湖南科技大学计算机科学与工程学院

计算机网络 课程设计报告

专业班级： 计算机科学与技术七班

姓　　名： 陈琪琪

学　　号： 2102010629

指导教师： 崔振河

时　　间： 2023.12.25-2023.12.29

地　　点： 逸夫楼 416

指导教师评语：
成绩：　　等级： 签名： 年　月　日

目录

实验一、网络聊天程序的设计与实现

一、实验目的

参照附录 1，了解 Socket 通信的原理，在此基础上编写一个聊天程序。

二、总体设计

1. 背景知识

(1) WinSock 编程：是一种网络编程接口，实际上是作为 TCP/IP 协议的一种封装。可以通过调用 WinSock 的接口函数来调用 TCP/IP 的各种功能。

(2) WinSock 编程简单流程：WinSock 编程分为服务器端和客户端两部分。

(3) TCP 服务器端的大体流程：对于任何基于 WinSock 的编程首先必须要初始化 WinSock DLL 库。int WSAStarup(WORD wVersionRequested,LPWSADATA lpWsAData)。wVersionRequested 是我们要求使用的 WinSock 的版本。调用这个接口函数可以初始化 WinSock。然后必须创建一个套接字。
(Socket)WinSock 通讯的所有数据传输，都是通过套接字来完成的，套接字包含了两个信息，一个是 IP 地址，一个是 Port 端口号，使用这两个信息，就可以确定网络中的任何一个通讯节点。

(4) Tcp 协议(Transmission Control Protocol 传输控制协议)：是面向连接的运输层协议，提供可靠的、有序的、双向的、面向连接的运输服务。

2. 设计步骤

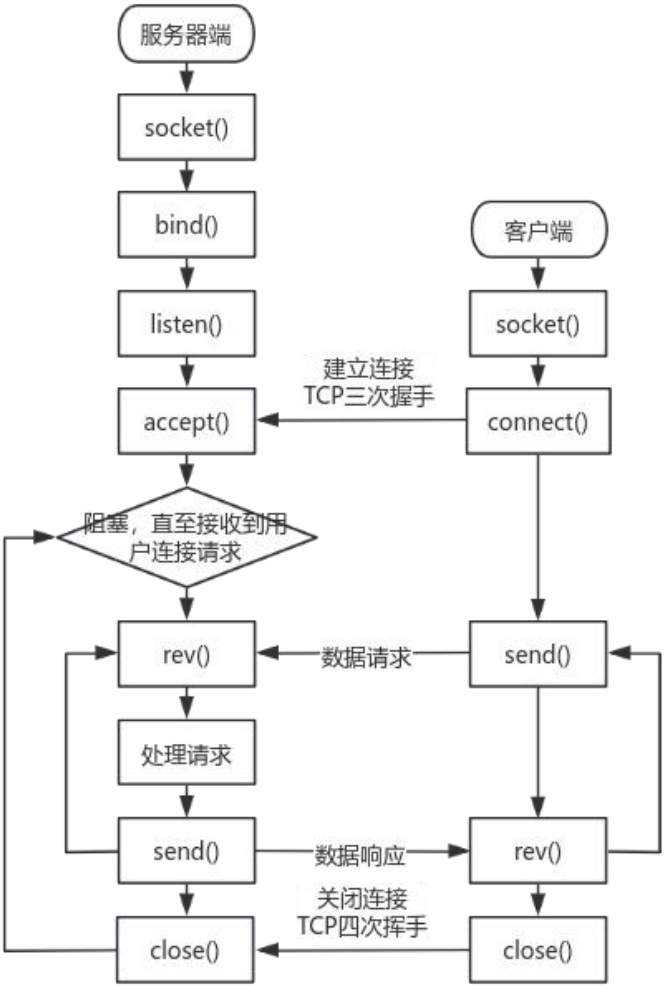
说明：sockets（套接字）编程有三种，流式套接字（SOCK_STREAM），数据报套接字（SOCK_DGRAM），原始套接字（SOCK_RAW）；基于 TCP 的 socket 编程是采用的流式套接字。在这个程序中，将两个工程添加到一个工作区。要链接一个 ws2_32.lib 的库文件(#pragma comment(lib,“ws2_32”))。

- 服务器端编程的步骤：
  (1) 加载套接字库，创建套接字(WSAStartup()/socket())；
  (2) 绑定套接字到一个 IP 地址和一个端口上(bind())；
  (3) 将套接字设置为监听模式等待连接请求(listen())；
  (4) 请求到来后，接受连接请求，返回一个新的对应于此次连接的套接字(accept())；
  (5) 用返回的套接字和客户端进行通信(send()/recv())；
  (6) 返回，等待另一连接请求；
  (7) 关闭套接字，关闭加载的套接字库(closesocket()/WSACleanup())。

- 客户端编程的步骤：
  (1) 加载套接字库，创建套接字(WSAStartup()/socket())；
  (2) 向服务器发出连接请求(connect())；
  (3) 和服务器端进行通信(send()/recv())；
  (4) 关闭套接字，关闭加载的套接字库(closesocket()/WSACleanup())。

三、详细设计

1. 流程图



2. 关键代码

(1) 服务器端

```
    //1、初始化 socket 库
    WSADATA wsaData;//获取版本信息，说明要使用的版本
    WSAStartup(MAKEWORD(2, 2), &wsaData);//MAKEWORD(主版本号, 副版本号)

    //2、创建 socket
    SOCKET servSock = socket(AF_INET, SOCK_STREAM, 0);//流式套接字（基于 TCP 协议）

    //3、将服务器地址打包在一个结构体里面
    SOCKADDR_IN servAddr; //sockaddr_in 是 internet 环境下套接字的地址形式
```

```
10.     servAddr.sin_family = AF_INET;//和服务器的socket 一样，sin_family
    表示协议簇，一般用AF_INET 表示TCP/IP 协议。
11.     servAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");//服务端地
    址设置为本地回环地址
12.     servAddr.sin_port = htons(12345);//host to net short 端口号设置为
    12345
13.
14.     //4、绑定服务端的socket 和打包好的地址
15.     bind(servSock, (SOCKADDR*)&servAddr, sizeof(servAddr));
16.
17.     //4.5 给服务端sokect 绑定一个事件对象，用来接收客户端连接的事件
18.     WSAEVENT servEvent = WSACreateEvent();//创建一个人工重设为传信的事
    件对象
19.     WSAEventSelect(servSock, servEvent, FD_ALL_EVENTS);//绑定事件对象，
    并且监听所有事件
20.
21.     cliSock[0] = servSock;
22.     cliEvent[0] = servEvent;
23.
24.     //5、开启监听，把套接字转成监听模式。
25.     listen(servSock, 10);//监听队列长度为10
26.
27.     //6、创建接受客户端连接，接收客户端消息的线程
28.     //不需要句柄所以直接关闭
29.     CloseHandle(CreateThread(NULL, 0,
30.      servEventThread, (LPVOID)&servSock, //线程函数名，参数
31.      0, 0));
32.
33.     cout << "聊天室服务器已开启，服务端允许的最大同时连接数
    为 " << MAX_LINK_NUM << "。" << endl;
34.
35.     bool run=true;
36.     //需要让主线程一直运行下去
37.     //发送消息给全部客户端
38.     while (run) {
39.
40.      char contentBuf[BUFFER_SIZE] = { 0 };
41.      char sendBuf[BUFFER_SIZE] = { 0 };
42.      cin.getline(contentBuf, sizeof(contentBuf));
43.      if (strcmp(contentBuf, "quit") == 0)//若输入“quit”，则关闭聊天室，
    服务器退出
44.      {
45.       run=false;
46.       strcpy(contentBuf,"服务器已经关闭！");
```

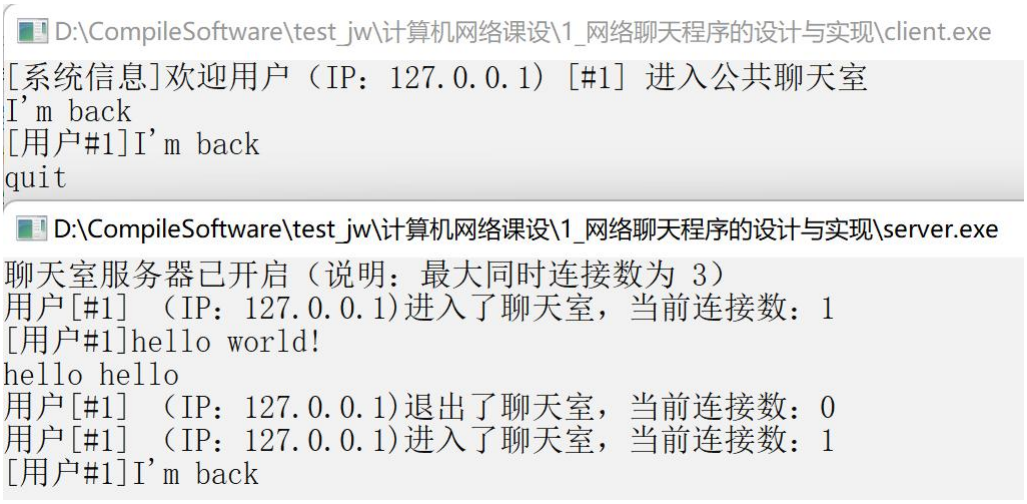
```
47.       }
48.       sprintf(sendBuf, "[系统信息]%s", contentBuf);
49.       //给每个客户端发送（从 1 开始，不给服务器发送）
50.       for (int j = 1; j <= total; j++) {
51.        send(cliSock[j], sendBuf, sizeof(sendBuf), 0);
52.       }
53.
54.      }
55.      //1-关闭 socket 库的收尾工作
56.      for(int i=0;i<total;i++){
57.       closesocket(cliSock[i]);
58.       WSACloseEvent(cliEvent[i]);
59.      }
60.      WSACleanup();
61.      return 0;
```

