实验一

一、实验题目

题目一：网络聊天程序的设计与实现

二、实验目的

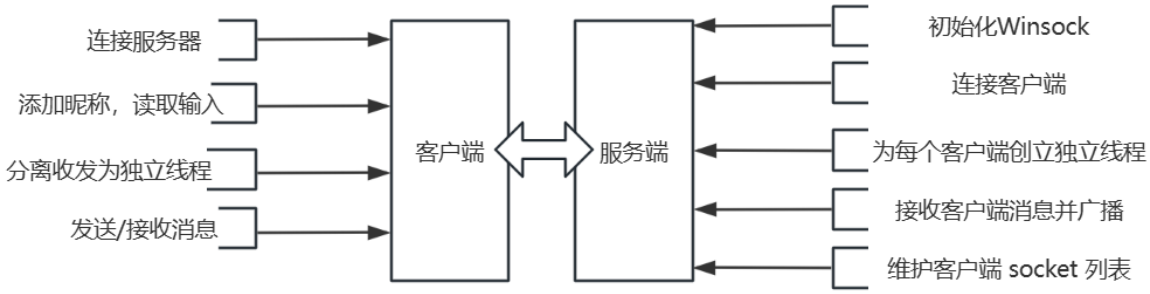
1.理解 TCP/IP 网络通信的基本原理，掌握 TCP 套接字（Socket）的使用方法。
2.熟悉 Windows 平台下 Winsock 网络编程接口及其基本调用流程。
3.掌握多线程编程的基本思想，理解并发环境下共享资源的同步与互斥问题。
4.通过设计并实现一个多用户聊天室，加深对客户端—服务器（C/S）模型的理解。
5.提升对网络应用层协议设计和程序整体架构设计能力的认识。

三、总体设计

1.背景知识

TCP 协议：面向连接的可靠传输协议，保证数据顺序和完整性
Winsock 编程：Windows 平台下的套接字编程接口
C/S 架构：客户端/服务器模式，服务器负责协调，客户端提供用户界面
多线程编程：使用独立线程处理不同任务，提高程序并发性

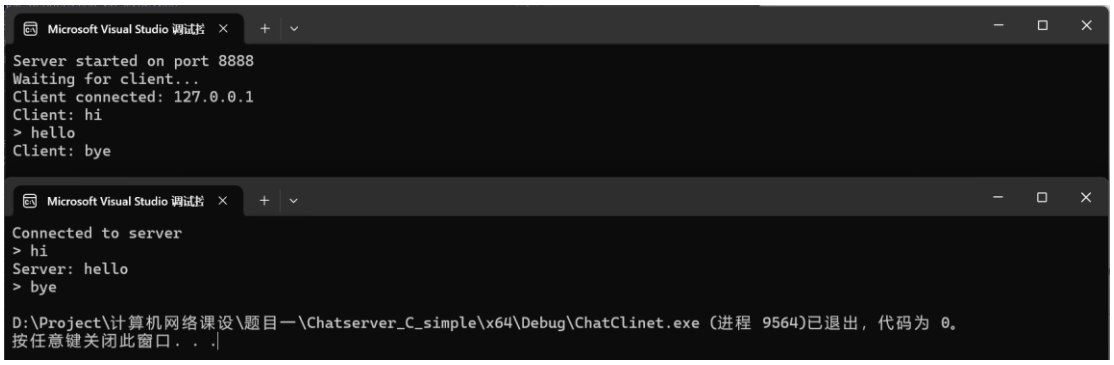
2.模块介绍



3.设计步骤

第一步：根据指导书中的示例代码，仅实现了客户端与服务端的通信，由客户端自动发送“hello”后结束程序，并未实现聊天功能

第二步：在此基础上，添加对话循环，实现双向通信，但仅为一问一答形式



第三步：实现双向实时通信，此时已初具聊天雏形，但仍有几个问题，如客户端和服务器在对方连续发消息时自己的界面会自动多创建一行聊天栏，以及此时仍为单线程聊天，未添加多线程

