

# 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

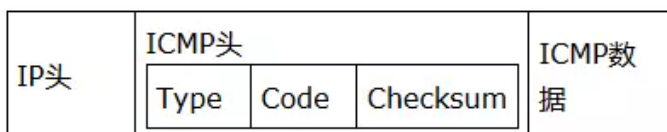
我们知道，ping命令是基于ICMP协议来实现的。那么我们再来看下图，就明白了ICMP协议又是通过IP协议来发送的，即ICMP报文是封装在IP包中。



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	校验和 CHECKSUM
标志符 ID		顺序号 SEQ

代码结构为：

```
1 class ICMPHeader
2 {
3     public:
4         u_char type;           // 类型
5         u_char code;          // 代码
6         u_short check_sum;     // 校验和
7         u_short id;            // 标示符 标识本进程
8         u_short seq;           // 序列号
9     };
```

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)。

```

1 //回显请求 ICMP_ECHO
2 struct EchoRequest
3 {
4     ICMPHeader icmp_header;           //ICMP头部
5     unsigned long long time;         //记录ping时间
6 };
7
8 //回显应答 ICMP_ECHOREPLY
9 struct EchoResponse
10 {
11     IPHeader ip_header;
12     EchoRequest echo_request;
13 };

```

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

```

1 class IPHeader
2 {
3     public:
4         //u_char 占1个字节
5         //u_short 占两个字节
6         //u_char version;           // 版本
7         //u_char head_len;          // 首部长度
8         u_char ver_headlen;         // 版本+首部长度
9         u_char service;             // 服务类型
10        u_short total_len;           // 总长度
11        u_short id;                  // 标识符

```

```

12     u_short flag;           // 标记+片偏移
13     u_char ttl;            // 存活时间
14     u_char protocol;       // 协议
15     u_short check_sum;      // 首部校验和
16     u_int src_IP;           // 源IP地址
17     u_int dst_IP;           // 目的IP地址
18 };

```

各字段说明：

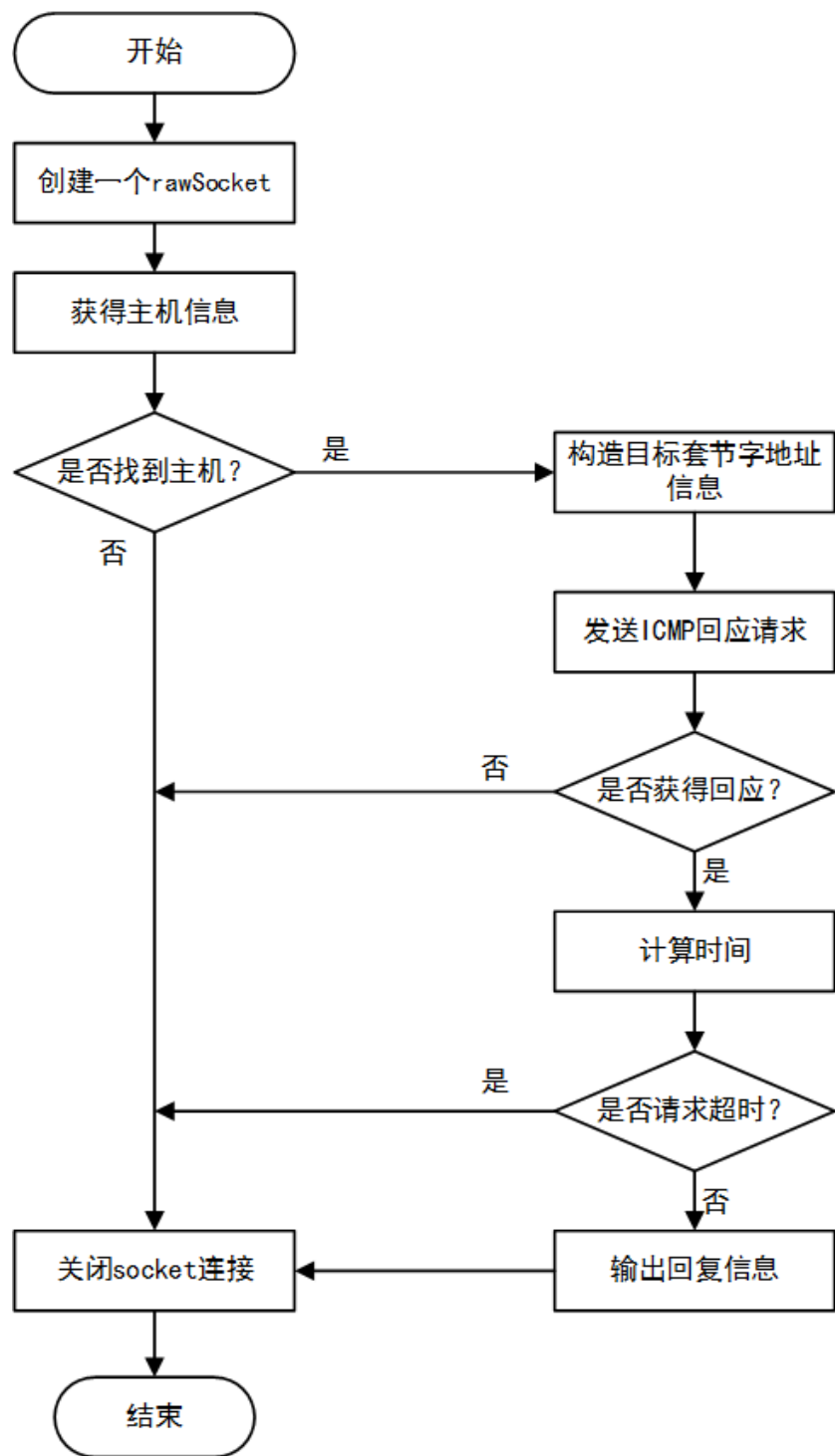
- 版本：占 4 位，表示 IP 协议的版本。通信双方使用的 IP 协议版本必须一致。目前广泛使用的 IP 协议版本号为 4，即 IPv4。
- 首部长度：最常用的首部长度就是 20 字节（即首部长度为 0101）。
- 区分服务：也被称为服务类型，占 8 位，用来获得更好的服务。
- 总长度：首部和数据之和，单位为字节。总长度字段为 16 位，因此数据报的最大长度为  $2^{16}-1=65535$  字节。
- 标识符：用来标识数据报，占 16 位。具有相同的标识字段值的分片报文会被重组成原来的数据报。
- 标志：占 3 位。第一位未使用，其值为 0。第二位称为 DF（不分片），表示是否允许分片。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