(2) 客户端

```
1.     DWORD WINAPI recvMsgThread(LPVOID IpParameter)//接收消息的线程
2.     {
3.      SOCKET cliSock = *(SOCKET*)IpParameter;//获取客户端的 SOCKET 参数
4.
5.      while (1)
6.      {
7.       char buffer[BUFFER_SIZE] = { 0 };//字符缓冲区，用于接收和发送消息
8.       int nrecv = recv(cliSock, buffer, sizeof(buffer), 0);//nrecv 是
    接收到的字节数
9.       if (nrecv > 0)//如果接收到的字符数大于 0
10.      {
11.       cout << buffer << endl;
12.      }
13.      else if (nrecv < 0)//如果接收到的字符数小于 0 就说明断开连接
14.      {
15.       cout << "与服务器断开连接" << endl;
16.       break;
17.      }
18.     }
19.     return 0;
20.    }
```

四、实验结果与分析

1. 实验结果



2. 实验分析

在本次网络聊天程序的设计与实现中，服务器端能够正确地接受、处理和广播客户端的连接、断开和消息。实验中，可以模拟多个客户端连接服务器，观察服务器的响应。通过输入消息，验证消息能够被服务器正确接收并广播给其他在线客户端。此外，服务器在达到连接上限时能够进行合理处理，接受连接并发送繁忙提示后再断开连接。通过实验，可以深入理解多线程编程和基于事件的网络编程，以及在服务器端如何处理并发连接和事件。这个简单而实用的聊天室服务器提供了一个实用的网络编程范例，为理解和实践服务器端编程奠定了基础。

五、小结与心得体会

在学习和实验网络聊天程序的设计与实现代码的过程中，我深刻领悟到网络编程的复杂性和挑战。通过阅读和分析代码，我更清晰地理解了多线程、套接字和事件处理在网络应用中的作用。实践中，通过模拟多个客户端连接服务器，观察服务器端的响应和广播效果，我得以验证代码的正确性。尤其在处理连接上限时，服务器能够妥善处理，给予繁忙提示并关闭连接，展现了良好的健壮性。此外，通过输入消息测试，我观察到服务器能够准确地接收并广播消息，完成了基本通信功能。这次实验为我提供了一个深入理解网络编程和实践的机会，也让我更加熟悉了 Windows 套接字编程的相关知识。通过不断调试和改进代码，我进一步提高了问题定位和解决的能力。

总体而言，这个实验为我打开了网络编程的大门，让我更自信地面对复杂的网络应用开发。

实验二、Tracert 与 Ping 程序设计与实现

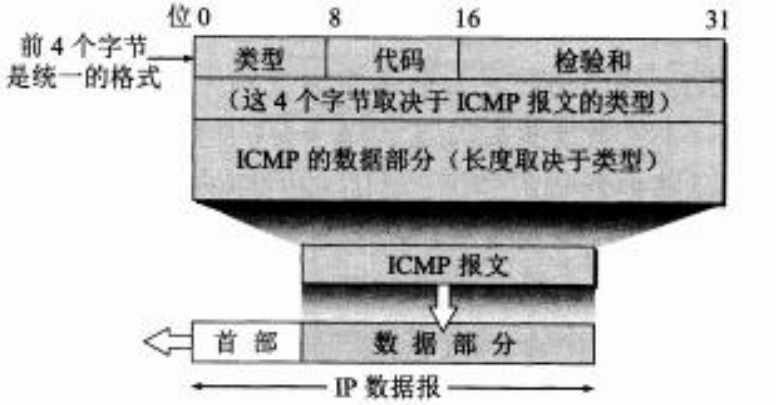
一、实验目的

参照附录 2，了解 Tracert 程序的实现原理，并调试通过。然后参考 Tracert 程序和教材 4.4.2 节，编写一个 Ping 程序，并能测试本局域网的所有机器是否在线，运行界面如下图所示的 QuickPing 程序。

二、总体设计

1. 背景知识

(1) 网际控制报文协议 ICMP：为了更有效地转发 IP 数据报和提高交付成功的机会，在网际层使用了网际控制报文协议 ICMP (Internet Control Message Protocol) [RFC 792]。ICMP 允许主机或路由器报告差错情况和提供有关异常情况的报告。ICMP 是 IP 层的协议。ICMP 报文作为 IP 层数据报的数据，加上数据报的首部，组成 IP 数据报发送出去。ICMP 报文格式如图所示。



(2) 分组网间探测 PING(Packet InterNet Groper)：用来测试两台主机之间的连通性。PING 使用了 ICMP 回送请求与回送回答报文。PING 是应用层直接使用网络层 ICMP 的一个例子。它没有通过运输层的 TCP 或 UDP。

(3) Traceroute：它用来跟踪一个分组从源点到终点的路径。在 Windows 操作系

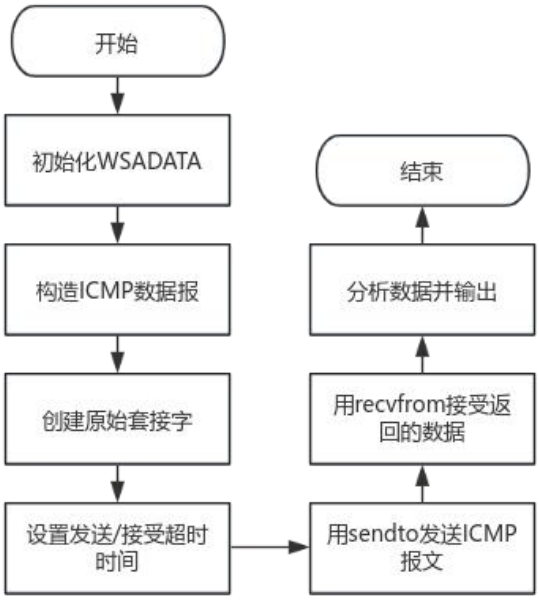
统中这个命令是 tracert。Traceroute 从源主机向目的主机发送一连串的 IP 数据报，数据报中封装的是无法交付的 UDP 用户数据报。第一个数据报 P1 的生存时间 TTL 设置为 1。当 P 到达路径上的第一个路由器 R1 时，路由器 R1 先收下它，接着把 TTL 的值减 1。由于 TTL 等于零了，R1 就把 P1 丢弃了，并向源主机发送一个 ICMP 时间超过差错报告报文。源主机接着发送第二个数据报 P2，并把 TTL 设置为 2。P2 先到达路由器 R1，R1 收下后把 TTL 减 1 再转发给路由器 R2。R2 收到 P2 时 TTL 为 1，但减 1 后 TTL 变为零了。R2 就丢弃 P2，并向源主机发送一个 ICMP 时间超过差错报告报文。这样一直继续下去。当最后一个数据报刚刚到达目的主机时，数据报的 TTL 是 1。主机不转发数据报，也不把 TTL 值减 1。但因 IP 数据报中封装的是无法交付的运输层的 UDP 用户数据报，因此目的主机要向源主机发送 ICMP 终点不可达差错报告报文。这样，源主机可以获得这些路由器和最后目的主机发来的 ICMP 报文给出了的路由信息——到达目的主机所经过的路由器的 IP 地址，以及到达其中的每一个路由器的往返时间。

2. 设计步骤

说明：Ping 和 Tracert 两个程序的原理是相通的，都是利用了 ICMP。因此只需在附录给的代码的基础上修改 TTL 的值，使之尽可能地顺利到达目标主机，避免在中间节点 TTL 就减为 0 了，并且因为 Ping 只需知道目标主机是否在线的结果，因此去掉了逐步递增 TTL 的循环，再加上递增 IP 地址的函数即可。

整个程序大致流程如下：

(1) 初始化 WSADATA
(2) 构造 ICMP 数据报
(3) 创建原始套接字
(4) 设置发送/接受超时时间
(5) 用 sendto 发送 ICMP 报文
(6) 用 recvfrom 接受返回的数据
(7) 分析数据并输出。

