节（即首部长度为 0101）。
- **区分服务**：也被称为服务类型，占 8 位，用来获得更好的服务。
- **总长度**：首部和数据之和，单位为字节。总长度字段为 16 位，因此数据报的最大长度为 2^16-1=65535 字节。
- **标识符**：用来标识数据报，占 16 位。具有相同的标识字段值的分片报文会被重组成原来的数据报。
- **标志**：占 3 位。第一位未使用，其值为 0。第二位称为 DF（不分片），表示是否允许分片。取值为 0 时，表示允许分片；取值为 1 时，表示不允许分片。第三位称为 MF（更多分片），表示是否还有分片正在传输，设置为 0 时，表示没有更多分片需要发送，或数据报没有分片。
- **片偏移**：占 13 位。当报文被分片后，该字段标记该分片在原报文中的相对位置。片偏移以 8 个字节为偏移单位。
- **生存时间(TTL)**：表示数据报在网络中的寿命，占 8 位。路由器在转发数据报之前，先把 TTL 值减 1。若 TTL 值减少到 0，则丢弃这个数据报，不再转发。因此，TTL 指明数据报在网络中最多可经过多少个路由器。
- **协议**：表示该数据报文所携带的数据所使用的协议类型，占 8 位。该字段可以方便目的主机的 IP 层知道按照什么协议来处理数据部分。
- **首部校验和**：用于校验数据报的首部，占 16 位。
- **源地址**：表示数据报的源 IP 地址，占 32 位。
- **目的地址**：表示数据报的目的 IP 地址，占 32 位。

2.2 实现功能

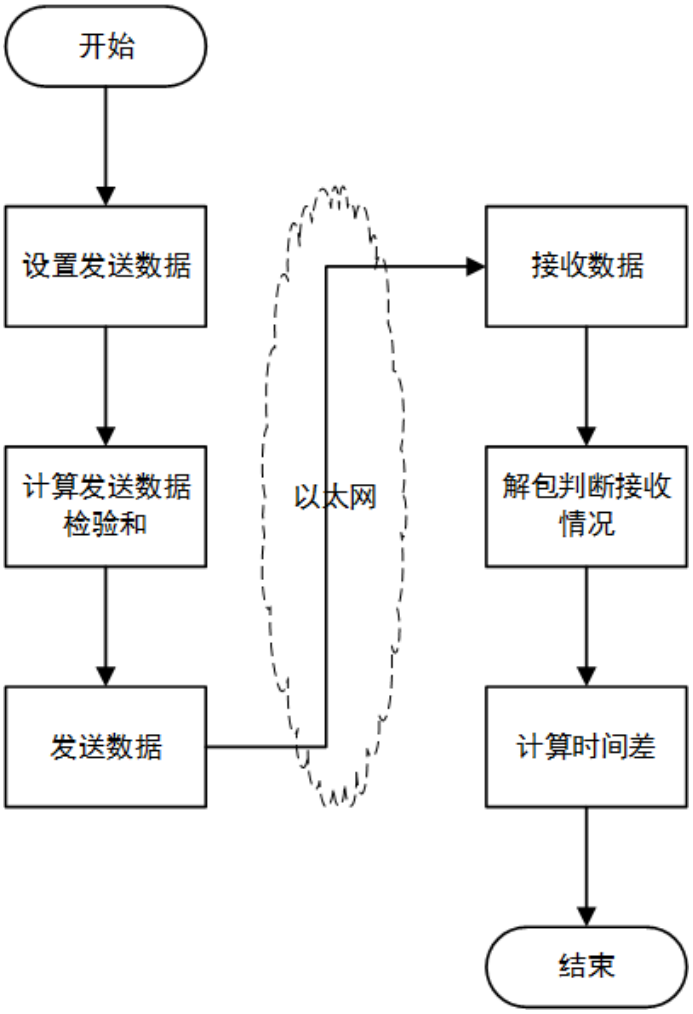
- 可以通过程序模拟对输入的目的地址进行 Ping 命令，在一秒钟内，如果收到，则为成功，如果收不到，则失败，打印输出该过程的信息，显示在图形化界面上
- 可以通过程序，查看子网中有哪些主机可以 Ping 通，打印输出该过程的信息，显示在图形化界面上

2.3 实现思路

实验整体框架示意图


开始
创建一个rawSocket
获得主机信息
是否找到主机？
是
构造目标套节字地址信息
否
发送ICMP回应请求
否
是否获得回应？
是
计算时间
是
是否请求超时？
否
关闭socket连接
输出回复信息
结束

实验主要流程图



三、主要代码分析

3.1 主要结构体/类

IPHeader　//IP 头

```
class IPHeader
{
    public:
        //u_char 占1个字节
        //u_short 占两个字节
        //u_char version;         // 版本
        //u_char head_len;        // 首部长度
        u_char ver_headlen;       // 版本+首部长度
        u_char service;           // 服务类型
        u_short total_len;        // 总长度
        u_short id;               // 标识符
        u_short flag;             // 标记+片偏移
        u_char ttl;               // 存活时间
        u_char protocol;          // 协议
        u_short check_sum;        // 首部校验和
        u_int src_IP;             // 源IP地址
        u_int dst_IP;             // 目的IP地址
};
```

`ICMPHeader` //ICMP 头

```
class ICMPHeader
{
    public:
        u_char type;              // 类型
        u_char code;              // 代码
        u_short check_sum;        // 校验和
        u_short id;               // 标示符 标识本进程
        u_short seq;              // 序列号
};
```

`Ping` //Ping 主类

```
class Ping
{
    public:
        char in[100];                                   //从输入框读入
        sockaddr_in dst_IP;                             //目标IP
        struct hostent *host;                           //主机
        IPHeader iPHeader;                             //IP头
        ICMPHeader iCMPHeader;                          //ICMP头
        SOCKET sock;                                    //套接字
        char message[5000];                             // Ping 内部的一些信息
        bool received;                                  // 是否收到
        bool getIP();                                   // 得到ping的IP地址
        bool send();                                    // 发送ICMP报文请求
        bool receive();                                 // 接收ICMP报文,解析并回显
        u_short check_sum(u_short* buffer, int len);    // 计算校验和

};
```

`sockaddr_in`：用来处理网络通信的地址；

`hostent`：记录主机的信息，包括主机名、别名、地址类型、地址长度和地址列表；

`sockaddr_in`和`hostent`在`getIP()`函数中使用：

```
bool Ping::getIP() //获取IP地址
{
    host = gethostbyname(in);   //in 为输入框输入的地址
    if (host == NULL)
    {
        return false;
    }
    dst_IP.sin_family = AF_INET; //IPv4格式
    dst_IP.sin_addr.S_un.S_addr = *(u_long*)host->h_addr;
    puts(host->h_name);
    puts(inet_ntoa(*(struct in_addr*)host->h_addr_list[0]));  //inet_ntoa()将网络地址转换成“.”点隔的字符串格式
    return true;
}
```