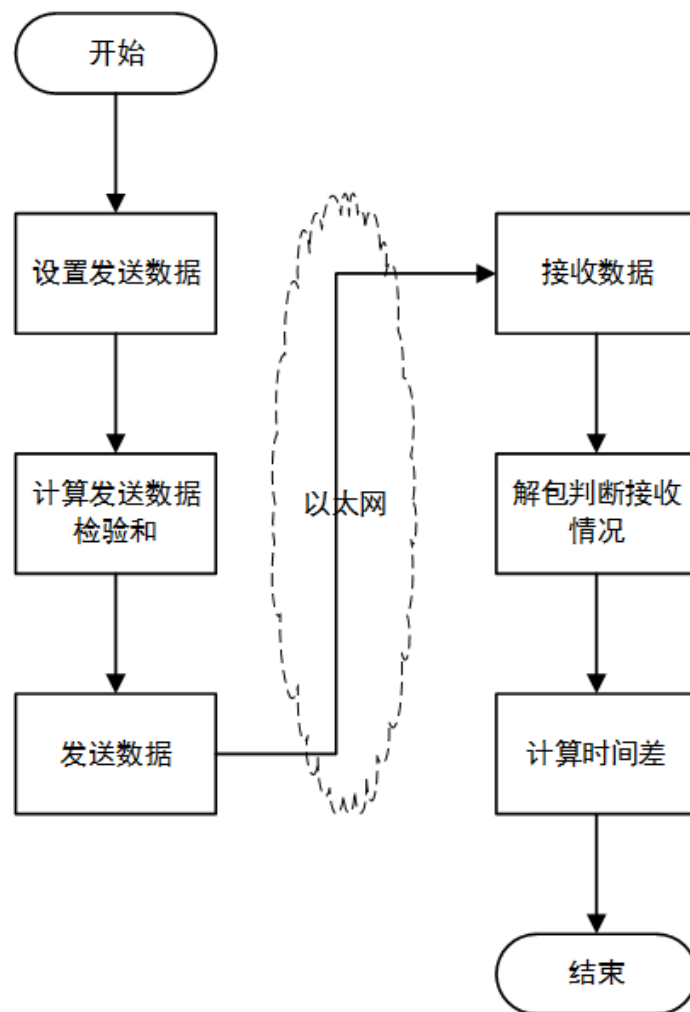
- 可以通过程序模拟对输入的目的地址进行 Ping 命令，在一秒钟内，如果收到，则为成功，如果收不到，则失败，打印输出该过程的信息，显示在图形化界面上
- 可以通过程序，查看子网中有哪些主机可以 Ping 通，打印输出该过程的信息，显示在图形化界面上

## 2.3 实现思路

实验整体框架示意图



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

```

1  class IPHeader
2  {
3      public:
4          //u_char 占1个字节
5          //u_short 占两个字节
6          //u_char version;           // 版本
7          //u_char head_len;          // 首部长
8          u_char ver_headlen;          // 版本+首部长
9          u_char service;              // 服务类型
10         u_short total_len;            // 总长度
11         u_short id;                  // 标识符
12         u_short flag;                // 标记+片偏移
13         u_char ttl;                  // 存活时间
14         u_char protocol;              // 协议
15         u_short check_sum;            // 首部校验和
16         u_int src_IP;                 // 源IP地址
17         u_int dst_IP;                 // 目的IP地址
18 };
  
```

## ICMPHeader //ICMP 头

```
1 class ICMPHeader
2 {
3     public:
4         u_char type;           // 类型
5         u_char code;           // 代码
6         u_short check_sum;      // 校验和
7         u_short id;             // 标示符 标识本进程
8         u_short seq;           // 序列号
9     };
```

## Ping //Ping 主类

```
1 class Ping
2 {
3     public:
4         char in[100];           //从输入框读入
5         sockaddr_in dst_IP;      //目标IP
6         struct hostent *host;    //主机
7         IPHeader ipHeader;       //IP头
8         ICMPHeader icmpHeader;   //ICMP头
9         SOCKET sock;            //套接字
10        char message[5000];       // Ping 内部的一些信息
11        bool received;           // 是否收到
12        bool getIP();            // 得到ping的IP地址
13        bool send();             // 发送ICMP报文请求
14        bool receive();          // 接收ICMP报文,解析并
15        回显
16        u_short check_sum(u_short* buffer, int len); // 计算校验和
17    };
```

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:



```

1  bool Ping::getIP() //获取IP地址
2  {
3      host = gethostbyname(in);    //in 为输入框输入的地址
4      if (host == NULL)
5      {
6          return false;
7      }
8      dst_IP.sin_family = AF_INET; //IPv4格式
9      dst_IP.sin_addr.S_un.S_addr = *(u_long*)host->h_addr;
10     puts(host->h_name);
11     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
12     return true;
13 }

```

**EchoRequest** //ICMP时间戳请求报文

```

1  struct EchoRequest
2  {
3      ICMPHeader icmp_header;           //ICMP头部
4      unsigned long long time;         //记录ping时间
5  };

```

**PingThread** //多线程类

```

1  class PingThread : public QThread
2  {
3      Q_OBJECT
4      public:
5          PingThread(char addr[100]);
6      signals:
7          void isDone(Ping); //处理完成信号
8      protected:
9          char addr[100];
10         void run(); //通过start()间接调用
11
12 };

```

## 3.2 函数

**getIP()** //获取IP地址

```

1 bool Ping::getIP() //获取IP地址
2 {
3     host = gethostbyname(in);    //in 为输入框输入的地址
4     if (host == NULL)
5     {
6         return false;
7     }
8     dst_IP.sin_family = AF_INET; //IPv4格式
9     dst_IP.sin_addr.S_un.S_addr = *(u_long*)host->h_addr;
10    puts(host->h_name);
11    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
12    return true;
13 }

```

**send() //发送数据包**

```

1 bool Ping::send() //发送数据包
2 {
3     if(getIP()==false) //获取IP失败
4     {
5         return false;
6     }
7     static int id = 1;
8     static int seq = 1;
9     EchoRequest req;
10    req.time = GetTickCount(); //从0开始计时，返回自设备启动后的毫秒数
11    req.icmp_header.type = 8; //ICMP_ECHO
12    req.icmp_header.code = 0; //编码
13    id = ::GetCurrentProcessId(); //获取当前的进程ID
14    req.icmp_header.id = id; //id++;
15    req.icmp_header.seq = seq++; //序号加一
16    req.icmp_header.check_sum = check_sum((u_short*)&req, sizeof(EchoRequest)); //校验
    和字段
17
18    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
19    // 成功则返回实际传送出去的字符数，失败返回 -1,
20    // message 为此次ping的一些信息，输出到
21    if(re == SOCKET_ERROR) //SOCKET_ERROR = -1
22    {
23        strcat(message, "发送错误，错误码:");
24        char temp[10];
25        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
26        strcat(message, temp);
27        strcat(message, "\n");
28        return false; //发送失败
29    }
30    if(receive())
31    {
32        received=true; //接收成功
    }

```

```

33     }
34     else
35     {
36         received=false;//接收失败
37     }
38     return true;    //发送成功
39 }

```

### check\_sum() 检验和计算

```

1  u_short Ping::check_sum(u_short* buf, int len) //校验和计算
2  {
3      unsigned int sum=0;
4      unsigned short *cbuf;
5
6      cbuf=(unsigned short *)buf;
7
8      while(len>1)
9      {
10         sum+=*cbuf++;
11         len-=2;
12     }
13     if(len)
14     {
15         sum+=*(unsigned char *)cbuf;
16     }
17     sum=(sum>>16)+(sum & 0xffff);
18     sum+=(sum>>16);
19     return ~sum;
20 }