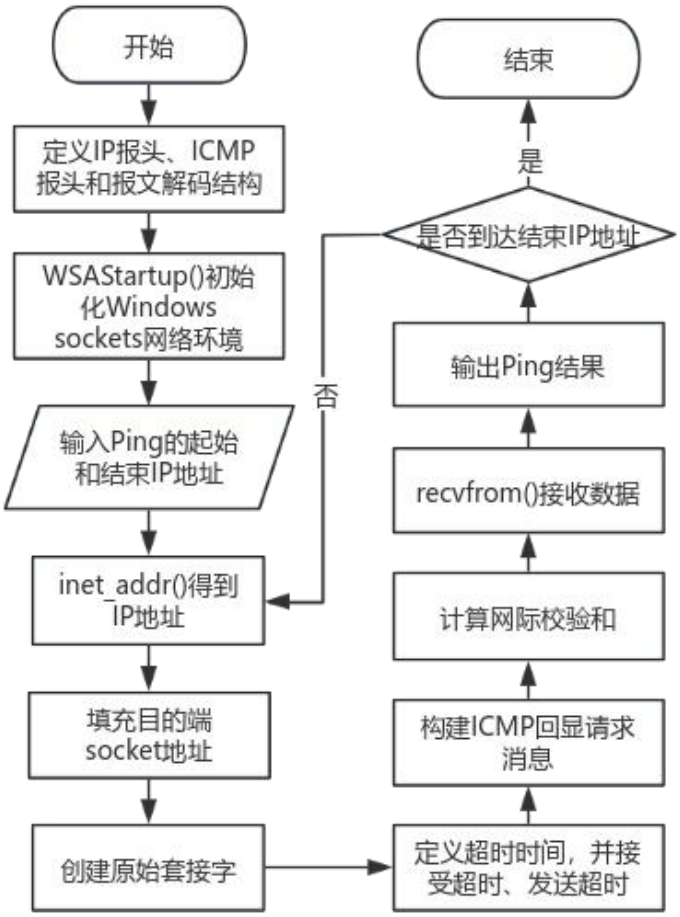


三、详细设计

1. 详细流程

(1) 结构体定义 IP 报头（4 位头部长度、4 位版本号、8 位服务类型、16 位总长度、16 位标识符、3 位标志 + 13 位片偏移、8 位生存时间、8 位上层协议号、16 位 校验和、32 位源 IP 地址、32 位目的 IP 地址）；
(2) 结构体定义 ICMP 报头（8 位类型字段、8 位代码字段、16 位校验和、16 位标识符、16 位序列号）；
(3) 结构体定义报文解码结构（序列号、往返时间、返回报文的 IP 地址）；
(4) WSAStartup()初始化 Windows sockets 网络环境；
(5) 输入需要 Ping 的起始 IP 和结束 IP 地址；
(6) inet_addr()得到 IP 地址；
(7) 填充目的端 socket 地址；
(8) WSASocket()创建原始套接字；
(9) 定义超时时间 iTimeout = 3000，并接受超时、发送超时；
(10) 构建 ICMP 回显请求消息，并以 TTL 递增的顺序发送报文；
(11) 调用编写的 checksum()函数计算网际校验和；
(12) 接收 ICMP 差错报文并解析，recvfrom()接收数据；
(13) 输出 Ping 结果；
(14) 找到下一个要 Ping 的 IP 地址，回到步骤 6，直至到达结束 IP 地址；

2. 流程图



3. 关键代码

```
    while (iMaxHot--)
    {
     //设置 IP 报头的 TTL 字段
     setsockopt(sockRaw, IPPROTO_IP, IP_TTL, (char*)&iTTL, sizeof(iTTL));
     //填充 ICMP 报文中每次发送变化的字段
     ((ICMP_HEADER*)IcmpSendBuf)->cksum = 0; //校验和先置为 0
     ((ICMP_HEADER*)IcmpSendBuf)->seq = htons(usSeqNo++); //填充序列号
     ((ICMP_HEADER*)IcmpSendBuf)->cksum = checksum((USHORT*)IcmpSendBuf,
      sizeof(ICMP_HEADER) + DEF_ICMP_DATA_SIZE); //计算校验和
      //记录序列号和当前时间
     DecodeResult.usSeqNo = ((ICMP_HEADER*)IcmpSendBuf)->seq; //当前序号
     DecodeResult.dwRoundTripTime = GetTickCount(); //当前时间
     //发送 TCP 回显请求信息
     sendto(sockRaw, IcmpSendBuf, sizeof(IcmpSendBuf), 0, (sockaddr*)&destSockAddr, sizeof(destSockAddr));
     //接收 ICMP 差错报文并进行解析处理
     sockaddr_in from; //对端 socket 地址
     int iFromLen = sizeof(from); //地址结构大小
     int iReadDataLen; //接收数据长度
     while (1)
     {
      //接收数据
      iReadDataLen = recvfrom(sockRaw, IcmpRecvBuf, MAX_ICMP_PACKET_SIZE, 0, (sockaddr*)&from, &
       iFromLen);
      if (iReadDataLen != SOCKET_ERROR)//有数据到达
      {
       //对数据包进行解码
       if (DecodeIcmpResponse(IcmpRecvBuf, iReadDataLen, DecodeResult, ICMP_ECHO_REPLY, ICMP_TIMEOUT))
       {
        //到达目的地，退出循环
        if (DecodeResult.dwIPaddr.s_addr == destSockAddr.sin_addr.s_addr) {
         //输出 IP 地址
         cout << '\t' << IpAddress << " 在线! " << endl;
         cout << endl;
         break;
        }
```

```
        }
       }
       else if (WSAGetLastError() == WSAETIMEDOUT) //接收超时，输出*号
       {
        cout << " *" << '\t' << "Request timed out. " << IpAddress << " 不在线！" << endl;
        cout << endl;
        break;
       }
       else
       {
        break;
       }
      }
      //iTTL++; //递增 TTL 值
     }
```

四、实验结果与分析

1. 实验结果

```
请输入一个 起始IP 地址或域名：39.101.201.12
请输入一个 终止IP 地址或域名：39.101.201.20
正在Ping 39.101.201.12 with a maximum of 30 hops.
 *      Request timed out. 39.101.201.12 不在线！

正在Ping 39.101.201.13 with a maximum of 30 hops.
 78ms   39.101.201.13 在线！

正在Ping 39.101.201.14 with a maximum of 30 hops.
 *      Request timed out. 39.101.201.14 不在线！

正在Ping 39.101.201.15 with a maximum of 30 hops.
 *      Request timed out. 39.101.201.15 不在线！

正在Ping 39.101.201.16 with a maximum of 30 hops.
 *      Request timed out. 39.101.201.16 不在线！

正在Ping 39.101.201.17 with a maximum of 30 hops.
 78ms   39.101.201.17 在线！

正在Ping 39.101.201.18 with a maximum of 30 hops.
 78ms   39.101.201.18 在线！

正在Ping 39.101.201.19 with a maximum of 30 hops.
 *      Request timed out. 39.101.201.19 不在线！
```

2. 实验分析

- 在线主机：对于 IP 地址 39.101.201.13、39.101.201.17 和 39.101.201.18，Ping 命令显示往返时间（Round-Trip Time，RTT）分别为 78ms、78ms 和 78ms，且显示为“在线”。这表示这些主机对 Ping 请求作出了回应，且 RTT 较短，网络连接相对较好。
- 不在线主机：对于 IP 地址 39.101.201.12、39.101.201.14、39.101.201.15、39.101.201.16 和 39.101.201.19，Ping 命令显示“Request timed out”和“不在线”。这表示这些主机未对 Ping 请求作出回应，可能由于网络不可达、主机防火墙设置等原因导致。

结果表明：在网络环境中，某些主机能够正常响应 Ping 请求，而另一些可能由于网络配置或设备设置等原因未能回应。RTT 的差异可能与网络拓扑、距离等因素有关。通过分析 Ping 结果，可以初步了解网络中主机的状态，帮助排除网络故障和评估网络性能。

可改进处：为提高程序的实用性，可以考虑添加更多异常情况的处理，如主机不可达时的错误信息提示。此外，通过增加对 Ping 命令的参数配置，如设置超时时间、控制发送报文的数量等，可以使程序更加灵活。

五、小结与心得体会

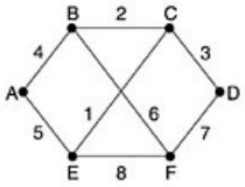
首先，程序利用 Winsock 库实现了对网络通信的基本控制，通过构建原始套接字，发送 ICMP 回显请求，并接收对应的回显应答或超时差错报文。程序结构清晰，采用了模块化设计，分别定义了 IP 报头、ICMP 报头、报文解码结构等数据结构，并封装了计算校验和、解码 ICMP 响应等功能函数，提高了代码的可维护性和可读性。其次，通过实现 IP 报头和 ICMP 报头的数据结构，充分理解了 ICMP 协议的格式和字段含义，包括头部长度、版本号、服务类型、校验和等信息。这有助于理解网络协议栈中各层次之间的关系，对网络通信原理有了更深入的认识。此外，实现了对 Ping 命令的基本功能，可以通过输入起始 IP 地址和终止 IP 地址，依次 Ping 通信，展示了网络中主机的在线状态和往返时间。通过实际运行和结果分析，可以对网络拓扑结构进行初步的了解，并在网络故障排查和性能评估中提供一些参考信息。

总体而言，通过该实验，我深入学习了网络通信原理、ICMP 协议和 Winsock 编程，提升了对网络编程的理解和实践能力。同时，实现 Ping 程序也为网络管理和故障排查提供了一个基础工具，对未来深入学习网络安全和系统运维方向奠定了基础。