EchoRequest //ICMP时间戳请求报文

```
struct EchoRequest
{
    ICMPHeader icmp_header;                    //ICMP头部
    unsigned long long time;                   //记录ping时间
};
```

PingThread //多线程类

```
class PingThread : public QThread
{
    Q_OBJECT
    public:
        PingThread(char addr[100]);
    signals:
        void isDone(Ping);  //处理完成信号
    protected:
        char addr[100];
        void run();//通过start()间接调用

};
```

3.2 函数

getIP() //获取IP地址

```
bool Ping::getIP() //获取IP地址
{
    host = gethostbyname(in);   //in 为输入框输入的地址
    if (host == NULL)
    {
        return false;
    }
    dst_IP.sin_family = AF_INET; //IPv4格式
    dst_IP.sin_addr.S_un.S_addr = *(u_long*)host->h_addr;
    puts(host->h_name);
    puts(inet_ntoa(*(struct in_addr*)host->h_addr_list[0]));  //inet_ntoa()将网络地址转换成“.”点隔的字符串格式
    return true;
}
```

send() //发送数据包

```
bool Ping::send() //发送数据包
{
    if(getIP()==false) //获取IP失败
    {
        return false;
    }
    static int id = 1;
    static int seq = 1;
    EchoRequest req;
    req.time = GetTickCount(); //从0开始计时，返回自设备启动后的毫秒数
    req.icmp_header.type = 8;  //ICMP_ECHO
    req.icmp_header.code = 0;  //编码
    id = ::GetCurrentProcessId();//获取当前的进程ID
    req.icmp_header.id = id;//id++;
    req.icmp_header.seq = seq++;//序号加一
    req.icmp_header.check_sum = check_sum((u_short*)&req, sizeof(EchoRequest));//校验和字段

    int re = sendto(sock, (char*)&req, sizeof(req), 0, (sockaddr*)&dst_IP, sizeof(dst_IP));//将数据由指定的socket 传给目标主机
    // 成功则返回实际传送出去的字符数，失败返回 -1，
    // message 为此次ping的一些信息，输出到
    if(re == SOCKET_ERROR)  //SOCKET_ERROR = -1
    {
        strcat(message,"发送错误，错误码:");
        char temp[10];
        sprintf(temp,"%d",WSAGetLastError()); //WSAGetLastError()返回该线程上一次Sockets API函数调用时的错误代码,即sendto()函数的错误调用
        strcat(message,temp);
        strcat(message,"\n");
        return false; //发送失败
    }
    if(receive())
    {
        received=true;//接收成功
```

```
    }
    else
    {
        received=false;//接收失败
    }
    return true;    //发送成功
}
```

check_sum() 检验和计算

```
u_short Ping::check_sum(u_short* buf, int len) //校验和计算
{
    unsigned int sum=0;
    unsigned short *cbuf;

    cbuf=(unsigned short *)buf;

    while(len>1)
    {
        sum+=*cbuf++;
        len-=2;
    }
    if(len)
    {
        sum+=*(unsigned char *)cbuf;
    }
    sum=(sum>>16)+(sum & 0xffff);
    sum+=(sum>>16);
    return ~sum;
}
```

receive() //接收并分析返回的数据包

```
bool Ping::receive() //接收并分析返回的数据包
{
    int timeout = 1000;//设置超时的时间
    if(setsockopt(sock,SOL_SOCKET,SO_RCVTIMEO,(char *)&timeout,sizeof(timeout)) ==
SOCKET_ERROR) //设置套接口
    {
        strcat(message,"接收设置错误，错误码:");//设置套接口返回错误
        char temp[10];
        sprintf(temp,"%d",WSAGetLastError());
        strcat(message,temp);
        strcat(message,"\n");
        return false;
    }
    char temp[100];
    strcat(message,"来自 ");
    strcat(message,inet_ntoa(*(struct in_addr*)host->h_addr_list[0]));//地址
    strcat(message," 的回复: ");
    EchoResponse* res=new EchoResponse;
    int size = sizeof(sockaddr);
```

```
    int re = recvfrom(sock, (char*)res, sizeof(EchoResponse), 0, (sockaddr*)&dst_IP, &size);//接收到的返回的套接字
    if (re == SOCKET_ERROR)//出错
    {
        int code = WSAGetLastError();
        if(code==10060)
        {
            strcat(message,"请求超时。\n");
        }
        return false;
    }
    unsigned long long time = GetTickCount() - res->echo_request.time;//两个相减即为TTL时间
    int type = res->echo_request.icmp_header.type;
    int code = res->echo_request.icmp_header.code;
    char TTL[10]={'\0'};
    sprintf(TTL,"%d",(int)res->ip_header.ttl);
    delete res;
    if(type==0&&code==0)//输出到message信息中
    {
        strcat(message,"字节=32 时间=");
        sprintf(temp,"%I64u",time);
        strcat(message,temp);
        strcat(message,"ms TTL=");
        strcat(message,TTL);
        strcat(message,"\n");
        return true;
    }
    else
    {
        strcat(message,"请求超时");
        strcat(message,"\n");
        return false;
    }

}
```

3.3 图像界面设计函数

下面为图形界面：

地址栏：Addr

包数栏：Addr_2

子网栏：Addr_3

执行单次Ping命令按钮：Button1

查询子网中可ping通主机按钮：Button2

结果栏：Result

Ping 命令信号槽：

```
void MainWindow::on_PingButton1_clicked()
{
    ui->PingButton1->setDisabled(true);
    ui->PingButton2->setDisabled(true);
    numSend=0;
    numReceive=0;
    times=4;
    strcat(message,"\n");
    char temp[100];//用来存储ping地址
    strcpy(temp,ui->Addr->text().toLatin1().data()); //ping 地址
    char tmp[10];//用来存储ping包数
```

```
    strcpy(tmp,ui->Addr_2->text().toLatin1().data()); //ping 包数
    sscanf(tmp,"%d",&times);
    if(times > 0)
    {
        Ping pingTemp;
        strcpy(pingTemp.in, temp);
        if(pingTemp.getIP())//获取IP
        {
            strcat(message,"正在 Ping ");
            strcat(message,pingTemp.host->h_name);
            strcat(message,"[");
            strcat(message,inet_ntoa(*(struct in_addr*)pingTemp.host->h_addr_list[0]));
            strcat(message,"] 具有 32 字节的数据:\n");
            ui->Result->setText(message);
            for(int i=0; i<times; i++)//分四个线程去ping
            {
                PingThread * ping = new PingThread(temp);//创建一个新的线程
                connect(ping,SIGNAL(isDone(Ping)),this,SLOT(Ping1Result(Ping)));//qt信号槽机制
                ping->start();
            }
        }
        else
        {
            strcat(message,"Ping 请求找不到主机 ");
            strcat(message,temp);
            strcat(message,"。请检查该名称，然后重试。\n");
            ui->Result->setText(message);
            PingThread * ping = new PingThread(temp);
            connect(ping,SIGNAL(isDone(Ping)),this,SLOT(Ping1Result(Ping)));
            ping->start();
        }
    }
    else
    {
        PingThread * ping = new PingThread(temp);
        connect(ping,SIGNAL(isDone(Ping)),this,SLOT(Ping1Result(Ping)));
        ping->start();
    }
}
```