```

### receive() //接收并分析返回的数据包

```

1  bool Ping::receive() //接收并分析返回的数据包
2  {
3      int timeout = 1000;//设置超时的时间
4      if(setsockopt(sock,SOL_SOCKET,SO_RCVTIMEO,(char *)&timeout,sizeof(timeout)) ==
5      SOCKET_ERROR) //设置套接口
6      {
7          strcat(message,"接收设置错误，错误码:");//设置套接口返回错误
8          char temp[10];
9          sprintf(temp,"%d",WSAGetLastError());
10         strcat(message,temp);
11         strcat(message,"\n");
12         return false;
13     }
14     char temp[100];
15     strcat(message,"来自 ");
16     strcat(message,inet_ntoa(*(struct in_addr*)host->h_addr_list[0]));//地址
17     strcat(message," 的回复: ");
18     EchoResponse* res=new EchoResponse;
19     int size = sizeof(sockaddr);

```

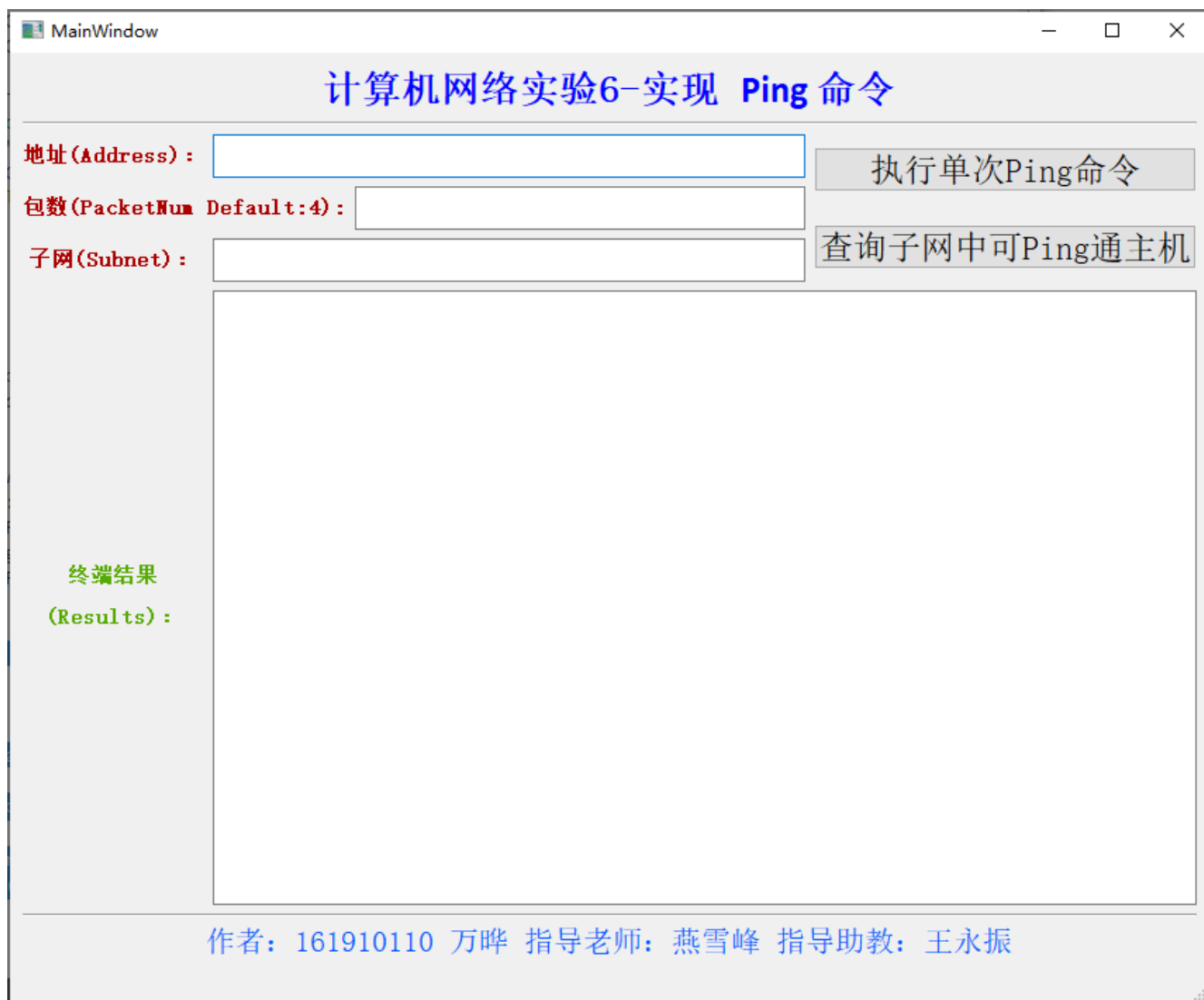
```

19     int re = recvfrom(sock, (char*)res, sizeof(EchoResponse), 0, (sockaddr*)&dst_IP,
//接收到的返回的套接字
20     &size);
21     if (re == SOCKET_ERROR)//出错
22     {
23         int code = WSAGetLastError();
24         if(code==10060)
25         {
26             strcat(message,"请求超时。 \n");
27         }
28         return false;
29     }
30     unsigned long long time = GetTickCount() - res->echo_request.time;//两个相减即为
TTL时间
31     int type = res->echo_request.icmp_header.type;
32     int code = res->echo_request.icmp_header.code;
33     char TTL[10]={'\0'};
34     sprintf(TTL, "%d", (int)res->ip_header.ttl);
35     delete res;
36     if(type==0&&code==0)//输出到message信息中
37     {
38         strcat(message,"字节=32 时间=");
39         sprintf(temp, "%I64u", time);
40         strcat(message, temp);
41         strcat(message, "ms TTL=");
42         strcat(message, TTL);
43         strcat(message, "\n");
44         return true;
45     }
46     else
47     {
48         strcat(message,"请求超时");
49         strcat(message, "\n");
50         return false;
51     }
52 }

```

### 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:

```
1 void MainWindow::on_PingButton1_clicked()
2 {
3     ui->PingButton1->setDisabled(true);
4     ui->PingButton2->setDisabled(true);
5     numSend=0;
6     numReceive=0;
7     times=4;
8     strcat(message, "\n");
9     char temp[100]; //用来存储ping地址
10    strcpy(temp, ui->Addr->text().toLatin1().data()); //ping 地址
11    char tmp[10]; //用来存储ping包数
```

```

12     strcpy(tmp, ui->Addr_2->text().toLatin1().data()); //ping 包数
13     sscanf(tmp, "%d", &times);
14     if(times > 0)
15     {
16         Ping pingTemp;
17         strcpy(pingTemp.in, tmp);
18         if(pingTemp.getIP())//获取IP
19         {
20             strcat(message, "正在 Ping ");
21             strcat(message, pingTemp.host->h_name);
22             strcat(message, "[");
23             strcat(message, inet_ntoa(*(struct in_addr*)pingTemp.host-
>h_addr_list[0]));
24             strcat(message, "] 具有 32 字节的数据:\n");
25             ui->Result->setText(message);
26             for(int i=0; i<times; i++)//分四个线程去ping
27             {
28                 PingThread * ping = new PingThread(tmp);//创建一个新的线程
29                 connect(ping, SIGNAL(isDone(Ping)), this, SLOT(Ping1Result(Ping)));//qt
信号槽机制
30                 ping->start();
31             }
32         }
33         else
34         {
35             strcat(message, "Ping 请求找不到主机 ");
36             strcat(message, tmp);
37             strcat(message, "。请检查该名称，然后重试。 \n");
38             ui->Result->setText(message);
39             PingThread * ping = new PingThread(tmp);
40             connect(ping, SIGNAL(isDone(Ping)), this, SLOT(Ping1Result(Ping)));
41             ping->start();
42         }
43     }
44     else
45     {
46         PingThread * ping = new PingThread(tmp);
47         connect(ping, SIGNAL(isDone(Ping)), this, SLOT(Ping1Result(Ping)));
48         ping->start();
49     }
50 }