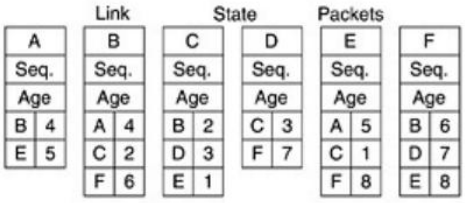
实验四、OSPF 路由协议原型系统设计与实现

一、实验目的

参考附录 4 及教材 164 页 OSPF 路由协议工作原理，在此基础上，实现一个简单的原型系统。主要完成工作有：路由节点泛洪发布本地节点的链路信息，其它节点接收信息，构造网络拓扑，然后利用 Dijkstra（或 Floyd）算法计算出到其它节点的最短路径，最后生成本节点的路由表。设计可以以下图为例，其中 a 图是一个 6 节点的网络，每个节点生成自己的链路状态包（图 b），然后将其扩散至其他节点。



(a)



(b)

二、总体设计

1. 背景知识

(1) 链路状态协议：OSPF 是一种链路状态协议，它通过交换网络中所有路由器所知道的链路状态信息来维护一个网络拓扑图。每个路由器都维护一个链路状态数据库，其中包含了整个网络的拓扑信息。

(2) 分层设计：OSPF 采用分层设计，将网络划分为区域（Area）。每个区域内的路由器了解该区域内的拓扑信息，而区域之间的路由器只需要了解彼此之间的连接关系。这有助于提高协议的可扩展性。

(3) 路由器类型：OSPF 定义了不同类型的路由器，包括内部路由器（IR，Internal Router）、边缘路由器（ER，Area Border Router）、骨干路由器（BR，Backbone Router）等。这些路由器在 OSPF 网络中扮演不同的角色，负责不同的功能。

(4) 骨干区：OSPF 网络中的骨干区（Backbone Area）是一个特殊的区域，所有其他区域都连接到骨干区。骨干区形成了整个 OSPF 网络的核心。

(5) Hello 协议：OSPF 使用 Hello 协议来发现相邻路由器、建立和维护邻接关系。Hello 消息包含了路由器的 ID、优先级、邻接路由器的信息等。

(6) 链路状态更新：路由器通过定期发送链路状态更新（LSU，Link State Update）来通告自己所知道的链路状态信息。这些更新包含了连接关系、链路状态和度量值等信息。

(7) SPF 算法：OSPF 使用 SPF（Shortest Path First）算法来计算最短路径树，确定到达网络中各目标的最佳路径。这个算法基于 Dijkstra 算法。

(8) 区域边界路由器： 边缘路由器（Area Border Router，ABR）连接不同的区域，负责在区域之间传递路由信息。ABR 通常连接到骨干区。

2. 设计步骤

(1) 初始化网络拓扑：创建节点类表示路由节点，每个节点保存邻接节点及其距离信息。初始化网络拓扑，设置起点 A、B、C、D、E、F，并初始化节点间距离。

(2) 泛洪发布链路信息：每个节点生成链路状态包，包含本节点的链路信息。节点将链路状态包通过泛洪方式发布到整个网络。

(3) 接收并构造网络拓扑：节点接收其他节点的链路状态包，根据接收到的信息构造网络拓扑图。

(4) Dijkstra 算法计算最短路径：对于每个节点，利用 Dijkstra 算法计算到达其他节点的最短路径。

(5) 生成路由表：根据最短路径结果，每个节点生成自己的路由表，记录到达其他节点的下一跳及距离信息。

(6) 输出结果：打印每个节点的路由表，验证最短路径计算是否正确。

三、详细设计

1. 流程图

2. 关键代码

```
void Dijkstra(vector<vector<int>>& inf, int fir, int& n) {//求解 fir 点到其他各点的最短路径
    unordered_set<int> result;//加入到结果集的点
    vector<pair<int, int>> path(n, make_pair(fir, INT_MAX / 2));//路径数组
    int cur = fir;//当前正寻找最短路径的点 fir -> cur
    path[cur] = make_pair(fir, 0);//到自己的距离为0
    result.insert(cur);

    for (int i = 1; i < n; i++) {//剩下的n - 1 个点都需要得出最短路径(第一个点不需要寻找)
     //选择当前状态下最短路径的点作为拓展节点
     for (int j = 0; j < n; j++) {
      if (inf[cur][j] != 0 && result.find(j) == result.end()) {
       //cur 与 j 存在路径 且 该点还未找到最优路径 : 不在结果集中 此时视情况更新j 的路径距离
       int tmp = inf[cur][j] + path[cur].second;
       if (tmp < path[j].second) {
        path[j].second = tmp;
        path[j].first = cur;
       }
      }
     }

     int minimum = INT_MAX;
     int next = -1;//用于寻找下一次的拓展节点
     for (int k = 0; k < n; k++) {
      if (result.find(k) == result.end()) {
       if (path[k].second < minimum) {
        minimum = path[k].second;
        next = k;
       }
      }
     }
     cur = next;
     result.insert(cur);
    }
    findPath(path, inf, fir, n);
   }
```

四、实验结果与分析

1. 实验结果

起点	终点	下一跳	距离	起点	终点	下一跳	距离
A	B	B	4	B	A	A	4
	C	B	4		C	C	2
	D	B	4		D	C	2
	E	E	5		E	C	2
	F	B	4		F	F	6
起点	终点	下一跳	距离	起点	终点	下一跳	距离
C	A	E	1	D	A	C	3
	B	B	2		B	C	3
	D	D	3		C	C	3
	E	E	1		E	C	3
	F	B	2		F	F	7
起点	终点	下一跳	距离	起点	终点	下一跳	距离
E	A	A	5	F	A	B	6
	B	C	1		B	B	6
	C	C	1		C	B	6
	D	C	1		D	D	7
	F	F	8		E	E	8

2. 实验分析

在实验中，通过设计简单的 OSPF 路由协议原型系统，成功模拟了链路状态信息的泛洪发布、接收、网络拓扑构建和最短路径计算过程。每个节点生成链路状态包，通过泛洪传播至整个网络，节点接收并构造网络拓扑。利用 Dijkstra 算法计算最短路径，每个节点生成路由表。实验结果表明，最短路径计算正确，路由表包含了到达其他节点的最优路径。该实验帮助深入理解 OSPF 协议的工作原理，特别是链路状态数据库和最短路径计算。通过分析路由表，验证了网络拓扑的正确性，为进一步优化和改进路由协议提供了基础。

五、小结与心得体会

通过设计和实现简单的 OSPF 路由协议原型系统，我对 OSPF 协议的工作原理有了更深入的理解。首先，成功模拟了链路状态信息的传播过程，验证了泛洪发布和接收的有效性。构建网络拓扑的过程中，节点间正确地建立了邻接关系，形成了准确的拓扑图。Dijkstra 算法的应用使得每个节点都能计算出到达其他节点的最短路径，生成了准确的路由表。在实验中，我学到了如何使用 C++语言实现基本的路由协议模型，包括节点类的设计、泛洪传播的实现以及 Dijkstra 算法的应用。这种实践有助于将理论知识转化为具体的编程能力。此外，通过观察和分析实验结果，我更清晰地理解了 OSPF 协议中链路状态数据库的作用，以及最短路径计算的原理。实验中的一些挑战包括正确处理链路状态信息的传递和在网络拓扑构建过程中的边界条件处理。通过解决这些问题，我更好地理解了路由协议的一致性和稳定性的重要性。

总体而言，这个实验为我提供了一个深入学习和理解路由协议工作原理的机会。通过亲自实践，我不仅更加熟悉了 OSPF 协议的细节，而且培养了解决实际问题的能力。

实验五、基于 IP 多播的网络会议程序

一、实验目的

参照附录 3 的局域网 IP 多播程序，设计一个网络会议程序（实现文本多播方式即可）。

二、总体设计

1. 背景知识

(1) IP 多播：IP 多播是一种网络通信模式，允许一个数据包同时传输到多个接收者，而不是点对点的单播。这减少了网络流量和资源的使用，特别适用于群播应用，如音视频会议。

(2) 组播组：多播通信中，接收者被组织成一个组播组。发送者将数据发送到这个组，组内的所有成员都可以接收到相同的数据。在会议应用中，每个会议可能形成一个独立的组。

(3) RTP 和 RTCP 协议：实时传输协议（RTP）用于在 IP 网络上传输音视频数据。RTP 控制协议（RTCP）则用于监控和控制 RTP 流。它们通常与 IP 多播一起使用，以实现实时会议的音视频传输。

(4) 网络拓扑：会议程序需要适应不同的网络拓扑结构，包括局域网、广域网和云环境。了解不同网络拓扑对多播效率和传输质量的影响至关重要。

2. 设计步骤

■ 发送端

(1) 加载套接字，创建套接字库；

(2) 创建一个 SOCK_DGRAM 类型的用于接收的套接字 socket。WSASocket()创建一个与指定服务器提供者捆绑的套接口，可选地创建和/或加入一个套接口组。将参数设置为 WSA_FLAG_MULTIPOINT_C_LEAF、WSA_FLAG_MULTIPOINT_D_LEAF 和 WSA_FLAG_ OVERLAPPED 的位和，指明 IP 多播通信在控制层面和数据层面都是“无根的”，只存在叶节点，它们可以任意加入一个多播组，而且从一个叶节点发送的数据会传送到每一个叶节点（包括它自己）。