第四步：最后实现多线程聊天室

四、详细设计

1. 主要数据结构

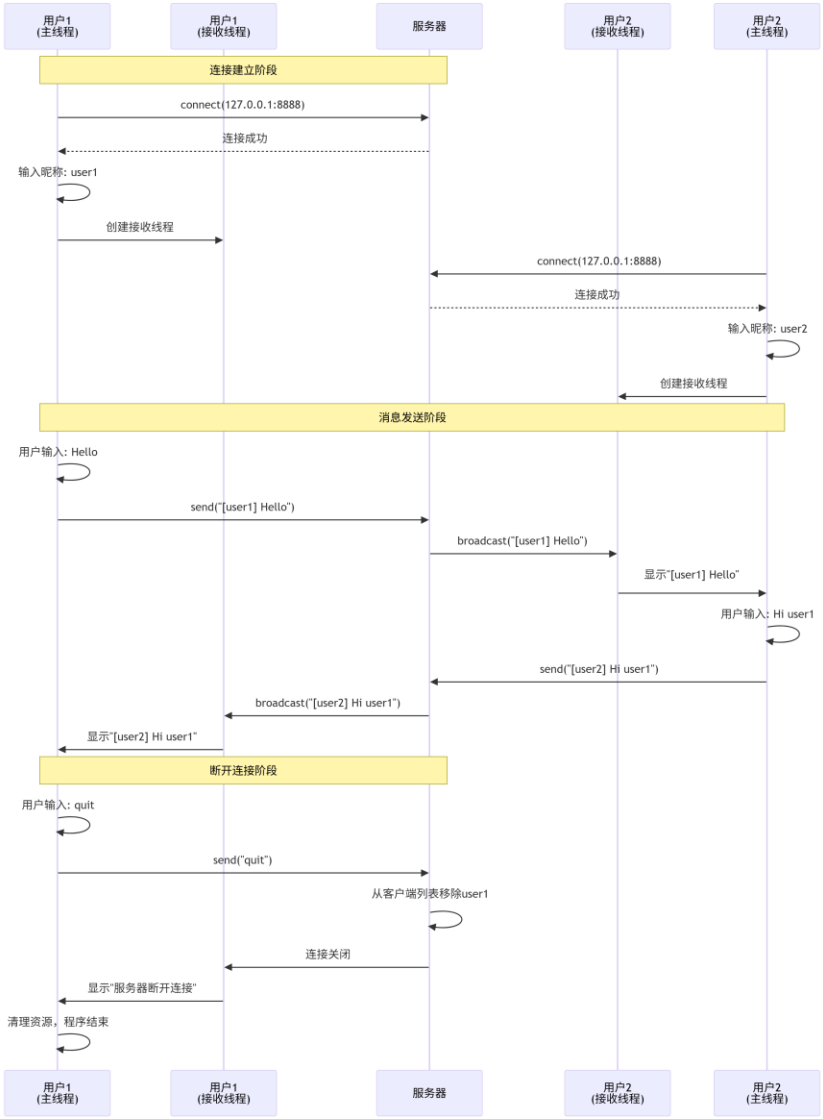
服务器：

```
SOCKET clients[MAX_CLIENTS]; // 保存所有客户端 socket
int client_count; // 当前在线客户端数量
CRITICAL_SECTION cs; // 临界区线程同步
```

客户端：

```
SOCKET sock;  //客户端 socket
char nickname[32]; // 用户昵称
```

2. 程序流程



五、实验结果与分析

1. 服务端启动正常，连接 8888 端口
2. 创建两个客户端：

   客户端显示连接正常

   服务端显示两个客户端连接

3. 分别输入客户端 1 和客户端 2 昵称“user1”与“user2”，以便后续聊天区分
4. “user1”与“user2”通话，此时服务端启用广播机制，客户端能够正常发送与接收消息，服务端显示全部消息