```

## Ping子网查询信号槽

```

1 void MainWindow::on_PingButton2_clicked()
2 {
3     ui->PingButton1->setDisabled(true);
4     ui->PingButton2->setDisabled(true);
5     strcat(message, "\n");
6     char *temp=(char *)malloc(100);
7     strcpy(temp, ui->Addr_3->text().toLatin1().data());
8     int i=0;

```

```

9      int flag=0;
10     while(temp[i]!='\0')
11     {
12         if(temp[i] == '/')//判断子网掩码部分是否存在
13         {
14             flag=1;
15             break;
16         }
17         i++;
18     }
19     if(flag == 0)
20     {
21         strcat(message,"子网输入格式错误!\n(示例: 112.80.248.75/28)\n");
22         ui->Result->setText(message);
23         ui->Result->moveCursor(QTextCursor::End,QTextCursor::MoveAnchor); //移动光标
到末尾
24     }
25     else
26     {
27         int j=0;
28         temp[i]='\0';
29         i++;
30         char t[3];
31         while(temp[i]!='\0')//子网掩码
32         {
33             t[j]=temp[i];
34             j++;
35             i++;
36         }
37         t[j]='\0';
38         //temp现在是一个Ip地址格式
39         //t现在是一个数字
40         u_int range = 0;
41         sscanf(t,"%u",&range);
42         range= 32-range;//主机号位数
43         //子网掩码
44         u_int subnet_mask=pow(2,32)-pow(2,range);//子网掩码
45         u_int ip[4];
46         u_int IP=0;
47         i=0;
48         j=0;
49         int k=0;
50         while(temp[j]!='\0')
51         {
52             if(temp[j] == '.')
53             {
54                 temp[j] = '\0';
55                 sscanf(temp,"%u",&ip[k]);
56                 i=j+1;
57                 temp=&temp[i];
58                 i=0;
59                 j=0;
60                 k++;

```

```

61         continue;
62     }
63     j++;
64 }
65 temp[j] = '\0';
66 sscanf(temp,"%u",&ip[k]);
67 i=j+1;
68 k++;
69 IP = ip[0]*256*256*256+ip[1]*256*256+ip[2]*256+ip[3];
70 IP= subnet_mask & IP;//子网掩码相与
71 u_int left = IP+1;
72 int n=0;
73 n=pow(2,range)-1;
74 u_int right = (IP | n) -1;
75 strcpy(message, "子网内主机的IP范围为: ");
76 strcat(message,IPToString(left));
77 strcat(message, "--");
78 strcat(message,IPToString(right));
79 strcat(message,"\n");
80 while(left <= right)
81 {
82     char _addr[30];
83     strcpy(_addr,IPToString(left));
84     numSend=0;
85     numReceive=0;
86     times=1;
87
88     if(times > 0)
89     {
90         Ping pingTemp;
91         strcpy(pingTemp.in, _addr);
92         if(pingTemp.getIP())
93         {
94             /*
95             strcat(message,"正在 Ping ");
96             strcat(message,pingTemp.host->h_name);
97             strcat(message,"[");
98             strcat(message,inet_ntoa(*(struct in_addr*)pingTemp.host-
>h_addr_list[0]));
99             strcat(message,"] 具有 32 字节的数据:\n");
100             */
101             ui->Result->setText(message);
102             PingThread * ping = new PingThread(_addr);
103             connect(ping,SIGNAL(isDone(Ping)),this,SLOT(Ping1Result(Ping)));
104             ping->start();
105         }
106         else
107         {
108             strcat(message,"Ping 请求找不到主机 ");
109             strcat(message,_addr);
110             strcat(message,"。请检查该名称, 然后重试.\n");
111             ui->Result->setText(message);
112         }

```



```

113     }
114     left++;
115 }
116 }
117 }

```

## 结果框

```

1 void MainWindow::Ping1Result(Ping ping)
2 {
3     ui->PingButton1->setDisabled(false);
4     ui->PingButton2->setDisabled(false);
5     if(times >1 && numSend == times )
6     {
7         strcat(message,inet_ntoa(*(struct in_addr*)ping.host->h_addr_list[0]));
8         strcat(message," 的 Ping 统计信息:\n    数据包: 已发送 = ");
9         char temp[10];
10        sprintf(temp,"%d",numSend);
11        strcat(message,temp);
12        strcat(message," 已接收 = ");
13        temp[0]='\0';
14        sprintf(temp,"%d",numReceive);
15        strcat(message,temp);
16        strcat(message," 丢失 = ");
17        temp[0]='\0';
18        sprintf(temp,"%d",numSend-numReceive);
19        strcat(message,temp);
20        strcat(message," (");
21        temp[0]='\0';
22        sprintf(temp,"%d",100*(numSend-numReceive)/numSend);
23        strcat(message,temp);
24        strcat(message,"% 丢失)。 \n");
25        strcat(message," \n");
26        numSend=0;
27        numReceive=0;
28    }
29    ui->Result->setText(message);
30    ui->Result->moveCursor(QTextCursor::End,QTextCursor::MoveAnchor); //移动光标到末尾
31 }

```

## 四、实验结果

### (1) 执行单词ping命令

① 在地址栏输入[www.163.com](http://www.163.com)的默认状态结果

MainWindow

计算机网络实验6-实现 Ping 命令

地址(Address):

www.163.com

执行单次Ping命令

包数(PacketNum Default:4):

查询子网中可Ping通主机

子网(Subnet):

终端结果  
(Results):

正在 Ping z163picipv6.v.bsgslb.cn[113.200.41.56] 具有 32 字节的数据:  
来自 113.200.41.56 的回复: 字节=32 时间=16ms TTL=53  
来自 113.200.41.56 的回复: 字节=32 时间=16ms TTL=53  
来自 113.200.41.56 的回复: 字节=32 时间=16ms TTL=53  
来自 113.200.41.56 的回复: 字节=32 时间=16ms TTL=53  
113.200.41.56 的 Ping 统计信息:  
数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失)。

作者: 161910110 万晔 指导老师: 燕雪峰 指导助教: 王永振

② 在地址栏填写www.163.com 在包数栏填写7 的结果

MainWindow

—□×

计算机网络实验6-实现 Ping 命令

地址(Address):

www.163.com

执行单次Ping命令

包数(PacketNum Default:4):

7

查询子网中可Ping通主机

子网(Subnet):

终端结果  
(Results):

正在 Ping z163picipv6.v.bsgslb.cn[113.200.41.56] 具有 32 字节的数据:  
来自 113.200.41.56 的回复: 字节=32 时间=16ms TTL=53  
来自 113.200.41.56 的回复: 字节=32 时间=16ms TTL=53  
来自 113.200.41.56 的回复: 字节=32 时间=16ms TTL=53  
来自 113.200.41.56 的回复: 字节=32 时间=16ms TTL=53  
来自 113.200.41.56 的回复: 字节=32 时间=16ms TTL=53  
来自 113.200.41.56 的回复: 字节=32 时间=16ms TTL=53  
来自 113.200.41.56 的回复: 字节=32 时间=16ms TTL=53  
113.200.41.56 的 Ping 统计信息:  
数据包: 已发送 = 7, 已接收 = 7, 丢失 = 0 (0% 丢失)。