Ping子网查询信号槽

```
void MainWindow::on_PingButton2_clicked()
{
    ui->PingButton1->setDisabled(true);
    ui->PingButton2->setDisabled(true);
    strcat(message,"\n");
    char *temp=(char *)malloc(100);
    strcpy(temp,ui->Addr_3->text().toLatin1().data());
    int i=0;
```

```
    int flag=0;
    while(temp[i]!='\0')
    {
        if(temp[i] == '/')//判断子网掩码部分是否存在
        {
            flag=1;
            break;
        }
        i++;
    }
    if(flag == 0)
    {
        strcat(message,"子网输入格式错误!\n(示例：112.80.248.75/28)\n");
        ui->Result->setText(message);
        ui->Result->moveCursor(QTextCursor::End,QTextCursor::MoveAnchor); //移动光标到末尾
    }
    else
    {
        int j=0;
        temp[i]='\0';
        i++;
        char t[3];
        while(temp[i]!='\0')//子网掩码
        {
            t[j]=temp[i];
            j++;
            i++;
        }
        t[j]='\0';
        //temp现在是一个Ip地址格式
        //t现在是一个数字
        u_int range = 0;
        sscanf(t,"%u",&range);
        range= 32-range;//主机号位数
        //子网掩码
        u_int subnet_mask=pow(2,32)-pow(2,range);//子网掩码
        u_int ip[4];
        u_int IP=0;
        i=0;
        j=0;
        int k=0;
        while(temp[j]!='\0')
        {
            if(temp[j] == '.')
            {
                temp[j] = '\0';
                sscanf(temp,"%u",&ip[k]);
                i=j+1;
                temp=&temp[i];
                i=0;
                j=0;
                k++;
```

```
                continue;
            }
            j++;
        }
        temp[j] = '\0';
        sscanf(temp,"%u",&ip[k]);
        i=j+1;
        k++;
        IP = ip[0]*256*256*256+ip[1]*256*256+ip[2]*256+ip[3];
        IP= subnet_mask & IP;//子网掩码相与
        u_int left = IP+1;
        int n=0;
        n=pow(2,range)-1;
        u_int right = (IP | n) -1;
        strcpy(message, "子网内主机的IP范围为: ");
        strcat(message,IPToString(left));
        strcat(message, "--");
        strcat(message,IPToString(right));
        strcat(message,"\n");
        while(left <= right)
        {
            char _addr[30];
            strcpy(_addr,IPToString(left));
            numSend=0;
            numReceive=0;
            times=1;

            if(times > 0)
            {
                Ping pingTemp;
                strcpy(pingTemp.in, _addr);
                if(pingTemp.getIP())
                {
                    /*
                    strcat(message,"正在 Ping ");
                    strcat(message,pingTemp.host->h_name);
                    strcat(message,"[");
                    strcat(message,inet_ntoa(*(struct in_addr*)pingTemp.host-
>h_addr_list[0]));
                    strcat(message,"] 具有 32 字节的数据:\n");
                    */
                    ui->Result->setText(message);
                    PingThread * ping = new PingThread(_addr);
                    connect(ping,SIGNAL(isDone(Ping)),this,SLOT(Ping1Result(Ping)));
                    ping->start();
                }
                else
                {
                    strcat(message,"Ping 请求找不到主机 ");
                    strcat(message,_addr);
                    strcat(message,"。请检查该名称，然后重试。\n");
                    ui->Result->setText(message);
                }
```

```
            }
            left++;
        }
    }
}
```