5.中途添加客户端 3，起名“user3”，服务端显示新客户端连接

6.“user3”输入，服务端与其他两个客户端均能正常接收显示消息

六、小结与心得体会

本次多用户聊天室实验的设计与实现过程中，程序经历了从“单客户端通信”到“支持昵称的多用户聊天室”的多次迭代。

首先根据指导书中的 demo 完成单线程通信，而后为解决“如何同时处理多个客户端连接”，服务器端引入了客户端 socket 数组，并采用“一个客户端对应一个线程”的方式处理并发通信，使服务器能够同时接收和转发来自多个客户端的消息。为防止多线程并发访问共享资源导致的数据竞争和程序异常，又进一步引入了临界区（CRITICAL_SECTION）机制，对客户端列表的增删操作进行同步控制。

本次实验，深入理解了基于 TCP 的网络通信流程以及多用户服务器的基本设计思想。体会到了多线程并发访问带来的问题，也加深了对同步与互斥机制重要性的认识。不仅巩固了 Socket 编程的基础知识，也让我对应用层协议设计、客户端与服务器职责划分有了更加清晰的理解。

实验二

一、实验题目

题目二：Tracert 与 Ping 程序设计与实现

二、实验目的

1.理解 Ping 与 Tracert 工具在网络连通性检测中的作用与原理。
2.掌握 ICMP 协议 的基本报文格式及工作机制。
3.学习在 Windows 平台下使用 Raw Socket 进行网络编程的方法。
4.通过程序设计，实现对局域网内主机在线状态的检测，加深对网络层协议的理解。
5.提高对网络调试工具底层实现原理的认识，培养网络程序的设计与调试能力。

三、总体设计

1.背景知识

- Ping 的工作原理
  Ping 通过向目标主机发送 ICMP Echo Request（回显请求） 报文，并等待对方返回 ICMP Echo Reply（回显应答） 报文，从而判断目标主机是否在线以及网络连通性情况。
- ICMP
  ICMP 协议用于在 IP 主机、路由器间传递控制消息。
- Raw Socket
  Raw Socket 是一种直接与网络层或数据链路层相连通信的套接字，因为 ICMP 不属于传输层，所以需要使用 Raw Socket 直接构造并发送报文

2.设计步骤

启动程序并初始化 Winsock

创建 Raw Socket（AF_INET，SOCK_RAW，IPPROTO_ICMP）

设置接收超时时间，防止程序阻塞

由用户输入局域网内任意 IP 地址

解析输入 IP，确定所在 C 类网段

对 1～254 范围内的 IP 地址逐个发送 Ping 请求

接收并解析 ICMP Echo Reply

判断每个 IP 是否在线，并实时输出扫描结果

扫描完成后释放资源并退出程序

四、详细设计

1.ICMP 报文结构体

```
typedef struct
{
    BYTE  type;
    BYTE  code;
    USHORT checksum;
    USHORT id;
    USHORT seq;
} ICMP_HEADER;
```

2.程序整体流程：

- 启动程序并初始化 Winsock
- 创建 Raw Socket（AF_INET，SOCK_RAW，IPPROTO_ICMP）
- 设置接收超时时间，防止程序阻塞
- 由用户输入局域网内任意 IP 地址
- 解析输入 IP，确定所在 C 类网段
- 对 1～254 范围内的 IP 地址逐个发送 Ping 请求
- 接收并解析 ICMP Echo Reply
- 判断每个 IP 是否在线，并实时输出扫描结果
- 扫描完成后释放资源并退出程序