作者: 161910110 万晔 指导老师: 燕雪峰 指导助教: 王永振

③ 在地址栏输入 [www.google.com](http://www.google.com) 出现异常情况

MainWindow

计算机网络实验6-实现 Ping 命令

地址(Address):

www.google.com

执行单次Ping命令

包数(PacketNum Default:4):

5

查询子网中可Ping通主机

子网(Subnet):

终端结果  
(Results):

正在 Ping www.google.com[142.251.42.228] 具有 32 字节的数据:  
来自 142.251.42.228 的回复: 请求超时。  
来自 142.251.42.228 的回复: 请求超时。  
来自 142.251.42.228 的回复: 请求超时。  
来自 142.251.42.228 的回复: 请求超时。

正在 Ping www.google.com[142.251.42.228] 具有 32 字节的数据:  
来自 142.251.42.228 的回复: 请求超时。  
来自 142.251.42.228 的回复: 请求超时。  
来自 142.251.42.228 的回复: 请求超时。  
来自 142.251.42.228 的回复: 请求超时。  
来自 142.251.42.228 的回复: 请求超时。

作者: 161910110 万晔 指导老师: 燕雪峰 指导助教: 王永振

## (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

MainWindow

计算机网络实验6-实现 Ping 命令

地址(Address):

执行单次Ping命令

包数(PacketNum Default:4):

查询子网中可Ping通主机

子网(Subnet):

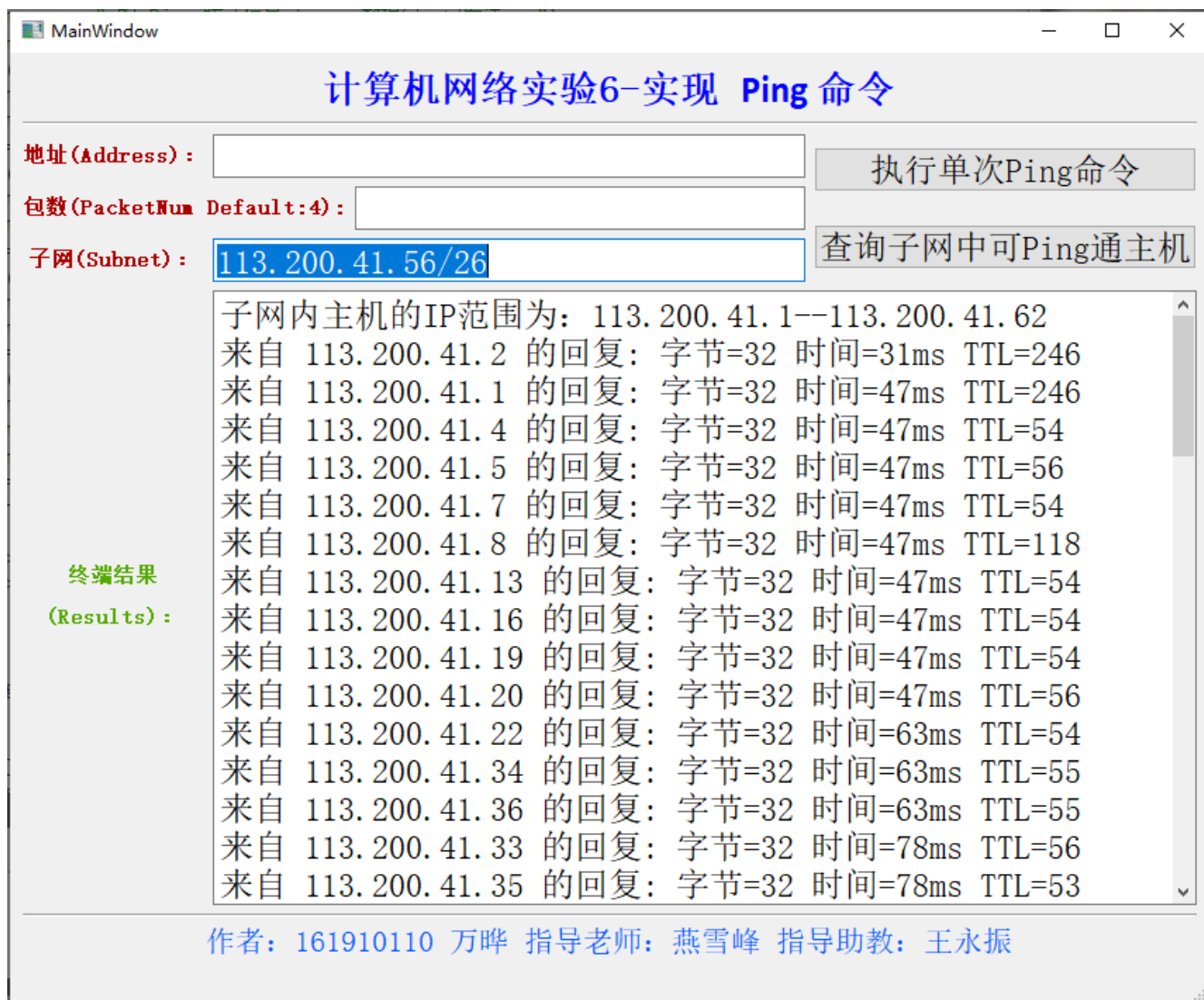
113.200.41.56/28

终端结果  
(Results):

子网内主机的IP范围为: 113.200.41.49--113.200.41.62  
来自 113.200.41.51 的回复: 字节=32 时间=31ms TTL=55  
来自 113.200.41.50 的回复: 字节=32 时间=31ms TTL=53  
来自 113.200.41.52 的回复: 字节=32 时间=31ms TTL=53  
来自 113.200.41.54 的回复: 字节=32 时间=31ms TTL=55  
来自 113.200.41.53 的回复: 字节=32 时间=31ms TTL=55  
来自 113.200.41.55 的回复: 字节=32 时间=31ms TTL=53  
来自 113.200.41.56 的回复: 字节=32 时间=31ms TTL=53  
来自 113.200.41.49 的回复: 请求超时。  
来自 113.200.41.57 的回复: 请求超时。  
来自 113.200.41.60 的回复: 请求超时。  
来自 113.200.41.58 的回复: 请求超时。  
来自 113.200.41.59 的回复: 请求超时。  
来自 113.200.41.61 的回复: 请求超时。  
来自 113.200.41.62 的回复: 请求超时。