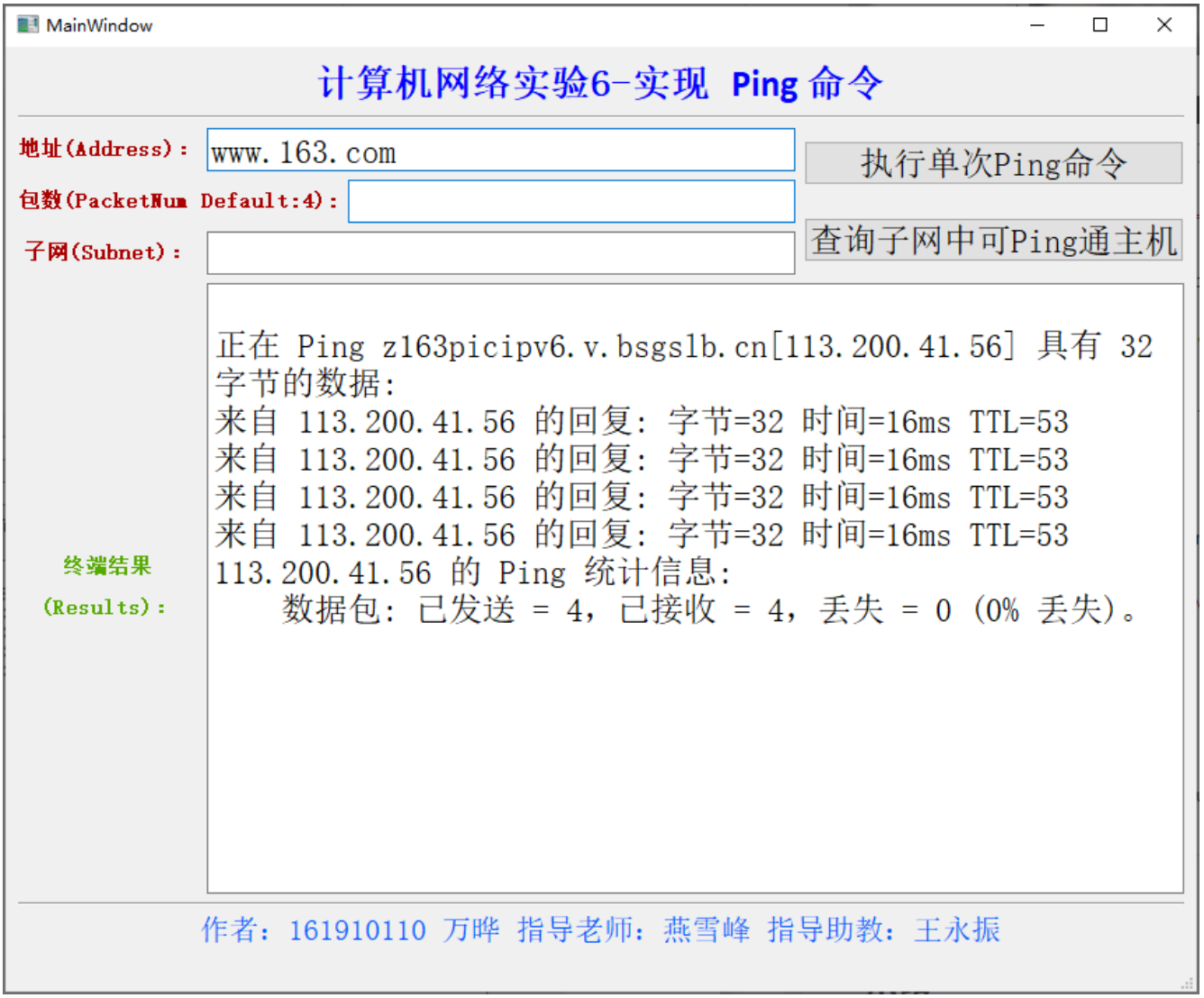
结果框

```
void MainWindow::Ping1Result(Ping ping)
{
    ui->PingButton1->setDisabled(false);
    ui->PingButton2->setDisabled(false);
    if(times >1 && numSend == times )
    {
        strcat(message,inet_ntoa(*(struct in_addr*)ping.host->h_addr_list[0]));
        strcat(message," 的 Ping 统计信息:\n    数据包: 已发送 = ");
        char temp[10];
        sprintf(temp,"%d",numSend);
        strcat(message,temp);
        strcat(message,", 已接收 = ");
        temp[0]='\0';
        sprintf(temp,"%d",numReceive);
        strcat(message,temp);
        strcat(message,", 丢失 = ");
        temp[0]='\0';
        sprintf(temp,"%d",numSend-numReceive);
        strcat(message,temp);
        strcat(message," (");
        temp[0]='\0';
        sprintf(temp,"%d",100*(numSend-numReceive)/numSend);
        strcat(message,temp);
        strcat(message,"% 丢失)。\n");
        strcat(message,"\n");
        numSend=0;
        numReceive=0;
    }
    ui->Result->setText(message);
    ui->Result->moveCursor(QTextCursor::End,QTextCursor::MoveAnchor); //移动光标到末尾
}
```