3.关键代码

USHORT checksum(USHORT buf, int size) //ICMP 校验和计算*

该函数用于计算 ICMP 报文的校验和，保证数据在传输过程中的完整性。

int ping_host(SOCKET sock, const char ip) //Ping 单个主机函数*

完成以下功能：

构造 ICMP Echo Request 报文

发送请求到目标 IP

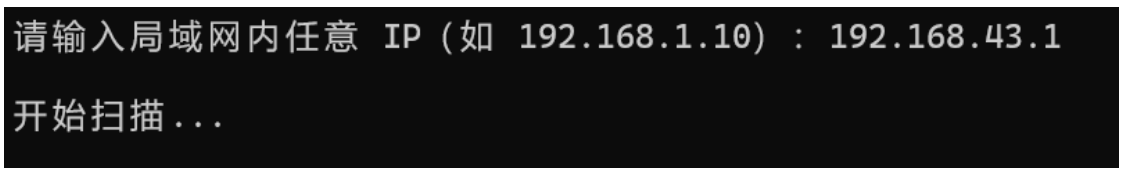
接收 ICMP Echo Reply

验证报文类型、ID 和源地址

判断目标主机是否在线

五、实验结果与分析

程序运行后，首先显示扫描的局域网网段信息



随后程序逐个 IP 进行检测，并输出结果

```
开始扫描...

[在线] 192.168.43.1
[离线] 192.168.43.2
[离线] 192.168.43.3
[离线] 192.168.43.4
[离线] 192.168.43.5
[离线] 192.168.43.6
[离线] 192.168.43.7
[离线] 192.168.43.8
[离线] 192.168.43.9
[离线] 192.168.43.10
```

最后列出在线 IP

```
========== 扫描完成 ==========
在线主机数量: 2

在线 IP 列表:
 1. 192.168.43.1
 2. 192.168.43.138
```

六、小结与心得体会

通过本次实验，加深了对 ICMP 协议 以及 Ping 工作原理 的理解，掌握了使用 Raw Socket 进行网络编程的方法。在实验过程中，体会到仅“接收到数据”并不能代表主机在线，必须对 ICMP 报文的类型、标识符和源地址进行严格校验。本实验不仅帮助理解了常用网络诊断工具的底层实现原理，也为后续学习 Tracert、多线程扫描及网络安全相关技术奠定了基础，对提高实际网络分析能力具有重要意义。

实验三

一、实验题目

题目五：网络路由模拟器设计与实现

二、实验目的

1.理解计算机网络中路由器、路由表与网络拓扑之间的关系。
2.掌握数据包在多跳路由环境中的转发流程。
3.通过扩展路由功能（如 R1 → NET4），加深对路由可达性与下一跳机制的理解。

三、总体设计

1.背景知识
在实际网络中，路由器的核心功能是：

- 接收数据包
- 解析目的 IP 地址
- 查找路由表
- 根据最长前缀匹配原则选择最合适的路由条目
- 从对应的接口将数据包转发给下一跳

当网络中存在多台路由器时，数据包往往需要经过多次转发才能到达最终目的网络。
2.总体设计思路
每台路由器维护独立的路由表
路由表包含：

- 目的网络地址
- 子网掩码
- 下一跳 IP
- 出接口名称

数据包在路由器之间逐跳转发
使用 TTL 机制防止转发环路
3.模块划分

- IP 地址处理模块：
  IP 字符串与 32 位整数之间的转换
- 路由表管理模块：
  路由条目的添加
- 路由表的显示：
  路由匹配模块
  根据目的 IP 进行最长前缀匹配
- 数据包转发模块：

模拟数据包在多路由器之间逐跳转发

四、详细设计（含主要的数据结构、程序流程图、关键代码等）

1.主要数据结构

路由表项结构：

```
typedef struct {
    uint32_t dest;      // 目的网络
    uint32_t mask;      // 子网掩码
    uint32_t nextHop;   // 下一跳 IP
    char iface[10];     // 出接口
} RouteEntry;
```

路由器结构：

```
typedef struct {
    RouteEntry table[MAX_ROUTES];
    int count;
    char name[10];
} Router;
```

2.网络拓补

```
========== 网络拓扑结构 ==========
    R1  R2  R3  R4  R5
R1  0   1   0   1   0
R2  1   0   1   0   0
R3  0   1   0   0   1
R4  1   0   0   0   1
R5  0   0   1   1   0
```

五、实验结果与分析

路由表信息

```
========== 路由表信息 ==========

[R1 路由表]
目的网络        下一跳  距离
NET3            R2      2
NET5            R4      3

[R2 路由表]
目的网络        下一跳  距离
NET3            R3      2

[R3 路由表]
目的网络        下一跳  距离
NET3            NET3    0

[R4 路由表]
目的网络        下一跳  距离
NET5            R5      2

[R5 路由表]
目的网络        下一跳  距离
NET5            NET5    0
================================
```