作者: 161910110 万晔 指导老师: 燕雪峰 指导助教: 王永振

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

```
1 #ifndef MAINWINDOW_H
2 #define MAINWINDOW_H
3
```

```

4  #include <QMainWindow>
5  #include <QThread>
6  #include "Ping.h"
7  QT_BEGIN_NAMESPACE
8  namespace Ui { class MainWindow; }
9  QT_END_NAMESPACE
10
11 class MainWindow : public QMainWindow
12 {
13     Q_OBJECT
14     public:
15         MainWindow(QWidget *parent = nullptr);
16         ~MainWindow();
17
18     private:
19         Ui::MainWindow *ui;
20     public slots:
21         void on_PingButton1_clicked();
22         void Ping1Result(Ping);
23         void on_PingButton2_clicked();
24 };
25
26 class PingThread : public QThread
27 {
28     Q_OBJECT
29     public:
30         PingThread(char addr[100]);
31     signals:
32         void isDone(Ping); //处理完成信号
33     protected:
34         char addr[100];
35         void run(); //通过start()间接调用
36
37 };
38
39 #endif // MAINWINDOW_H
40

```

## ping.h

```

1  #ifndef PING_H
2  #define PING_H
3  #include <winsock.h>
4  #include <stdio.h>
5
6
7
8  class IPHeader
9  {
10     public:
11         //u_char 占1个字节
12         //u_short 占两个字节

```

```

13         //u_char version;           // 版本
14         //u_char head_len;          // 首部长度
15         u_char ver_headlen;          // 版本+首部长度
16         u_char service;              // 服务类型
17         u_short total_len;           // 总长度
18         u_short id;                  // 标识符
19         u_short flag;                // 标记+片偏移
20         u_char ttl;                  // 存活时间
21         u_char protocol;             // 协议
22         u_short check_sum;           // 首部校验和
23         u_int src_IP;                // 源IP地址
24         u_int dst_IP;                // 目的IP地址
25     };
26
27     class ICMPHeader
28     {
29     public:
30         u_char type;                  // 类型
31         u_char code;                  // 代码
32         u_short check_sum;            // 校验和
33         u_short id;                   // 标示符 标识本进程
34         u_short seq;                  // 序列号
35     };
36
37
38
39
40     class Ping
41     {
42     public:
43         char in[100];                 //从输入框读入
44         sockaddr_in dst_IP;           //目标IP
45         struct hostent *host;
46         IPHeader ipHeader;
47         ICMPHeader icmpHeader;
48         SOCKET sock;
49         char message[5000];
50         bool received;
51         bool getIP();                 // 得到ping的IP地址
52         bool send();                  // 发送ICMP报文请求
53         bool receive();               // 接收ICMP报文,解析并
54
55         回显
56         u_short check_sum(u_short* buffer, int len); // 计算校验和
57     };
58
59
60
61     //ICMP时间戳请求报文
62     struct EchoRequest
63     {
64         ICMPHeader icmp_header;       //ICMP头部

```



```

65     unsigned long long time;                //记录ping时间
66 };
67
68 #pragma pack(push) //保存对齐状态
69 #pragma pack(1) //设定为1字节对齐
70 struct EchoResponse
71 {
72     IPHeader ip_header;
73     EchoRequest echo_request;
74 };
75 #pragma pack(pop) //恢复对齐状态
76
77 #endif // PING_H
78

```

## 6.2 Sources 源文件

### main.cpp

```

1  #include "mainwindow.h"
2  #include <QApplication>
3  #include <winsock.h>
4
5  int main(int argc, char *argv[])
6  {
7      WSADATA wsa;
8      WSStartup(MAKEWORD(2, 2), &wsa); //初始化Windows Socket
9
10     QApplication a(argc, argv);
11     MainWindow w;
12     w.show();
13
14     return a.exec();
15 }

```

### mainwindow.cpp

```

1  #include "mainwindow.h"
2  #include "ui_mainwindow.h"
3  #include <Windows.h>
4  #include <QMessageBox>
5  #include "Ping.h"
6  #include <thread>
7  #include <string.h>
8  #include <math.h>
9
10 char message[50000];
11 char IP[30];
12 int numSend=0;
13 int numReceive=0;
14 int times=4;
15

```

```

16 MainWindow::MainWindow(QWidget *parent)
17     : QMainWindow(parent)
18     , ui(new Ui::MainWindow)
19 {
20
21     qRegisterMetaType<Ping>("Ping");
22     ui->setupUi(this);
23
24 }
25
26 MainWindow::~MainWindow()
27 {
28     delete ui;
29     WSACleanup();
30 }
31 void MainWindow::on_PingButton1_clicked()
32 {
33     ui->PingButton1->setDisabled(true);
34     ui->PingButton2->setDisabled(true);
35     numSend=0;
36     numReceive=0;
37     times=4;
38     strcat(message, "\n");
39     char temp[100];
40     strcpy(temp, ui->Addr->text().toLatin1().data());
41
42     char tmp[10];
43
44     strcpy(tmp, ui->Addr_2->text().toLatin1().data());
45     sscanf(tmp, "%d", &times);
46     if(times > 0)
47     {
48         Ping pingTemp;
49         strcpy(pingTemp.in, temp);
50         if(pingTemp.getIP())
51         {
52             strcat(message, "正在 Ping ");
53             strcat(message, pingTemp.host->h_name);
54             strcat(message, "[");
55             strcat(message, inet_ntoa(*(struct in_addr*)pingTemp.host-
56 >h_addr_list[0]));
57             strcat(message, "] 具有 32 字节的数据:\n");
58             ui->Result->setText(message);
59             for(int i=0; i<times; i++)
60             {
61                 PingThread * ping = new PingThread(temp);
62                 connect(ping, SIGNAL(isDone(Ping)), this, SLOT(Ping1Result(Ping)));
63                 ping->start();
64             }
65         }
66         else
67         {
68             strcat(message, "Ping 请求找不到主机 ");
69         }
70     }
71 }

```

```

68         strcat(message,temp);
69         strcat(message,"。请检查该名称，然后重试。\\n");
70         ui->Result->setText(message);
71         PingThread * ping = new PingThread(temp);
72         connect(ping,SIGNAL(isDone(Ping)),this,SLOT(Ping1Result(Ping)));
73         ping->start();
74     }
75 }
76 else
77 {
78     PingThread * ping = new PingThread(temp);
79     connect(ping,SIGNAL(isDone(Ping)),this,SLOT(Ping1Result(Ping)));
80     ping->start();
81 }
82 }
83
84 void PingThread::run()
85 {
86     Ping ping;
87     strcpy(ping.in, addr);
88     ping.sock=socket(AF_INET,SOCK_RAW,IPPROTO_ICMP);
89     if(times >0)
90     {
91         if(ping.send())
92         {
93             numSend++;
94             printf("%d\\n",numSend);
95         }
96         if(ping.received)
97         {
98             numReceive++;
99         }
100         strcat(message,ping.message);
101     }
102     emit isDone(ping);
103 }
104 PingThread::PingThread(char _addr[100])
105 {
106     strcpy(addr, _addr);
107 }
108
109 void MainWindow::Ping1Result(Ping ping)
110 {
111     ui->PingButton1->setDisabled(false);
112     ui->PingButton2->setDisabled(false);
113     if(times >1 && numSend == times )
114     {
115         strcat(message,inet_ntoa(*(struct in_addr*)ping.host->h_addr_list[0]));
116         strcat(message," 的 Ping 统计信息:\\n    数据包：已发送 = ");
117         char temp[10];
118         sprintf(temp,"%d",numSend);
119         strcat(message,temp);
120         strcat(message,"，已接收 = ");