四、实验结果

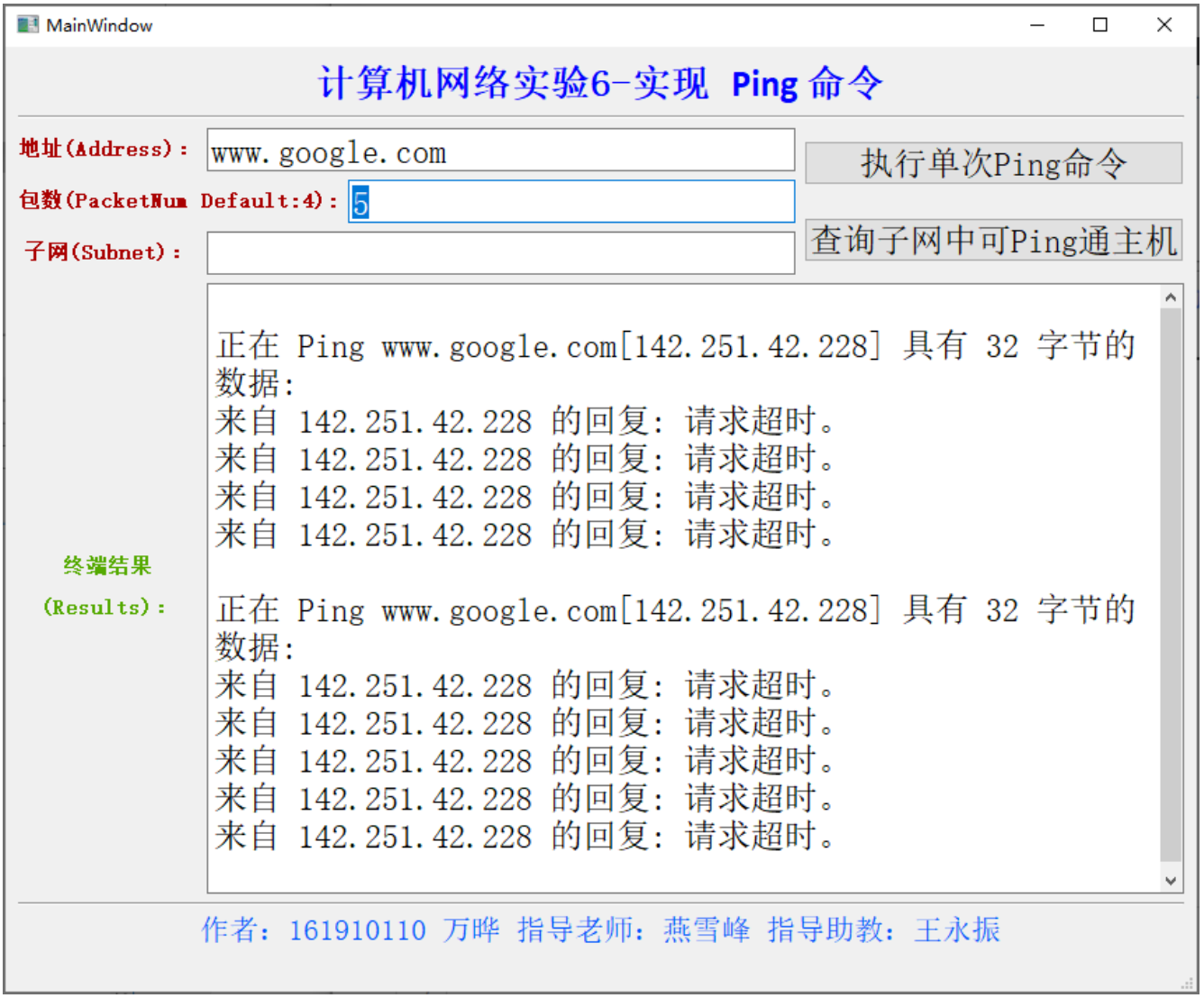
（1）执行单词ping命令

① 在地址栏输入www.163.com的默认状态结果



② 在地址栏填写www.163.com 在包数栏填写7 的结果

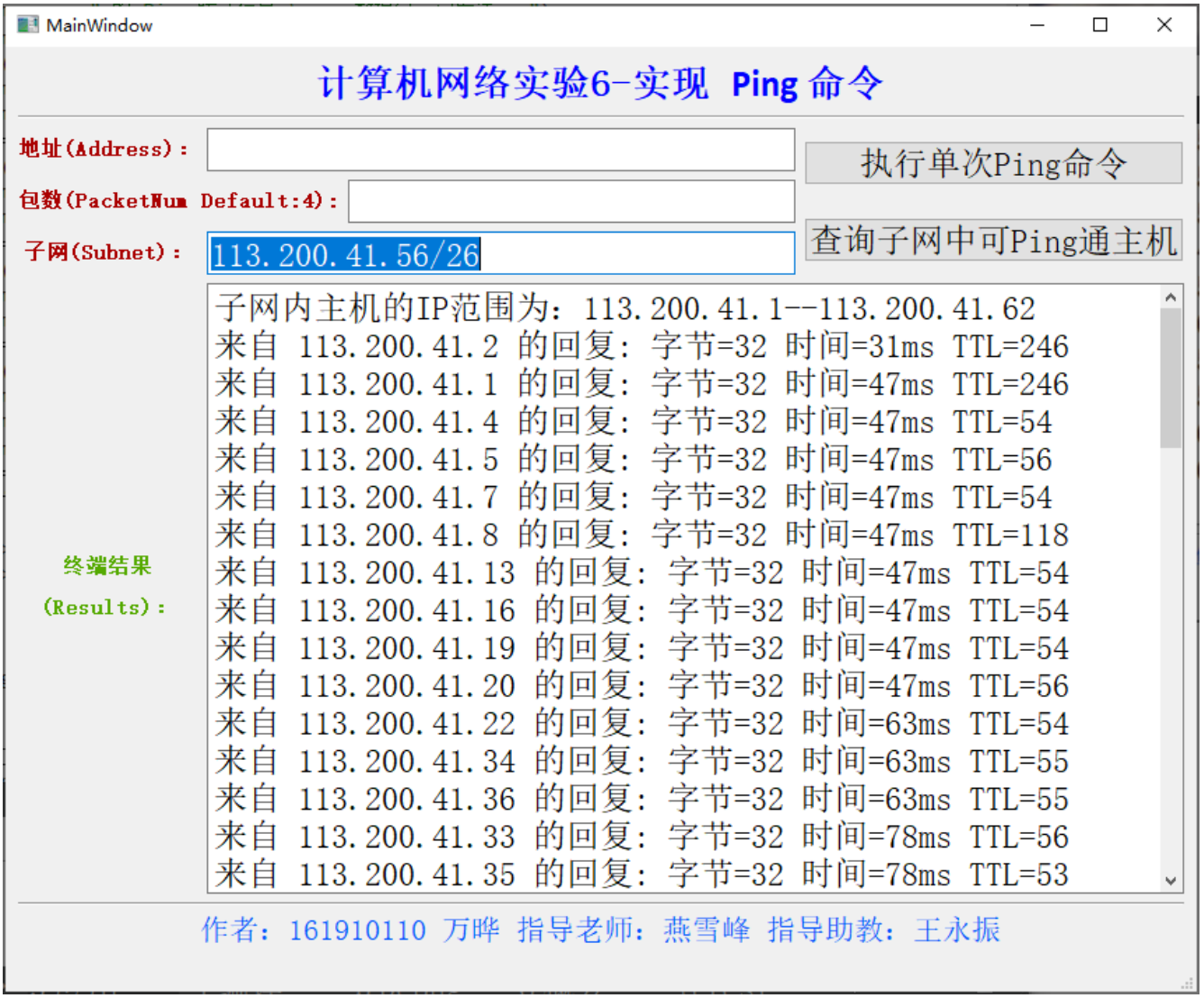
③ 在地址栏输入 www.google.com 出现异常情况



（2）查询子网中可ping 通主机

① 在子网地址栏输入 113.200.41.56/28

② 在地址栏输入 192.168.1.105/26



五、实验小结

经过一个多月的努力，终于把计算机网络实验–ping命令的实现完成了。由于受疫情的影响，没有能在实验室上机完成，还是有一点可惜。这次实验接触到了自己不熟悉的网络SOCKET编程，可以说一切都是从头开始学。由于要用到IP协议和ICMP协议的知识，我首先把课本好好研读了一下，如果对原理都不熟悉那什么也做不了。在熟悉了底层原理之后，我开始根据助教提供的实验指导书开始写代码。由于最后是要图形界面，所以我把开发工作大概分为了两个部分，首先先不考虑界面，实现实验要求的功能，等基本功能实现之后再对界面设计与开发。在实验功能实现的过程中，其实遇到了很多的困难，尤其是在实现查看子网内有哪些主机可以ping通的功能时，尽管知道实现的基本思路，但是就是不知道怎么敲代码。主要还是因为自己在字符串处理方面还不是很熟练，遇到不会的处理方式每次都要上网查使用说明。但好在，最后都实现了。感觉自己的代码能力也获得了较大的提高。总的来说，通过这次实验，我对IP协议以及ICMP协议有了一个更深的了解，还是收获满满的。

六、源代码

6.1 Headers 头文件

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

```

```
#include <QMainWindow>
#include <QThread>
#include "Ping.h"
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT
    public:
        MainWindow(QWidget *parent = nullptr);
        ~MainWindow();

    private:
        Ui::MainWindow *ui;
    public slots:
        void on_PingButton1_clicked();
        void Ping1Result(Ping);
        void on_PingButton2_clicked();
};

class PingThread : public QThread
{
    Q_OBJECT
    public:
        PingThread(char addr[100]);
    signals:
        void isDone(Ping);  //处理完成信号
    protected:
        char addr[100];
        void run();//通过start()间接调用

};

#endif // MAINWINDOW_H

```

ping.h

```
#ifndef PING_H
#define PING_H
#include <winsock.h>
#include <stdio.h>



class IPHeader
{
    public:
        //u_char 占1个字节
        //u_short 占两个字节

```

```
        //u_char version;         // 版本
        //u_char head_len;        // 首部长度
        u_char ver_headlen;       // 版本+首部长度
        u_char service;           // 服务类型
        u_short total_len;        // 总长度
        u_short id;               // 标识符
        u_short flag;             // 标记+片偏移
        u_char ttl;               // 存活时间
        u_char protocol;          // 协议
        u_short check_sum;        // 首部校验和
        u_int src_IP;             // 源IP地址
        u_int dst_IP;             // 目的IP地址
};

class ICMPHeader
{
    public:
        u_char type;              // 类型
        u_char code;              // 代码
        u_short check_sum;        // 校验和
        u_short id;               // 标示符 标识本进程
        u_short seq;              // 序列号
};




class Ping
{
    public:
        char in[100];                                          //从输入框读入
        sockaddr_in dst_IP;                                    //目标IP
        struct hostent *host;
        IPHeader iPHeader;
        ICMPHeader iCMPHeader;
        SOCKET sock;
        char message[5000];
        bool received;
        bool getIP();                                          // 得到ping的IP地址
        bool send();                                           // 发送ICMP报文请求
        bool receive();                                        // 接收ICMP报文,解析并
回显
        u_short check_sum(u_short* buffer, int len);           // 计算校验和