根据 R1 的路由表，目的网络 NET5 的下一跳为 R4，因此数据包首先由 R1 转发至 R4；路径代价为 3。
随后，在 R4 的路由表中，NET5 的下一跳为 R5，因此数据包继续从 R4 转发至 R5，路径代价为 2。此时累计路径开销为 5。

```
===== 路由模拟器 =====
1. 显示路由表
2. 查询路由并模拟转发
0. 退出
请选择: 2
请输入当前路由器: R1
请输入目的网络: NET5

=== 转发路径 ===
R1 -> R4 (cost=3)
R4 -> R5 (cost=2)
路由错误: 下一跳不可达
总距离 = 5
```

六、小结与心得体会

在本实验中，通过构建抽象网络拓扑与路由表模型，成功实现了多跳路由转发过程的可视化模拟。通过本实验，不仅加深了对“目的网络－下一跳－代价”这一经典路由模型的理解，也掌握了用程序表达复杂网络结构的方法。

实验四

一、实验题目

题目六：编程模拟 NAT 网络地址转换

二、实验目的

1.理解网络地址转换（NAT）及端口地址转换（NAPT）的基本工作原理；
2.掌握 NAT 路由器在出方向与入方向对 IP 地址与端口号的转换过程；
3.通过 C 语言实现一个简化的 NAT 路由器模型，加深对端口映射表作用的理解；
4.提升对网络协议中“状态表”“会话映射”等核心思想的认识。

三、总体设计

1.背景知识

- 端口映射：
  用“公网 IP + 端口”去唯一对应一个内网主机的“私有 IP + 端口”

NAT 通过端口映射机制，在仅有一个公网 IP 的情况下，使用不同的端口号区分不同内网主机和会话。出方向建立映射表，入方向根据端口反查并还原内网地址，从而实现公网访问和内网地址隐藏

- 出方向（内网 → 外网）:
  将内网私有 IP 地址替换为 NAT 的公网 IP；
  将内网源端口替换为 NAT 分配的公网端口；
  在 NAT 映射表中记录对应关系。+
- 入方向（外网 → 内网）:
  根据报文目的端口查找 NAT 映射表；
  还原出对应的内网 IP 地址与端口；
  将报文转发给正确的内网主机。

2.模块划分

- NAT 映射表模块：
  存储私有地址与公网地址之间的转换关系；
- 端口分配模块：
  为每个新连接分配唯一的公网端口；
- 出方向转换模块：
  实现私有 IP/端口 → 公网 IP/端口 的转换；
- 入方向反向查表模块：
  根据公网端口查找并恢复内网地址；
- 交互模块：
  接收用户输入，模拟内外网数据包。

3.设计步骤

- 定义 NAT 映射表的数据结构；
- 设置固定的 NAT 公网 IP 地址；
- 实现端口分配与映射表添加功能；
- 实现根据公网端口反向查找映射表功能；
- 通过交互式输入模拟内外网报文转换过程；
- 输出 NAT 映射表以验证转换结果。

四、详细设计（含主要的数据结构、程序流程图、关键代码等）

1.主要数据结构

```
//NAT 映射表中的记录
    typedef struct {
        char private_ip[16];
        int private_port;
        char public_ip[16];
        int public_port;
    } NATEntry;
//映射表设计
//nat_table 存储所有映射关系；
//nat_count 记录当前映射表中的有效条目数量。
    NATEntry nat_table[100];
    int nat_count = 0;
```

2.关键功能实现

- 端口映射建立（出方向）：

```
    void add_nat_entry(const char* pip, int pport, int pub_port);
```