(3) 加入多播组。sockM 是用于发送的套接字，WSAJoinLeaf()仅仅将 sock 用于加入多播组，一个组是用多播地址确定的，remote 是将与 sock 连接的远端名字，JL_BOTH 用于指定 sock 既为发送者，又为接收者。

(4) 发送多播数据，当用户在控制台输入"QUIT"时退出；sendto()将数据由指定的 sockM 发送给对方主机，remote 为指向多播组套接口的地址。

■ 接收端

(1) 加载套接字，创建套接字库；

(2) 创建一个 SOCK_DGRAM 类型的用于接收的套接字 socket；

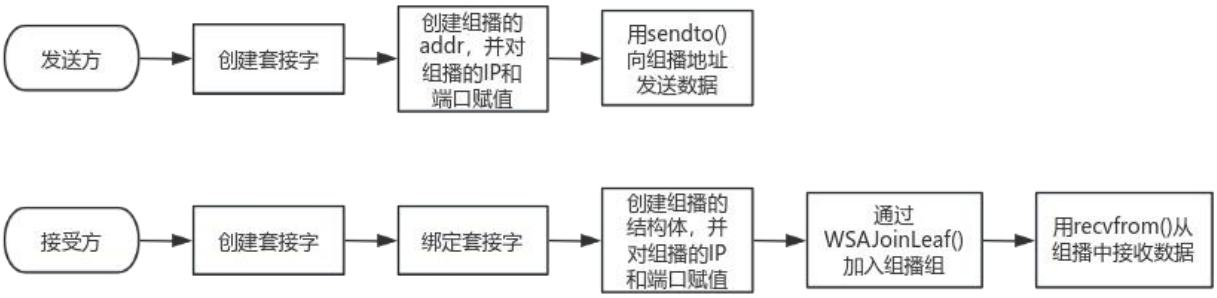
(3) 将此 socket 绑定到本地的一个端口上，为了接收服务器端发送的多播数据；bind()绑定套接字 sock 到一个 IP 地址和一个端口上，local 是一个指向 sockaddr 结构体的指针，赋予套接字地址。

(4) 加入多播组。sockM 是用于发送的套接字，WSAJoinLeaf()仅仅将 sock 用于加入多播组，一个组是用多播地址确定的，remote 是将与 sock 连接的远端名字，JL_BOTH 用于指定 sock 既为发送者，又为接收者。

(5) 接收多播数据，当接收到的数据为"QUIT"时退出。recvfrom()用于从已连接的套接口上接收数据，并捕获数据发送源的地址。

三、详细设计

1. 流程图



2. 关键代码

(1) 发送方

```
1.      while(1)
2.      {
3.       if(begin){
4.        //一开始通知接收方（主持人）自己上线了
5.        strcpy(sendbuf,HELLO);
6.       }else{
7.        printf("SEND : ");
8.        scanf("%s",sendbuf);
9.       }
10.
11.      //sendto()将数据由指定的 sockM 发送给对方主机
12.      //remote 为指向多播组套接口的地址
13.      //如果发送出现错误
14.      if( sendto(sockM,(char*)sendbuf,strlen(sendbuf),0,(struct socka
    ddr*)&remote,sizeof(remote))==SOCKET_ERROR)
15.      {
16.       printf("sendto failed with: %d\n",WSAGetLastError());
17.       closesocket(sockM);//关闭套接字 sockM
18.       closesocket(sock);//关闭套接字 sock
19.       WSACleanup();//释放资源
20.       return -1;
21.      }
22.      if(strcmp(sendbuf,"QUIT")==0){//如果发送 QUIT，则发送方和接收方都退
    出
```

```
        printf("INFO: 您已将会议结束！\n");
        break;
       }else if(strcmp(sendbuf,BYE)==0){
        printf("INFO: 您已退出会议！\n");
        break;
       }
       begin=false;
       Sleep(500);//休眠 500 毫秒
      }
```

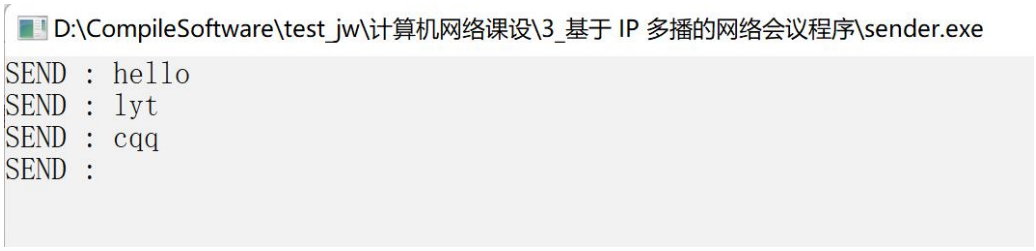
(2) 接收方

```
      while (1) {
       //recvfrom()用于从已连接的套接口上接收数据，并捕获数据发送源的地址
       //recvbuf 表示接收数据缓冲区
       //from 指向装有源地址的缓冲区，len 指向 from 缓冲区长度值
       //如果接收发生错误
       if (( ret = recvfrom(sock, recvbuf, BUFSIZE, 0, (struct sockaddr*)&from, &len)) == SOCKET_ERROR) {
        printf("recvfrom failed with:%d\n", WSAGetLastError()); //WSAGetLastError()获取相应的错误代码
        closesocket(sockM);//关闭套接字 sockM
        closesocket(sock);//关闭套接字 sock
        WSACleanup();//释放资源
        return -1;
       }
       if ( strcmp(recvbuf, "QUIT") == 0 ) { //如果接收到 QUIT，则退出
        printf("INFO: 会议被 <%s> 结束！\n", inet_ntoa(from.sin_addr));
        break;
       }
       else if ( strcmp(recvbuf, HELLO) == 0 ) { //如果接收到 HELLO，则
        printf("INFO: <%s> 加入会议。\n", inet_ntoa(from.sin_addr));
       }
       else if ( strcmp(recvbuf, BYE) == 0 ) { //
        printf("INFO: <%s> 退出会议。\n", inet_ntoa(from.sin_addr));
       }
       else {
        //输出收到的消息及发送方的 IP 地址
        recvbuf[ret] = '\0';
        printf("RECV:' %s ' FROM <%s> \n", recvbuf, inet_ntoa(from.sin_addr));
       }
       memset(recvbuf,0,sizeof(recvbuf));
      }
```

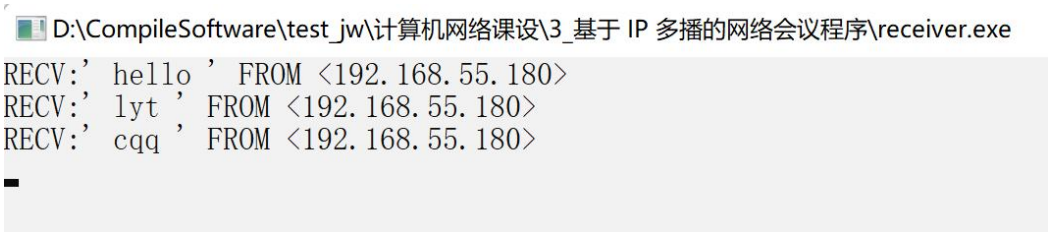
四、实验结果与分析

1. 实验结果

(1) 发送方



(2) 接收方



2. 实验分析

(1) 该实验需要处在局域网下的两台电脑或使用虚拟机来代替，并且两台设备都必须使用 windows 系统（api 使用的是 win 环境下的）。

(2) 服务器启动后会持续检测是否有连接请求，当监听到请求当前连接到的 Client 的 IP 及端口，若接收不到消息需关闭发送方和接收方防火墙；

(3) 注意发送和接收方不能是同台机器，因为同一端口号不能同时分配两个进程；

(4) 可以用 netsh interface ipv4 show joins 命令查看所用网卡有哪些多播地址，通信双方都要是同样的多播地址。

五、小结与心得体会

通过实现基于 IP 多播的网络会议程序，我深刻体会到 IP 多播在信息传输中的重要性。该程序成功实现了多用户之间的实时信息传递，提供了高效的群体通信解决方案。IP 多播技术有效减少了网络流量，提高了信息传输的效率。在开发过程中，我加深了对网络通信原理的理解，并学会了一种局域网中主机间相互交换数据的方法，同时可以利用 IP 多播可以建立一个多人聊天室。这次经历让我更加熟悉了网络编程和实时通信的实际应用，为我的技术能力的提升提供了宝贵经验。这个项目的成功实现让我对信息传输中的网络优化有了更深刻的认识。

实验六、编程模拟 NAT 网络地址转换

一、实验目的

参考教材 188 页内容，模拟 NAT 路由器的工作过程，主要有 2 个步骤的工作：1、将收到的来自内网报文中的私有源 IP 地址转换为 NAT 的外部合法 IP 地址，同时将传输层源端口号转换为 NAT 路由器分配的端口号，建立转换映射表；2、将收到的来自外网的应答报文提取其目的 IP 地址及端口号，查找映射表，找到其对应的内网机器的 IP 地址及端口号并替换。转换表如下图所示：

方向	字段	旧的IP地址和端口号	新的IP地址和端口号
出	源IP地址:TCP源端口	192.168.0.3:30000	172.38.1.5:40001
出	源IP地址:TCP源端口	192.168.0.4:30000	172.38.1.5:40002
入	目的IP地址:TCP目的端口	172.38.1.5:40001	192.168.0.3:30000
入	目的IP地址:TCP目的端口	172.38.1.5:40002	192.168.0.4:30000