};




//ICMP时间戳请求报文
struct EchoRequest
{
    ICMPHeader icmp_header;                    //ICMP头部
```

```
    unsigned long long time;                    //记录ping时间
};

#pragma pack(push) //保存对齐状态
#pragma pack(1)//设定为1字节对齐
struct EchoResponse
{
    IPHeader ip_header;
    EchoRequest echo_request;
};
#pragma pack(pop)//恢复对齐状态

#endif // PING_H

```

6.2 Sources 源文件

main.cpp

```
#include "mainwindow.h"
#include <QApplication>
#include <winsock.h>

int main(int argc, char *argv[])
{
    WSADATA wsa;
    WSAStartup(MAKEWORD(2, 2), &wsa); //初始化Windows Socket

    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <Windows.h>
#include <QMessageBox>
#include "Ping.h"
#include <thread>
#include <string.h>
#include <math.h>

char message[50000];
char IP[30];
int numSend=0;
int numReceive=0;
int times=4;

```

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{

    qRegisterMetaType<Ping>("Ping");
    ui->setupUi(this);

}

MainWindow::~MainWindow()
{
    delete ui;
    WSACleanup();
}
void MainWindow::on_PingButton1_clicked()
{
    ui->PingButton1->setDisabled(true);
    ui->PingButton2->setDisabled(true);
    numSend=0;
    numReceive=0;
    times=4;
    strcat(message,"\n");
    char temp[100];
    strcpy(temp,ui->Addr->text().toLatin1().data());

    char tmp[10];

    strcpy(tmp,ui->Addr_2->text().toLatin1().data());
    sscanf(tmp,"%d",&times);
    if(times > 0)
    {
        Ping pingTemp;
        strcpy(pingTemp.in, temp);
        if(pingTemp.getIP())
        {
            strcat(message,"正在 Ping ");
            strcat(message,pingTemp.host->h_name);
            strcat(message,"[");
            strcat(message,inet_ntoa(*(struct in_addr*)pingTemp.host-
>h_addr_list[0]));
            strcat(message,"] 具有 32 字节的数据:\n");
            ui->Result->setText(message);
            for(int i=0; i<times; i++)
            {
                PingThread * ping = new PingThread(temp);
                connect(ping,SIGNAL(isDone(Ping)),this,SLOT(Ping1Result(Ping)));
                ping->start();
            }
        }
        else
        {
            strcat(message,"Ping 请求找不到主机 ");
```

```
            strcat(message,temp);
            strcat(message,"。请检查该名称，然后重试。\n");
            ui->Result->setText(message);
            PingThread * ping = new PingThread(temp);
            connect(ping,SIGNAL(isDone(Ping)),this,SLOT(Ping1Result(Ping)));
            ping->start();
        }
    }
    else
    {
        PingThread * ping = new PingThread(temp);
        connect(ping,SIGNAL(isDone(Ping)),this,SLOT(Ping1Result(Ping)));
        ping->start();
    }
}

void PingThread::run()
{
    Ping ping;
    strcpy(ping.in, addr);
    ping.sock=socket(AF_INET,SOCK_RAW,IPPROTO_ICMP);
    if(times >0)
    {
        if(ping.send())
        {
            numSend++;
            printf("%d\n",numSend);
        }
        if(ping.received)
        {
            numReceive++;
        }
        strcat(message,ping.message);
    }
    emit isDone(ping);
}
PingThread::PingThread(char _addr[100])
{
    strcpy(addr, _addr);
}

void MainWindow::Ping1Result(Ping ping)
{
    ui->PingButton1->setDisabled(false);
    ui->PingButton2->setDisabled(false);
    if(times >1 && numSend == times )
    {
        strcat(message,inet_ntoa(*(struct in_addr*)ping.host->h_addr_list[0]));
        strcat(message," 的 Ping 统计信息:\n    数据包: 已发送 = ");
        char temp[10];
        sprintf(temp,"%d",numSend);
        strcat(message,temp);
        strcat(message,", 已接收 = ");
```

```
        temp[0]='\0';
        sprintf(temp,"%d",numReceive);
        strcat(message,temp);
        strcat(message,", 丢失 = ");
        temp[0]='\0';
        sprintf(temp,"%d",numSend-numReceive);
        strcat(message,temp);
        strcat(message," (");
        temp[0]='\0';
        sprintf(temp,"%d",100*(numSend-numReceive)/numSend);
        strcat(message,temp);
        strcat(message,"% 丢失)。\n");
        strcat(message,"\n");
        numSend=0;
        numReceive=0;
    }
    ui->Result->setText(message);
    ui->Result->moveCursor(QTextCursor::End,QTextCursor::MoveAnchor); //移动光标到末尾
}


char *IPToString(u_int IP_int)
{
    IP[0]='\0';
    char temp[5];
    sprintf(temp,"%u",(IP_int >> 24) & 0x000000FF);
    strcat(IP,temp);
    strcat(IP,".");
    temp[0]='\0';
    sprintf(temp,"%u",(IP_int >> 16) & 0x000000FF);
    strcat(IP,temp);
    strcat(IP,".");
    temp[0]='\0';
    sprintf(temp,"%u",(IP_int >>8) & 0x000000FF);
    strcat(IP,temp);
    strcat(IP,".");
    temp[0]='\0';
    sprintf(temp,"%u",IP_int & 0x000000FF);
    strcat(IP,temp);
    temp[0]='\0';
    return IP;
}


void MainWindow::on_PingButton2_clicked()
{
    ui->PingButton1->setDisabled(true);
    ui->PingButton2->setDisabled(true);
    strcat(message,"\n");
    char *temp=(char *)malloc(100);
    strcpy(temp,ui->Addr_3->text().toLatin1().data());
    int i=0;
```

```
    int flag=0;
    while(temp[i]!='\0')
    {
        if(temp[i] == '/')
        {
            flag=1;
            break;
        }
        i++;
    }
    if(flag == 0)
    {
        strcat(message,"子网输入格式错误!\n(示例：192.168.253.16/28)\n");
        ui->Result->setText(message);
        ui->Result->moveCursor(QTextCursor::End,QTextCursor::MoveAnchor); //移动光标到末尾
    }
    else
    {
        int j=0;
        temp[i]='\0';
        i++;
        char t[3];
        while(temp[i]!='\0')
        {
            t[j]=temp[i];
            j++;
            i++;
        }
        t[j]='\0';
        //temp现在是一个Ip地址格式
        //t现在是一个数字
        u_int range = 0;
        sscanf(t,"%u",&range);
        range= 32-range;
        //子网掩码
        u_int subnet_mask=pow(2,32)-pow(2,range);
        u_int ip[4];
        u_int IP=0;
        i=0;
        j=0;
        int k=0;
        while(temp[j]!='\0')
        {
            if(temp[j] == '.')
            {
                temp[j] = '\0';
                sscanf(temp,"%u",&ip[k]);
                i=j+1;
                temp=&temp[i];
                i=0;
                j=0;
                k++;
```

```
                continue;
            }
            j++;
        }
        temp[j] = '\0';
        sscanf(temp,"%u",&ip[k]);
        i=j+1;
        k++;
        IP = ip[0]*256*256*256+ip[1]*256*256+ip[2]*256+ip[3];
        IP= subnet_mask & IP;
        u_int left = IP+1;
        int n=0;
        n=pow(2,range)-1;
        u_int right = (IP | n) -1;
        strcpy(message, "子网内主机的IP范围为: ");
        strcat(message,IPToString(left));
        strcat(message, "--");
        strcat(message,IPToString(right));
        strcat(message,"\n");
        while(left <= right)
        {
            char _addr[30];
            strcpy(_addr,IPToString(left));
            numSend=0;
            numReceive=0;
            times=1;

            if(times > 0)
            {
                Ping pingTemp;
                strcpy(pingTemp.in, _addr);
                if(pingTemp.getIP())
                {
                    /*
                    strcat(message,"正在 Ping ");
                    strcat(message,pingTemp.host->h_name);
                    strcat(message,"[");
                    strcat(message,inet_ntoa(*(struct in_addr*)pingTemp.host-
>h_addr_list[0]));
                    strcat(message,"] 具有 32 字节的数据:\n");
                    */
                    ui->Result->setText(message);
                    PingThread * ping = new PingThread(_addr);
                    connect(ping,SIGNAL(isDone(Ping)),this,SLOT(Ping1Result(Ping)));
                    ping->start();
                }
                else
                {
                    strcat(message,"Ping 请求找不到主机 ");
                    strcat(message,_addr);
                    strcat(message,"。请检查该名称，然后重试。\n");
                    ui->Result->setText(message);
                }
```

```
            }

            left++;
        }
    }
}