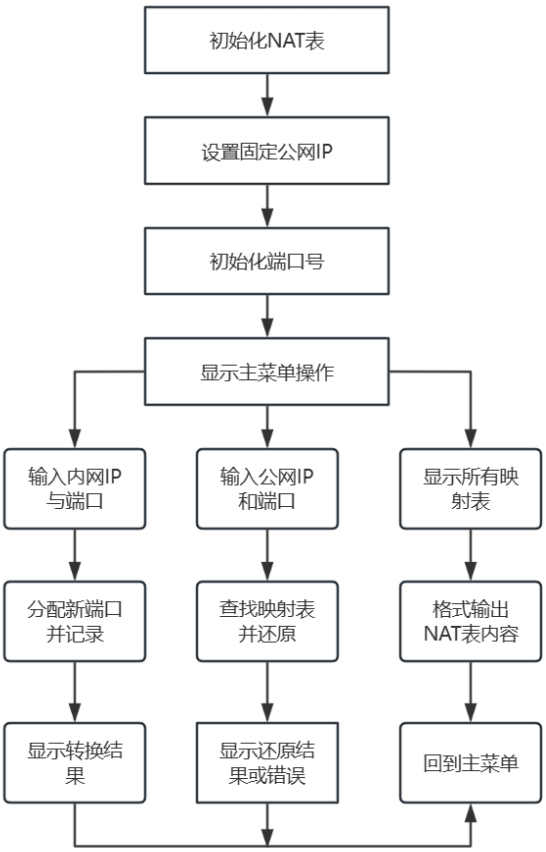
接收内网主机的私有 IP 和端口；
分配 NAT 公网端口；
将映射关系保存到 NAT 表中。

- 反向查找映射（入方向）

```
    NATEntry* find_by_public_port(int port);
```

根据外网应答报文中的目的端口；
在 NAT 映射表中查找对应条目；
恢复内网 IP 地址和端口号。

3.程序流程图



五、实验结果与分析

1.出方向正常转换

```
===== NAT 路由器模拟 =====
1. 出方向（内网 → 外网）
2. 入方向（外网 → 内网）
3. 查看 NAT 转换表
0. 退出
请选择: 1
请输入私有 IP: 192.168.0.3
请输入私有端口: 30000

【NAT 出方向转换】
源地址 192.168.0.3:30000  →  172.38.1.5:40001
```

2.入方向正常还原

```
===== NAT 路由器模拟 =====
1. 出方向（内网 → 外网）
2. 入方向（外网 → 内网）
3. 查看 NAT 转换表
0. 退出
请选择: 2
请输入目的 IP（公网）: 172.38.1.5
请输入目的端口（公网）: 40001

【NAT 入方向还原】
目的地址 192.168.0.3:30000
```

3.查看 NAT 映射表

```
===== NAT 路由器模拟 =====
1. 出方向（内网 → 外网）
2. 入方向（外网 → 内网）
3. 查看 NAT 转换表
0. 退出
请选择: 3

====== NAT 转换表 ======
私有IP:端口          公网IP:端口
192.168.0.3       :30000  172.38.1.5:40001
192.168.0.4       :30000  172.38.1.5:40002
========================
```

六、小结与心得体会

通过编写 NAT 模拟程序，我深入理解了 NAT/NAPT 的基本工作原理。特别是端口映射机制，让我明白了单一公网 IP 如何通过不同的端口号来区分多个内网主机的网络会话。NAT 映射表本质上是一个状态表，它记录了当前活跃的网络连接。

实验五

一、实验题目

题目七：网络嗅探器

二、实验目的

1.理解 IP、TCP、UDP、ICMP 等协议在网络通信中的分层结构与作用。
2.掌握 Windows 平台下使用原始套接字（Raw Socket）捕获网络数据包的方法。
3.实现对网络中传输的数据包的源地址、目的地址、端口号和协议类型的解析。
4.能够分析 TCP、UDP、ICMP 报文的结构，并提取其数据负载内容。
5.通过实验认识明文协议与加密协议（如 HTTPS）在抓包中的差异。