二、总体设计

1. 背景知识

(1) IP 地址分为私有和公有：IPv4 地址被分为私有地址和公有地址。私有地址用于内部网络，例如 10.0.0.0/8、172.16.0.0/12 和 192.168.0.0/16。公有地址用于互联网。

(2) NAT 的工作原理：NAT 在路由器或防火墙上实现。当内部设备尝试访问互联网时，NAT 会将私有 IP 地址转换为公有 IP 地址，并在路由表中维护转换信息。返回流量时，NAT 将公有 IP 地址转换回相应的私有 IP 地址。

(3) NAT 类型：有三种主要的 NAT 类型：静态 NAT、动态 NAT 和 PAT（Port Address Translation）。静态 NAT 将内部 IP 地址映射到固定的公共 IP 地址，动态 NAT 动态地分配公共 IP 地址，而 PAT 通过使用端口号来区分多个内部设备。

(4) 端口：PAT 使用端口号来区分多个内部设备共享同一个公共 IP 地址。这使得多个设备可以同时使用相同的公共 IP 地址，而不发生冲突。

(5) NAT 的优势：NAT 提供了一种有效的方式来延长 IPv4 地址的使用寿命，减缓地址短缺问题。此外，NAT 还提供了一定程度的网络安全性，因为内部设备的实际 IP 地址对外部网络是不可见的。

2. 设计步骤

(1) 设置网络拓扑：创建一个包含内网和外网的网络拓扑。内网包括两个主机（192.168.0.3 和 192.168.0.4），外网包括 NAT 路由器。

(2) 配置 NAT 路由器：在 NAT 路由器上设置转换表，包括出方向的映射（内网到外网）和入方向的映射（外网到内网）。

(3) 生成报文：在内网主机上生成 TCP 报文，其中包含私有源 IP 地址（192.168.0.3 或 192.168.0.4）和源端口号（30000）。报文经过 NAT 路由器时，NAT 路由器执行转换操作。

(4) 进行转换：NAT 路由器接收内网报文，将私有源 IP 地址和源端口号转换为

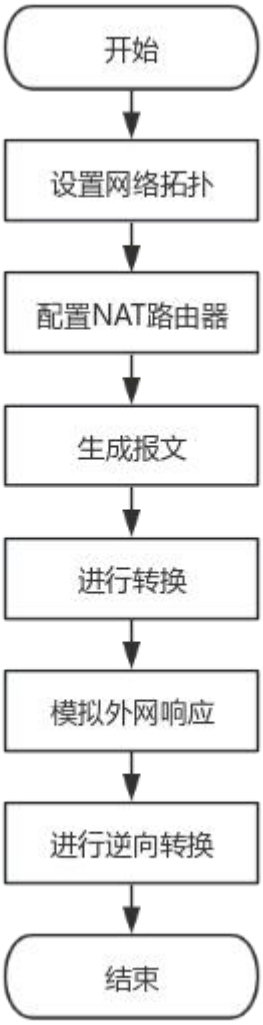
NAT 的外部合法 IP 地址和 NAT 路由器分配的端口号。同时建立转换映射表。

(5) 模拟外网响应：在外网模拟响应报文，其中包含目的 IP 地址（NAT 路由器的外部合法 IP 地址）和目的端口号（NAT 路由器分配的端口号）。报文经过 NAT 路由器时，NAT 路由器执行转换操作。

(6) 进行逆向转换：NAT 路由器接收外网响应报文，查找映射表，找到其对应的内网机器的 IP 地址和端口号，并进行逆向转换。

三、详细设计

1. 流程图



2. 关键代码

```
        // 处理从内网到外网的报文，进行地址转换
        void translateOutgoing(string& sourceIpPort) {
            // 如果映射表中没有该条目，创建新的映射并输出转换信息
            if (outgoingMappings.find(sourceIpPort) == outgoingMappings.end()) {

                // 在内网到外网的映射表中存储相应的数据
```

```
                string newIpPort = assignNewIpPort();
                MappingEntry entry_in = {sourceIpPort, newIpPort};
                outgoingMappings[sourceIpPort] = entry_in;

                // 在外网到内网的映射表中存储相应的数据
                MappingEntry entry_out = {newIpPort, sourceIpPort};
                incomingMappings[newIpPort] = entry_out;

                cout << "出向映射: " << sourceIpPort << " -> " << newIpPort << endl;
            }

            // 执行地址转换
            sourceIpPort = outgoingMappings[sourceIpPort].newIpPort;
        }

        // 处理从外网到内网的报文，进行地址转换还原
        void translateIncoming(string& destinationIpPort) {
            // 如果映射表中没有该条目，输出错误信息
            if (incomingMappings.find(destinationIpPort) == incomingMappings.end()) {
                cerr << "错误：未找到入向映射表中的条目。" << endl;
                return;
            }

     cout << "入向映射: " << destinationIpPort << " -> " << incomingMappings[destinationIpPort].newIpPort << endl;
            // 执行地址还原
            destinationIpPort = incomingMappings[destinationIpPort].newIpPort;

            // 可选：从映射表中删除该条目（NAT 具有状态的行为）
            incomingMappings.erase(destinationIpPort);
        }
        // 分配新的内网地址和端口
        string assignNewIpPort() {
            // 这是一个简化的例子；在实际场景中需要更加谨慎地管理 IP 和端口的分配
            static int portCounter = 40000;
            return "172.38.1.5:" + to_string(++portCounter);
        }
```

四、实验结果与分析

1. 实验结果

```
请输入出向IP地址（输入"quit"停止）：
> 192.168.0.3:30000
出向映射：192.168.0.3:30000 -> 172.38.1.5:40001
转换后的出向报文：172.38.1.5:40001
> 192.168.0.4:30000
出向映射：192.168.0.4:30000 -> 172.38.1.5:40002
转换后的出向报文：172.38.1.5:40002
> quit
输入结束。

请输入入向IP地址（输入"quit"停止）：
> 172.38.1.5:40001
入向映射：172.38.1.5:40001 -> 192.168.0.3:30000
转换后的入向报文：192.168.0.3:30000
> 172.38.1.5:40002
入向映射：172.38.1.5:40002 -> 192.168.0.4:30000
转换后的入向报文：192.168.0.4:30000
> 172.38.1.5:40003
错误：未找到入向映射表中的条目。
转换后的入向报文：172.38.1.5:40003
> quit
输入结束。
```

2. 实验分析

该实验模拟 NAT 路由器的工作过程，通过设计网络拓扑、配置 NAT 路由器和生成报文，验证 NAT 在内外网间的地址和端口号转换。从内网发起 TCP 报文到 NAT 路由器，经过转换映射表后，抵达外网。模拟外网响应后，NAT 路由器进行逆向转换将响应传递给正确的内网主机。通过实际操作，能够深入了解 NAT 的工作原理，巩固网络地址转换的概念。实验结果可通过观察报文和转换表的变化来验证 NAT 路由器的正确性。

五、小结与心得体会

在完成这个 NAT 网络地址转换的实验过程中，我深刻认识到 NAT 在网络通信中的重要作用。通过搭建网络拓扑、配置 NAT 路由器，以及生成报文的实际操作，我亲身体验了 NAT 如何将内网的私有 IP 地址和端口号映射为外部合法 IP 地址和端口号的过程。通过手动创建转换表，我更加清晰地理解了 NAT 路由器是如何维护连接状态的。实验中的模拟外网响应和逆向转换过程让我更好地理解了 NAT 路由器如何处理外部响应，并将响应准确传递给内网主机。这不仅提高了我的网络知识水平，还培养了我对网络配置和协议工作原理的实际操作能力。

总的来说，通过这个实验，我不仅对 NAT 的基本原理有了更深刻的理解，而且锻炼了网络配置和故障排除的实际技能。这个实验为我提供了一个宝贵的机会，让我在学术知识和实际操作中取得平衡，对我的专业发展产生了积极的影响。

实验九、简单 Web Server 程序的设计与实现

一、实验目的

编写简单的 Web Server 有助于读者了解 Web Server 的工作流程，掌握超文本传送协议(HTTP)基本原理，掌握 Windows 环境中用 socket 实现 C/S 结构程序的编程方法。附录 7 介绍了一个简单 Web Server 的程序设计过程。

二、总体设计

1. 背景知识

(1) Web 浏览器（Web Browser）：是一个用于文档检索和显示的客户应用程序，并通过超文本传输协议 Http（Hyper Text Transfer Protocol）与 Web 服务器相连。通用的、低成本的浏览器节省了两层结构的 C/S 模式客户端软件的开发和维护费用。

(2) Web 服务是 Internet 最方便与受用户欢迎的服务类型，它的影响力也远远超出了专业技术范畴，已广泛应用于电子商务、远程教育、远程医疗与信息服务等领域，并且有继续扩大的趋势。目前很多的 Internet 应用都是基于 Web 技术的，因此掌握 Web 环境的软件编程技术对软件人员是至关重要的。

(3) TCP/IP 协议栈：Web 服务器基于 TCP/IP 协议栈，通过 TCP 连接处理 HTTP 请求和响应。了解 TCP/IP 协议栈的基本原理对于理解 Web 服务器的工作方式至关重要。