```

ping.cpp

```cpp
#include "ping.h"
#include <Windows.h>
#include <string.h>

bool Ping::getIP() //获取IP地址
{
    host = gethostbyname(in);
    if (host == NULL)
    {
        return false;
    }
    dst_IP.sin_family = AF_INET; //IPv4
    dst_IP.sin_addr.S_un.S_addr = *(u_long*)host->h_addr;
    puts(host->h_name);
    puts(inet_ntoa(*(struct in_addr*)host->h_addr_list[0]));  //inet_ntoa()将网络地址转换成“.”点隔的字符串格式
    return true;
}

bool Ping::send() //发送数据包
{
    if(getIP()==false) //获取IP失败
    {
        return false;
    }
    static int id = 1;
    static int seq = 1;
    EchoRequest req;
    req.time = GetTickCount(); //从0开始计时，返回自设备启动后的毫秒数
    req.icmp_header.type = 8;  //ICMP_ECHO
    req.icmp_header.code = 0;
    id = ::GetCurrentProcessId();//获取当前的进程ID
    req.icmp_header.id = id;
    //id++;
    req.icmp_header.seq = seq++;//序号加一
    req.icmp_header.check_sum = check_sum((u_short*)&req, sizeof(EchoRequest));//校验和

    int re = sendto(sock, (char*)&req, sizeof(req), 0, (sockaddr*)&dst_IP, sizeof(dst_IP));
    //将数据由指定的socket 传给目标主机

```

```
    //成功则返回实际传送出去的字符数，失败返回 -1，
    // message 为此次ping的一些信息，输出到
    if(re == SOCKET_ERROR)  //SOCKET_ERROR = -1
    {
        strcat(message,"发送错误，错误码:");
        char temp[10];
        sprintf(temp,"%d",WSAGetLastError()); //WSAGetLastError()返回该线程上一次Sockets API函数调用时的错误代码,即sendto()函数的错误调用
        strcat(message,temp);
        strcat(message,"\n");
        return false;
    }
    if(receive())
    {
        received=true;
    }
    else
    {
        received=false;
    }
    return true;
}
u_short Ping::check_sum(u_short* buf, int len) //校验和计算
{
    unsigned int sum=0;
    unsigned short *cbuf;

    cbuf=(unsigned short *)buf;

    while(len>1)
    {
        sum+=*cbuf++;
        len-=2;
    }
    if(len)
    {
        sum+=*(unsigned char *)cbuf;
    }
    sum=(sum>>16)+(sum & 0xffff);
    sum+=(sum>>16);
    return ~sum;
}
bool Ping::receive() //接收并分析返回的数据包
{
    int timeout = 1000;//设置超时的时间
    if(setsockopt(sock,SOL_SOCKET,SO_RCVTIMEO,(char *)&timeout,sizeof(timeout)) == SOCKET_ERROR) //设置套接口
    {
        strcat(message,"接收设置错误，错误码:");//设置套接口返回错误
        char temp[10];
        sprintf(temp,"%d",WSAGetLastError());
        strcat(message,temp);
        strcat(message,"\n");
```

```
        return false;
    }
    char temp[100];
    strcat(message,"来自 ");
    strcat(message,inet_ntoa(*(struct in_addr*)host->h_addr_list[0]));//地址
    strcat(message," 的回复: ");
    EchoResponse* res=new EchoResponse;
    int size = sizeof(sockaddr);
    int re = recvfrom(sock, (char*)res, sizeof(EchoResponse), 0, (sockaddr*)&dst_IP, &size);//接收到的返回的套接字
    if (re == SOCKET_ERROR)//出错
    {
        int code = WSAGetLastError();
        if(code==10060)
        {
            strcat(message,"请求超时。\n");
        }
        return false;
    }
    unsigned long long time = GetTickCount() - res->echo_request.time;//两个相减即为TTL时间
    int type = res->echo_request.icmp_header.type;
    int code = res->echo_request.icmp_header.code;
    char TTL[10]={'\0'};
    sprintf(TTL,"%d",(int)res->ip_header.ttl);
    delete res;
    if(type==0&&code==0)//输出到message信息中
    {
        strcat(message,"字节=32 时间=");
        sprintf(temp,"%I64u",time);
        strcat(message,temp);
        strcat(message,"ms TTL=");
        strcat(message,TTL);
        strcat(message,"\n");
        return true;
    }
    else
    {
        strcat(message,"请求超时");
        strcat(message,"\n");
        return false;
    }

}