三、总体设计

1.实验原理
Windows 提供的 Raw Socket（原始套接字）允许应用程序直接接收 IP 层的数据包。
通过创建 SOCK_RAW 类型的套接字并开启混杂模式（SIO_RCVALL），可以捕获经过本机网卡的所有 IP 数据包。
程序的工作流程如下：

- 创建原始套接字：socket(AF_INET, SOCK_RAW, IPPROTO_IP)
- 绑定本机 IP 地址
- 启用混杂模式（Promiscuous Mode）
- 使用 recvfrom() 不断接收网络数据包
- 解析 IP 头，根据 proto 字段判断是 TCP、UDP 还是 ICMP
- 根据协议类型解析对应的传输层头部
- 计算数据负载偏移，输出数据内容

2.主要模块

- Socket 初始化：创建 Raw Socket 并绑定网卡
- 混杂模式控制：允许接收所有经过网卡的数据
- IP 解析模块：提取源 IP、目的 IP、协议类型
- TCP/UDP/ICMP 解析模块：提取端口、协议字段
- 数据负载显示模块：输出 TCP/UDP/ICMP 数据

四、详细设计（含主要的数据结构、程序流程图、关键代码等）

1.主要数据结构

- IP 头部：

```
typedef struct ip_header {
unsigned char  ver_ihl;
unsigned char  tos;
```

```
    unsigned short tlen;
    unsigned short identification;
    unsigned short flags_fo;
    unsigned char  ttl;
    unsigned char  proto;
    unsigned short crc;
    unsigned int   saddr;
    unsigned int   daddr;
} IP_HEADER;
```

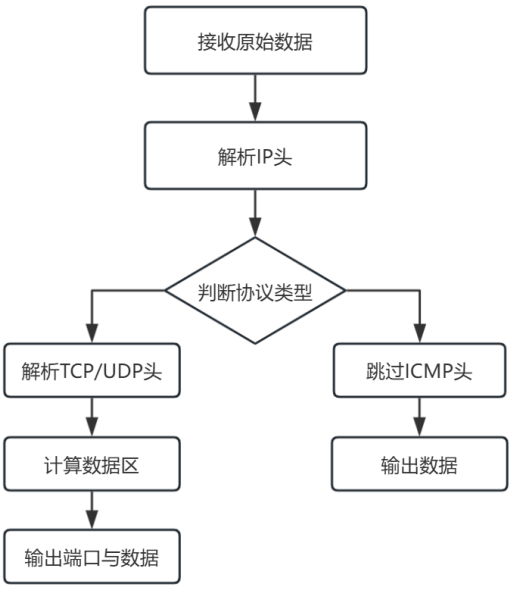
- TCP 头部：

```
typedef struct tcp_header {
    USHORT source;
    USHORT dest;
    ULONG  seq;
    ULONG  ack_seq;
    UCHAR  data_offset;
    UCHAR  flags;
    USHORT window;
    USHORT checksum;
    USHORT urg_ptr;
} TCP_HEADER;
```

- UDP 头部：

```
typedef struct udp_header {
    USHORT source;
    USHORT dest;
    USHORT len;
    USHORT checksum;
} UDP_HEADER;
```