(4) Socket 编程：在实现 Web 服务器时，使用 Socket 编程来建立和管理 TCP 连接。Socket 是一种用于网络通信的编程接口，通过套接字（socket）进行数据传输。

(5) 多线程/多进程编程：为了同时处理多个客户端请求，许多 Web 服务器采用多线程或多进程的并发模型。这可以提高服务器的性能和响应能力。

(6) URI 和 URL：统一资源标识符（URI）和统一资源定位符（URL）是 Web 中标识和定位资源的标准。Web 服务器需要解析客户端请求中的 URL，并定位到相应的资源。

(7) HTTP 连接方式

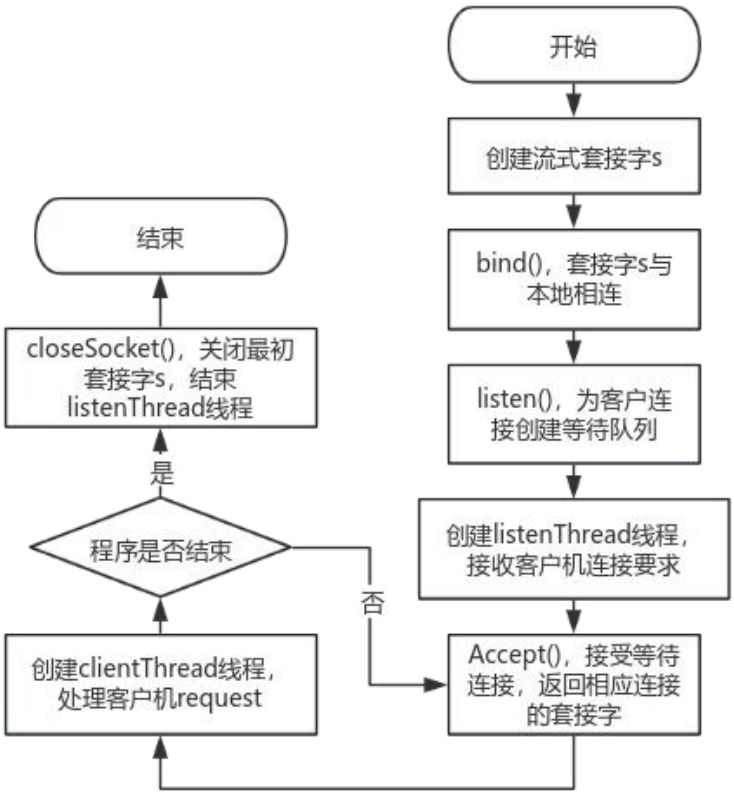
- 短连接：每次 HTTP 通信，都需要建立一个独立的 TCP 连接，HTTP1.0 默认使用短连接，请求头为 Connection：close。
- 长连接：非流水线方式只需要建立一次 TCP 连接就能进行多次 HTTP 通信。流水线方式将多个 HTTP 请求整批提交的技术，在传送过程中不需要先等待服务端的回应。HTTP1.1 默认使用长连接，会在请求头加上 Connection：keep-alive。

2. 设计步骤

说明：Web Server 可以分为两个组成模块，客户请求处理模块和响应生成发送模块。其中客户请求处理模块的任务就是负责接收客户的连接，它监听系统的端口，以获取客户机到达本服务器的连接请求信息并分析请求中的各个协议参数。响应生成发送模块根据客户请求的分析结果查找资源，生成响应和发送响应信息。

三、详细设计

1. 流程图



2. 关键代码

```
1.    while (true) {
2.      printf("\nListening ... \n");
3.      sockaddr_in addrClient;
4.      int nClientAddrLen = sizeof(addrClient);
5.      //服务器端建立连接
6.      SOCKET socketClient = accept(socketServer, (sockaddr*)&addrClie
   nt, &nClientAddrLen);
7.      if (SOCKET_ERROR == socketClient) {
8.       printf("接收失败!");
9.       break;
10.     }
11.     char buffer[BUFFER_SIZE];
12.     memset(buffer, 0, BUFFER_SIZE);
13.     //接收数据
14.     if (recv(socketClient, buffer, BUFFER_SIZE, 0) < 0) {
15.      printf("接收数据失败!");
16.      break;
17.     }
```

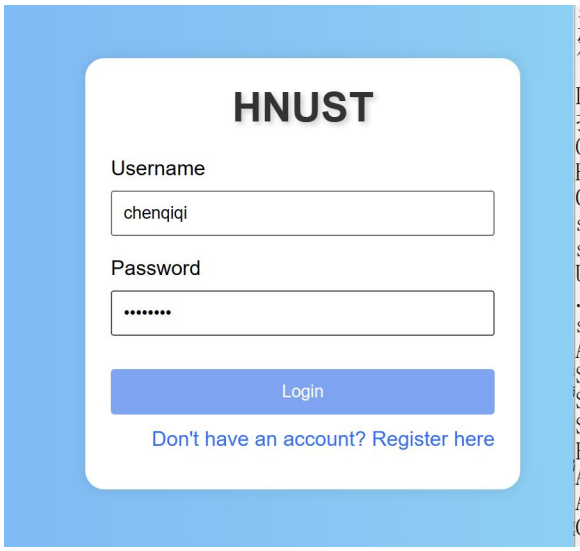
```
18.     printf("接收到的请求数据 : \n%s", buffer);
19.
20.     //获取请求的路径
21.     string path = getRequestUrl(buffer);
22.
23.     getRaletivePath(path);
24.
25.     //首页
26.     if (path.compare("") == 0) {
27.      path = index;
28.     }
29.     cout << "定位到要找的文件相对路径为: " << path;
30.
31.     if (FILE *file = fopen(path.c_str(), "r")) {
32.   //    该文件存在
33.      fclose(file);
34.      cout << " 该文件存在" << endl;
35.     } else {
36.   //    文件不存在
37.      path = error;
38.      cout << " 该文件不存在，返回错误页面" << endl;
39.     }
40.
41.     // response
42.     // send header
43.     memset(buffer, 0, BUFFER_SIZE);
44.
45.     //读取文本
46.     ifstream fin(path.c_str(), ios::in | ios::binary);
47.     fin.seekg(0, ios_base::end);//将读取位置设置为文件末尾
48.     streampos pos = fin.tellg();//返回当前文件位置
49.     long lSize = static_cast<long>(pos);//获取文件长度
50.     fin.seekg(0, ios_base::beg);//将读取位置设置为文件开头
51.     sprintf(buffer, HEADER, lSize);//把文件和头文件合并然后发送数据
52.
53.     if (fin.is_open()) {
54.      int result=-1;
55.      do{
56.       //第一次是发送响应头，之后发送响应体
57.       if ((result=send(socketClient, buffer, strlen(buffer), 0)) <
   0) {
58.        break;
59.       }
60.       memset(buffer, 0, BUFFER_SIZE);
```

```
61.         fin.read(buffer, sizeof(buffer));
62.        }while(fin.gcount());//获取上一次 read 读取了多少字节
63.
64.        if (result < 0) {
65.         printf("发送数据失败!");
66.         fin.close();
67.         closesocket(socketClient);
68.         break;
69.        }
70.      }
71.      //printf("发送!");
72.      //关于网络连接与数据传递
73.      fin.close();
74.      closesocket(socketClient);
75.     }
```

四、实验结果与分析

1. 实验结果

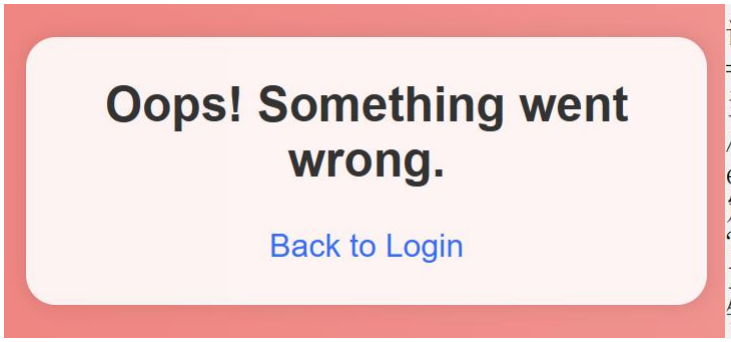
访问存在的页面，即 login.html




```
当前请求的绝对路径为：http://127.0.0.1:8000/。
定位到要找的文件相对路径为：login.html 该文件存在

Listening ...
接收到的请求数据 ：
GET /favicon.ico HTTP/1.1
Host: 127.0.0.1:8000
Connection: keep-alive
sec-ch-ua: "Not_A Brand";v="8", "Chromium";v="120",
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x
.36 Edg/120.0.0.0
sec-ch-ua-platform: "Windows"
Accept: image/webp,image/apng,image/svg+xml,image/
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: http://127.0.0.1:8000/
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.
Cookie: _xmLog=h5&1521ed00-289b-4e12-8ad5-f352485f
```

访问不存在的页面，返回 error.html




```
请求的相对路径为：/?username=chenqiqi&password=12345678
当前请求的绝对路径为：http://127.0.0.1:8000/?username=chenqiqi&password=12345678。
定位到要找的文件相对路径为：?username=chenqiqi&password=12345678 该文件不存在，返回错误页面
```