```

```

121     temp[0]='\0';
122     sprintf(temp,"%d",numReceive);
123     strcat(message,temp);
124     strcat(message," 丢失 = ");
125     temp[0]='\0';
126     sprintf(temp,"%d",numSend-numReceive);
127     strcat(message,temp);
128     strcat(message," (");
129     temp[0]='\0';
130     sprintf(temp,"%d",100*(numSend-numReceive)/numSend);
131     strcat(message,temp);
132     strcat(message,"% 丢失)。 \n");
133     strcat(message, "\n");
134     numSend=0;
135     numReceive=0;
136 }
137 ui->Result->setText(message);
138 ui->Result->moveCursor(QTextCursor::End,QTextCursor::MoveAnchor); //移动光标到末
尾
139 }
140
141
142 char *IPToString(u_int IP_int)
143 {
144     IP[0]='\0';
145     char temp[5];
146     sprintf(temp,"%u",(IP_int >> 24) & 0x000000FF);
147     strcat(IP,temp);
148     strcat(IP,".");
149     temp[0]='\0';
150     sprintf(temp,"%u",(IP_int >> 16) & 0x000000FF);
151     strcat(IP,temp);
152     strcat(IP,".");
153     temp[0]='\0';
154     sprintf(temp,"%u",(IP_int >> 8) & 0x000000FF);
155     strcat(IP,temp);
156     strcat(IP,".");
157     temp[0]='\0';
158     sprintf(temp,"%u",IP_int & 0x000000FF);
159     strcat(IP,temp);
160     temp[0]='\0';
161     return IP;
162 }
163
164
165 void MainWindow::on_PingButton2_clicked()
166 {
167     ui->PingButton1->setDisabled(true);
168     ui->PingButton2->setDisabled(true);
169     strcat(message, "\n");
170     char *temp=(char *)malloc(100);
171     strcpy(temp,ui->Addr_3->text().toLatin1().data());
172     int i=0;

```

```

173     int flag=0;
174     while(temp[i]!='\0')
175     {
176         if(temp[i] == '/')
177         {
178             flag=1;
179             break;
180         }
181         i++;
182     }
183     if(flag == 0)
184     {
185         strcat(message,"子网输入格式错误!\n(示例: 192.168.253.16/28)\n");
186         ui->Result->setText(message);
187         ui->Result->moveCursor(QTextCursor::End,QTextCursor::MoveAnchor); //移动光标
到末尾
188     }
189     else
190     {
191         int j=0;
192         temp[i]='\0';
193         i++;
194         char t[3];
195         while(temp[i]!='\0')
196         {
197             t[j]=temp[i];
198             j++;
199             i++;
200         }
201         t[j]='\0';
202         //temp现在是一个Ip地址格式
203         //t现在是一个数字
204         u_int range = 0;
205         sscanf(t,"%u",&range);
206         range= 32-range;
207         //子网掩码
208         u_int subnet_mask=pow(2,32)-pow(2,range);
209         u_int ip[4];
210         u_int IP=0;
211         i=0;
212         j=0;
213         int k=0;
214         while(temp[j]!='\0')
215         {
216             if(temp[j] == '.')
217             {
218                 temp[j] = '\0';
219                 sscanf(temp,"%u",&ip[k]);
220                 i=j+1;
221                 temp=&temp[i];
222                 i=0;
223                 j=0;
224                 k++;

```

```

225         continue;
226     }
227     j++;
228 }
229 temp[j] = '\0';
230 sscanf(temp,"%u",&ip[k]);
231 i=j+1;
232 k++;
233 IP = ip[0]*256*256*256+ip[1]*256*256+ip[2]*256+ip[3];
234 IP= subnet_mask & IP;
235 u_int left = IP+1;
236 int n=0;
237 n=pow(2,range)-1;
238 u_int right = (IP | n) -1;
239 strcpy(message, "子网内主机的IP范围为: ");
240 strcat(message,IPToString(left));
241 strcat(message, "--");
242 strcat(message,IPToString(right));
243 strcat(message,"\n");
244 while(left <= right)
245 {
246     char _addr[30];
247     strcpy(_addr,IPToString(left));
248     numSend=0;
249     numReceive=0;
250     times=1;
251
252     if(times > 0)
253     {
254         Ping pingTemp;
255         strcpy(pingTemp.in, _addr);
256         if(pingTemp.getIP())
257         {
258             /*
259             strcat(message,"正在 Ping ");
260             strcat(message,pingTemp.host->h_name);
261             strcat(message,"[");
262             strcat(message,inet_ntoa(*(struct in_addr*)pingTemp.host-
>h_addr_list[0]));
263             strcat(message,"] 具有 32 字节的数据:\n");
264             */
265             ui->Result->setText(message);
266             PingThread * ping = new PingThread(_addr);
267             connect(ping,SIGNAL(isDone(Ping)),this,SLOT(Ping1Result(Ping)));
268             ping->start();
269         }
270         else
271         {
272             strcat(message,"Ping 请求找不到主机 ");
273             strcat(message,_addr);
274             strcat(message,"。请检查该名称, 然后重试.\n");
275             ui->Result->setText(message);
276         }

```

```

277         }
278
279         left++;
280     }
281 }
282 }
283
284

```

## ping.cpp

```

1  #include "ping.h"
2  #include <Windows.h>
3  #include <string.h>
4
5  bool Ping::getIP() //获取IP地址
6  {
7      host = gethostbyname(in);
8      if (host == NULL)
9      {
10         return false;
11     }
12     dst_IP.sin_family = AF_INET; //IPv4
13     dst_IP.sin_addr.S_un.S_addr = *(u_long*)host->h_addr;
14     puts(host->h_name);
15     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
16     return true;
17 }
18
19 bool Ping::send() //发送数据包
20 {
21     if(getIP()==false) //获取IP失败
22     {
23         return false;
24     }
25     static int id = 1;
26     static int seq = 1;
27     EchoRequest req;
28     req.time = GetTickCount(); //从0开始计时，返回自设备启动后的毫秒数
29     req.icmp_header.type = 8; //ICMP_ECHO
30     req.icmp_header.code = 0;
31     id = ::GetCurrentProcessId(); //获取当前的进程ID
32     req.icmp_header.id = id;
33     //id++;
34     req.icmp_header.seq = seq++; //序号加一
35     req.icmp_header.check_sum = check_sum((u_short*)&req, sizeof(EchoRequest)); //校
    验和
36
37     int re = sendto(sock, (char*)&req, sizeof(req), 0, (sockaddr*)&dst_IP,
    sizeof(dst_IP));
38     //将数据由指定的socket 传给目标主机

```

```

39 //成功则返回实际传送出去的字符数，失败返回 -1，
40 // message 为此次ping的一些信息，输出到
41 if(re == SOCKET_ERROR) //SOCKET_ERROR = -1
42 {
43     strcat(message,"发送错误，错误码:");
44     char temp[10];
45     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
46     strcat(message,temp);
47     strcat(message,"\n");
48     return false;
49 }
50 if(receive())
51 {
52     received=true;
53 }
54 else
55 {
56     received=false;
57 }
58 return true;
59 }
60 u_short Ping::check_sum(u_short* buf, int len) //校验和计算
61 {
62     unsigned int sum=0;
63     unsigned short *cbuf;
64
65     cbuf=(unsigned short *)buf;
66
67     while(len>1)
68     {
69         sum+=*cbuf++;
70         len-=2;
71     }
72     if(len)
73     {
74         sum+=*(unsigned char *)cbuf;
75     }
76     sum=(sum>>16)+(sum & 0xffff);
77     sum+=(sum>>16);
78     return ~sum;
79 }
80 bool Ping::receive() //接收并分析返回的数据包
81 {
82     int timeout = 1000; //设置超时的时间
83     if(setsockopt(sock,SOL_SOCKET,SO_RCVTIMEO,(char *)&timeout,sizeof(timeout)) ==
SOCKET_ERROR) //设置套接口
84     {
85         strcat(message,"接收设置错误，错误码:"); //设置套接口返回错误
86         char temp[10];
87         sprintf(temp,"%d",WSAGetLastError());
88         strcat(message,temp);
89         strcat(message,"\n");