2.数据包解析流程



五、实验结果与分析

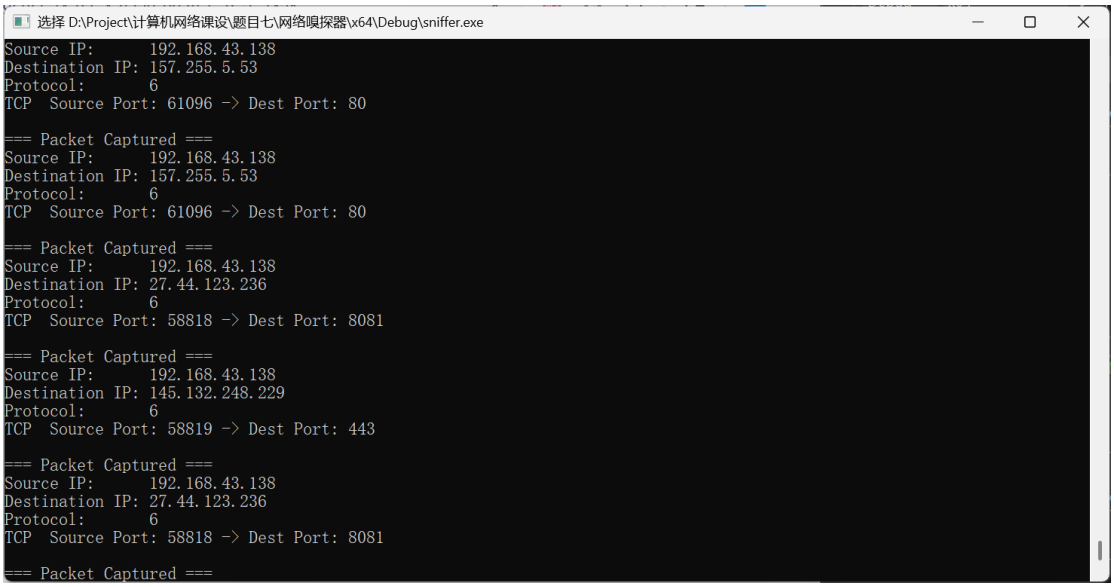
本程序能够捕获：

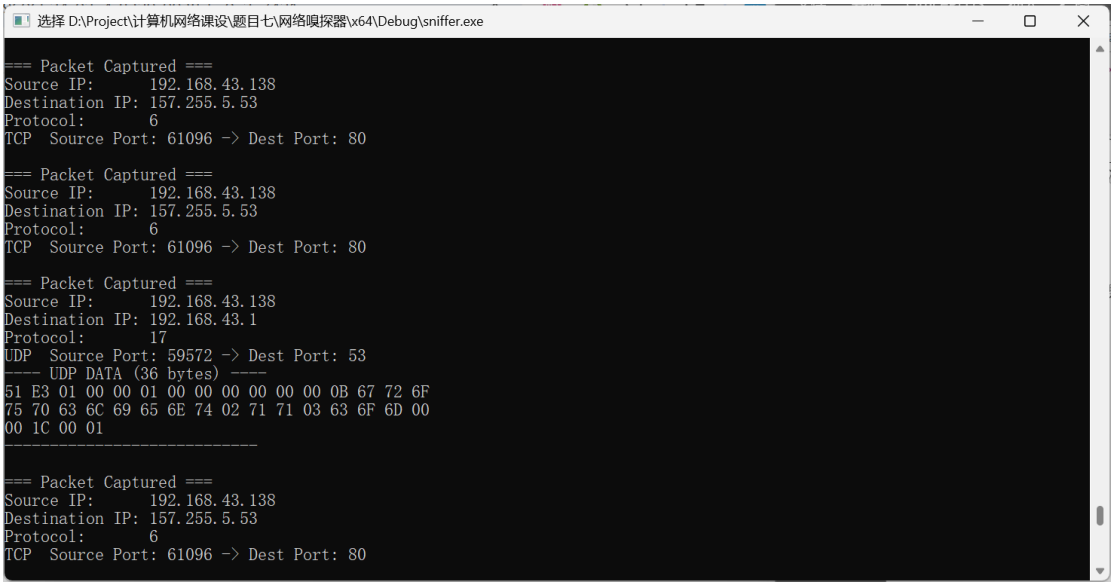
TCP、UDP、ICMP 等所有 IP 层协议

本机发送和接收的所有网络流量

HTTP（80 端口）的明文内容

DNS、ICMP 等 UDP 协议数据





六、小结与心得体会

通过本次实验，成功实现了一个基于 Windows Raw Socket 的网络嗅探器，能够实时捕获并解析 TCP、UDP、ICMP 数据包，并显示源 IP、目的 IP、端口号和数据负载。

网络通信是分层结构，IP 只负责寻址，TCP/UDP 负责传输

HTTPS 的数据在传输过程中已经被 TLS 加密，即使抓到也无法直接读取

TCP 是流协议，一个应用层数据可能被拆分为多个 TCP 段