2. 实验分析

首先指定 web 服务器的端口，之后服务器持续监听这个端口的 http 请求，通过浏览器中访问的资源路径，服务器给出相应的响应。端点和资源映射关系如下：如果访问/或者/index.html 则服务器寻找其目录下的 index.html 作为主页，否则以当前服务器 exe 下的路径为基础，按照相对路径格式寻找资源并读取，返回数据，如果没有找到对应资源，则默认返回其目录下的 error.html 作为错误信息。

五、小结与心得体会

首先，Web 服务器使用了 WinSock 库进行 Socket 编程，实现了基本的服务器套接字的初始化、绑定、监听以及接受连接。通过 Socket，服务器能够在指定端口监听客户端请求，建立连接并接收数据。其次，服务器实现了对 HTTP 请求的基本解析，通过解析请求头中的 URL，获取客户端请求的相对路径。这一步是实现简单 Web 服务器的基础，对 HTTP 协议的理解和处理是至关重要的。在接收到请求后，服务器根据请求的相对路径定位到相应的资源文件。如果文件存在，则发送 HTTP 响应头，并将文件内容作为响应体发送给客户端。这里使用了 HTTP 的标准响应头格式，包括状态行、内容类型、服务器信息和内容长度等。值得注意的是，服务器通过文件操作和流处理的方式，支持了读取并发送大文件，提高了程序的健壮性和可扩展性。最后，通过循环监听，服务器能够持续接受并处理客户端的连接请求，实现了基本的多客户端处理能力。这是 Web 服务器的常见特性，提高了服务器的并发性能。

总的来说，实现了一个简单而有效的 Web 服务器，尽管它仅支持静态文件的传输，但作为基础框架，为理解 Web 服务器的基本原理和实现奠定了良好的基础。进一步的扩展可以包括支持动态内容、HTTP 方法的扩展等，以满足更复杂的 Web 应用需求。学习并实现简单 Web 服务器有助于深入理解网络编程和 HTTP 协议，提升编程技能。这能为我日后开发 Web 应用或与网络相关的项目打下坚实基础。通过此实践，我更熟悉服务器端开发，提高对网络通信的把握。

实验十、路由器查表过程模拟

一、实验目的

参考教材 140 页 4.3 节内容，编程模拟路由器查找路由表的过程，用（目的地址 掩码 下一跳） 的 IP 路由表以及目的地址作为输入，为目的地址查找路由表，找出正确的下一跳并输出结果。

说明：本次实验使用的路由器转发表为课本P_{203}题目 4-18，以下为该转发表。

前缀匹配	下一跳
192.4.153.0/26	R3
128.96.39.0/25	接口 m0
128.96.39.128/25	接口 m1
128.96.40.0/25	R2
*（默认）	R4

二、总体设计

1. 背景知识

(1) IP 地址：IP 地址是计算机在网络中的唯一标识符，分为 IPv4 和 IPv6。在路由器查表过程中，目的地址通常是 IPv4 地址，例如，192.168.1.1。

(2) 子网掩码：子网掩码用于将 IP 地址划分为网络地址和主机地址两部分。在路由表中，规则通常包含目的地址和对应的子网掩码，以确定匹配的范围。

(3) 路由表：路由表是路由器存储的数据结构，其中包含了路由规则。每个规则由目的地址、子网掩码和下一跳信息组成。路由器在接收到数据包时，会查询路由表，找到匹配的规则，从而决定下一跳的路径。

(4) 下一跳：下一跳表示数据包应该发送到的下一个路由器或目的地。在路由表中，下一跳信息指示了数据包应该经过的路径。

(5) 最长前缀匹配：路由器通常采用最长前缀匹配的策略，即在路由表中查找与目的地址最匹配的规则。这确保了路由器选择最精确的路径进行数据包转发。

(6) 静态路由 vs. 动态路由：路由表的信息可以通过静态配置手动输入，也可以通过动态路由协议（如 OSPF、BGP）自动学习更新。了解静态路由和动态路由的差异对于理解路由表的构建和维护至关重要。

(7) 路由表更新策略：动态路由协议通常采用一定更新策略，例如定期更新、事件触发更新等。了解这些更新策略有助于网络管理员合理规划和维护路由表。

(8) 网络分段和 VLSM：网络的分段和可变长度子网掩码是网络设计中的重要概念，影响着路由表的结构和管理方式。

2. 设计步骤

(1) 准备 IP 路由表：创建一个包含多条规则的 IP 路由表。

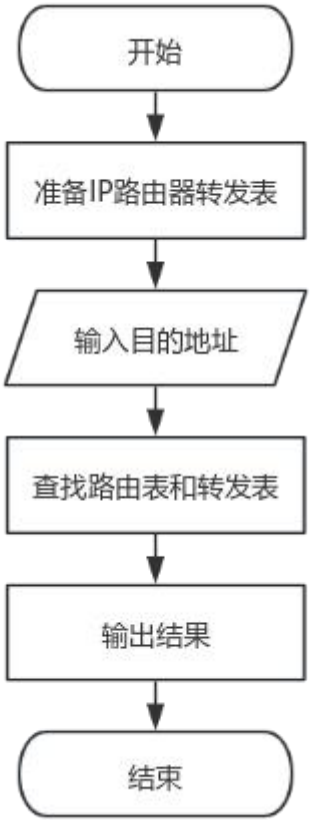
(2) 选择目的地址：输入一个目的地址，例如，192.168.1.10。

(3) 查找路由表：从路由表中逐条匹配目的地址和掩码，找到匹配的规则。

(4) 输出结果：输出匹配规则的下一跳信息，表示数据包应该通过该下一跳路由器进行转发。

三、详细设计

1. 流程图



2. 关键代码

```
bitset<32> convertToBinary(const string& ipAddress) {
 istringstream iss(ipAddress);
    vector<string> octets;

 // 将 ip 地址根据"."进行拆分并转换为 32 位二进制数
    string octet;
    while (getline(iss, octet, '.')) {
        octets.push_back(octet);
    }

    // 用于存储拼接后的二进制数
    bitset<32> combinedBinary;

    // 将每个部分转换为 8 位二进制数并拼接
    for (const auto& part : octets) {
        bitset<32> binaryPart(stoi(part));
        combinedBinary <<= 8;  // 左移 8 位
        combinedBinary |= binaryPart;
```

```
        }

        return combinedBinary;
    }

    string findNextHop(const string& destAddress) {
     // 先将给出的目的地址转换为32位二进制数
        bitset<32> destAddrBits = convertToBinary(destAddress);

     // 通过遍历路由表，查找其下一跳；路由表中存放的数据要先根据"/"进行划分，将前半部分转换为32位二进制数，后半部分则将转换后的数再一步划分
     // 然后判断这两个数是否相等，相等则取对应的下一跳
        for (const auto& entry : ipRoutingTable) {
      // 查找第一个"/"
         size_t pos = entry.first.find('/');
         string routingEntry = entry.first.substr(0, pos);
         int mask = stoi(entry.first.substr(pos + 1));

           bitset<32> routingEntryBits = convertToBinary(routingEntry);

           if (destAddrBits.to_string().substr(0, mask) == routingEntryBits.to_string().substr(0, mask)) {
            return entry.second;
      }
        }

        // 默认项
     return "R4";

    }
```

四、实验结果与分析

1. 实验结果

```
请输入目的地址（输入"quit"停止）：
> 192.168.0.1
下一跳：R4
> 128.96.39.127
下一跳：接口m0
> 128.96.39.129
下一跳：接口m1
> 128.96.40.1
下一跳：R2
> 192.4.153.3
下一跳：R3
> quit
输入结束。
```

2. 实验分析

在模拟路由器查表的实验中，成功地展示了路由器如何根据 IP 路由表智能地选择下一跳路径。通过使用简单的 IP 路由表，模拟了目的地址查找过程，找到了正确的下一跳信息。实验通过逐步执行目的地址匹配、查询路由表和输出下一跳的步骤，清晰地呈现了路由器查表过程。实验中的 IP 路由表展现了路由器对网络拓扑的智能感知，实现了最长前缀匹配策略，确保在复杂网络中的准确性。

五、小结与心得体会

通过模拟路由器查表的实验，我深刻理解了网络通信中路由器的关键角色和其查表过程的基本原理。实验中使用的 IP 路由表清晰地展示了目的地址匹配、下一跳选择等重要概念，使我对路由器如何智能地处理数据包转发有了更直观的认识。实验的成功进行强调了 IP 地址、子网掩码、路由表等基础网络概念在路由器工作中的重要性。最长前缀匹配策略的运用使路由器能够高效地决定数据包的下一跳路径，适应不同子网和网络拓扑结构。此外，了解了静态路由、动态路由、硬件加速等相关知识，对于深入网络管理和优化提供了更多思路。

总而言之，通过模拟路由器查表的实验，我深刻理解了 IP 路由表的核心原理，对路由器智能数据包转发过程有了清晰认识。实验中的目的地址匹配、最长前缀匹配等概念的应用使我更具网络管理和设计的实际应用能力。这次实践不仅巩固了我对网络通信基础知识的理解，也激发了我对网络工程实践的兴趣。通过手动模拟路由器的工作过程，我更加自信地迎接未来网络领域的挑战，深感实验的实际价值。