```

6.3 Forms UI文件

mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
```

```
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>852</width>
    <height>676</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <layout class="QGridLayout" name="gridLayout">
    <item row="0" column="0" colspan="4">
     <widget class="QLabel" name="label">
      <property name="text">
       <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p
align=&quot;center&quot;&gt;&lt;span style=&quot; font-size:20pt; font-weight:600;
color:#0000ff;&quot;&gt;计算机网络实验6-&lt;/span&gt;&lt;span style=&quot; font-
family:'宋体'; font-size:20pt; font-weight:600; color:#0000ff;&quot;&gt;实现
&lt;/span&gt;&lt;span style=&quot; font-family:'Calibri,12'; font-size:20pt; font-
weight:600; color:#0000ff;&quot;&gt;Ping &lt;/span&gt;&lt;span style=&quot; font-
family:'宋体'; font-size:20pt; font-weight:600; color:#0000ff;&quot;&gt;命令
&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
      </property>
     </widget>
    </item>
    <item row="1" column="0" colspan="4">
     <widget class="Line" name="line_2">
      <property name="orientation">
       <enum>Qt::Horizontal</enum>
      </property>
     </widget>
    </item>
    <item row="2" column="0">
     <widget class="QLabel" name="label_3">
      <property name="text">
       <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p
align=&quot;center&quot;&gt;&lt;span style=&quot; font-size:11pt; font-weight:600;
color:#aa0000;&quot;&gt;地址(Address):
&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
      </property>
     </widget>
    </item>
    <item row="2" column="1" colspan="2">
     <widget class="QLineEdit" name="Addr">
      <property name="cursor">
       <cursorShape>IBeamCursor</cursorShape>
      </property>
      <property name="toolTip">
```

```
       <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;输入地址 如：www.baidu.com 后点击“执行单次ping命令”按钮&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
      </property>
      <property name="placeholderText">
       <string/>
      </property>
     </widget>
    </item>
    <item row="2" column="3" rowspan="2">
     <widget class="QPushButton" name="PingButton1">
      <property name="cursor">
       <cursorShape>PointingHandCursor</cursorShape>
      </property>
      <property name="text">
       <string>执行单次Ping命令</string>
      </property>
     </widget>
    </item>
    <item row="3" column="0" rowspan="2" colspan="2">
     <widget class="QLabel" name="label_5">
      <property name="text">
       <string>&lt;!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.0//EN&quot; &quot;http://www.w3.org/TR/REC-html40/strict.dtd&quot;&gt;
&lt;html&gt;&lt;head&gt;&lt;meta name=&quot;qrichtext&quot; content=&quot;1&quot; /&gt;&lt;style type=&quot;text/css&quot;&gt;
p, li { white-space: pre-wrap; }
&lt;/style&gt;&lt;/head&gt;&lt;body style=&quot; font-family:'SimSun'; font-size:9pt; font-weight:400; font-style:normal;&quot;&gt;
&lt;p style=&quot; margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;&quot;&gt;&lt;span style=&quot; font-size:11pt; font-weight:600; color:#aa0000;&quot;&gt;包数(PacketNum Default:4):&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
      </property>
     </widget>
    </item>
    <item row="3" column="2" rowspan="2">
     <widget class="QLineEdit" name="Addr_2">
      <property name="toolTip">
       <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;输入包数 如：5&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
      </property>
      <property name="placeholderText">
       <string/>
      </property>
     </widget>
    </item>
    <item row="4" column="3" rowspan="2">
     <widget class="QPushButton" name="PingButton2">
      <property name="cursor">
       <cursorShape>PointingHandCursor</cursorShape>
      </property>
      <property name="whatsThis">
```

```
       <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;按钮
&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
      </property>
      <property name="text">
       <string>查询子网中可Ping通主机</string>
      </property>
     </widget>
    </item>
    <item row="5" column="0">
     <widget class="QLabel" name="label_6">
      <property name="text">
       <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p
align=&quot;center&quot;&gt;&lt;span style=&quot; font-size:11pt; font-weight:600;
color:#aa0000;&quot;&gt;子网(Subnet)：
&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
      </property>
     </widget>
    </item>
    <item row="5" column="1" colspan="2">
     <widget class="QLineEdit" name="Addr_3">
      <property name="toolTip">
       <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;请输入子网及其子网掩码
如：112.80.248.75/28&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
      </property>
      <property name="whatsThis">
       <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;阿斯顿
&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
      </property>
      <property name="placeholderText">
       <string/>
      </property>
     </widget>
    </item>
    <item row="6" column="0">
     <widget class="QLabel" name="label_4">
      <property name="text">
       <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p
align=&quot;center&quot;&gt;&lt;span style=&quot; font-size:11pt; font-weight:600;
color:#55aa00;&quot;&gt;终端结果&lt;/span&gt;&lt;/p&gt;&lt;p
align=&quot;center&quot;&gt;&lt;span style=&quot; font-size:11pt; font-weight:600;
color:#55aa00;&quot;&gt;(Results)： &lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;
</string>
      </property>
     </widget>
    </item>
    <item row="6" column="1" colspan="3">
     <widget class="QTextEdit" name="Result">
      <property name="enabled">
       <bool>true</bool>
      </property>
      <property name="maximumSize">
       <size>
        <width>16777215</width>
```

```
        <height>800</height>
       </size>
      </property>
      <property name="cursor" stdset="0">
       <cursorShape>ArrowCursor</cursorShape>
      </property>
     </widget>
    </item>
    <item row="7" column="0" colspan="4">
     <widget class="Line" name="line">
      <property name="orientation">
       <enum>Qt::Horizontal</enum>
      </property>
     </widget>
    </item>
    <item row="8" column="0" colspan="4">
     <widget class="QLabel" name="label_2">
      <property name="text">
       <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p
align=&quot;center&quot;&gt;&lt;span style=&quot; font-size:16pt;
color:#0055ff;&quot;&gt;作者：161910110 万晔 指导老师：燕雪峰 指导助教：王永振
&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
      </property>
     </widget>
    </item>
   </layout>
  </widget>
  <widget class="QStatusBar" name="statusbar"/>
  <widget class="QMenuBar" name="menubar">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>852</width>
     <height>21</height>
    </rect>
   </property>
  </widget>
 </widget>
 <resources/>
 <connections/>
</ui>
```