```



```

90         return false;
91     }
92     char temp[100];
93     strcat(message, "来自 ");
94     strcat(message, inet_ntoa(*(struct in_addr*)host->h_addr_list[0])); //地址
95     strcat(message, " 的回复: ");
96     EchoResponse* res=new EchoResponse;
97     int size = sizeof(sockaddr);
98     int re = recvfrom(sock, (char*)res, sizeof(EchoResponse), 0, (sockaddr*)&dst_IP,
    &size); //接收到的返回的套接字
99     if (re == SOCKET_ERROR) //出错
100     {
101         int code = WSAGetLastError();
102         if(code==10060)
103         {
104             strcat(message, "请求超时。 \n");
105         }
106         return false;
107     }
108     unsigned long long time = GetTickCount() - res->echo_request.time; //两个相减即为
    TTL时间
109     int type = res->echo_request.icmp_header.type;
110     int code = res->echo_request.icmp_header.code;
111     char TTL[10]={'\0'};
112     sprintf(TTL, "%d", (int)res->ip_header.ttl);
113     delete res;
114     if(type==0&&code==0) //输出到message信息中
115     {
116         strcat(message, "字节=32 时间=");
117         sprintf(temp, "%I64u", time);
118         strcat(message, temp);
119         strcat(message, "ms TTL=");
120         strcat(message, TTL);
121         strcat(message, "\n");
122         return true;
123     }
124     else
125     {
126         strcat(message, "请求超时");
127         strcat(message, "\n");
128         return false;
129     }
130 }
131 }
132

```

## 6.3 Forms UI文件

mainwindow.ui

```

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

```

```

3 <class>MainWindow</class>
4 <widget class="QMainWindow" name="MainWindow">
5 <property name="geometry">
6 <rect>
7 <x>0</x>
8 <y>0</y>
9 <width>852</width>
10 <height>676</height>
11 </rect>
12 </property>
13 <property name="windowTitle">
14 <string>MainWindow</string>
15 </property>
16 <widget class="QWidget" name="centralwidget">
17 <layout class="QGridLayout" name="gridLayout">
18 <item row="0" column="0" colspan="4">
19 <widget class="QLabel" name="label">
20 <property name="text">
21 <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>
22 </property>
23 </widget>
24 </item>
25 <item row="1" column="0" colspan="4">
26 <widget class="Line" name="line_2">
27 <property name="orientation">
28 <enum>Qt::Horizontal</enum>
29 </property>
30 </widget>
31 </item>
32 <item row="2" column="0">
33 <widget class="QLabel" name="label_3">
34 <property name="text">
35 <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>
36 </property>
37 </widget>
38 </item>
39 <item row="2" column="1" colspan="2">
40 <widget class="QLineEdit" name="Addr">
41 <property name="cursor">
42 <cursorShape>IBeamCursor</cursorShape>
43 </property>
44 <property name="toolTip">

```

```

45     <string>&lt;html&gt;&lt;head/amp;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;
&lt;/string>
46     </property>
47     <property name="placeholderText">
48     <string/>
49     </property>
50 </widget>
51 </item>
52 <item row="2" column="3" rowspan="2">
53     <widget class="QPushButton" name="PingButton1">
54     <property name="cursor">
55     <cursorShape>PointingHandCursor</cursorShape>
56     </property>
57     <property name="text">
58     <string>执行单次Ping命令&lt;/string>
59     </property>
60     </widget>
61 </item>
62 <item row="3" column="0" rowspan="2" colspan="2">
63     <widget class="QLabel" name="label_5">
64     <property name="text">
65     <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;
/amp;style type=&quot;text/css&quot;&gt;
66 &lt;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;
67 &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;&lt;/string>
70     </property>
71     </widget>
72 </item>
73 <item row="3" column="2" rowspan="2">
74     <widget class="QLineEdit" name="Addr_2">
75     <property name="toolTip">
76     <string>&lt;html&gt;&lt;head/amp;body&gt;&lt;p&gt;输入包数 如:
5&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;&lt;/string>
77     </property>
78     <property name="placeholderText">
79     <string/>
80     </property>
81     </widget>
82 </item>
83 <item row="4" column="3" rowspan="2">
84     <widget class="QPushButton" name="PingButton2">
85     <property name="cursor">
86     <cursorShape>PointingHandCursor</cursorShape>
87     </property>
88     <property name="whatsThis">

```

```

89         <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>
90     </property>
91     <property name="text">
92         <string>查询子网中可Ping通主机</string>
93     </property>
94 </widget>
95 </item>
96 <item row="5" column="0">
97     <widget class="QLabel" name="label_6">
98         <property name="text">
99             <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>
100         </property>
101     </widget>
102 </item>
103 <item row="5" column="1" colspan="2">
104     <widget class="QLineEdit" name="Addr_3">
105         <property name="toolTip">
106             <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>
107         </property>
108         <property name="whatsThis">
109             <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>
110         </property>
111         <property name="placeholderText">
112             <string/>
113         </property>
114     </widget>
115 </item>
116 <item row="6" column="0">
117     <widget class="QLabel" name="label_4">
118         <property name="text">
119             <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;
&lt;/string>
120         </property>
121     </widget>
122 </item>
123 <item row="6" column="1" colspan="3">
124     <widget class="QTextEdit" name="Result">
125         <property name="enabled">
126             <bool>true</bool>
127         </property>
128         <property name="maximumSize">
129             <size>
130                 <width>16777215</width>

```

```

131         <height>800</height>
132     </size>
133 </property>
134 <property name="cursor" stdset="0">
135     <cursorShape>ArrowCursor</cursorShape>
136 </property>
137 </widget>
138 </item>
139 <item row="7" column="0" colspan="4">
140     <widget class="Line" name="line">
141         <property name="orientation">
142             <enum>Qt::Horizontal</enum>
143         </property>
144     </widget>
145 </item>
146 <item row="8" column="0" colspan="4">
147     <widget class="QLabel" name="label_2">
148         <property name="text">
149             <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p
align="center"&gt;&lt;span style="font-size:16pt;
color:#0055ff;"&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>
150         </property>
151     </widget>
152 </item>
153 </layout>
154 </widget>
155 <widget class="QStatusBar" name="statusbar"/>
156 <widget class="QMenuBar" name="menubar">
157     <property name="geometry">
158         <rect>
159             <x>0</x>
160             <y>0</y>
161             <width>852</width>
162             <height>21</height>
163         </rect>
164     </property>
165 </widget>
166 </widget>
167 <resources/>
168 <connections/>
169 </ui>
170

```