**北京邮电大学《计算机网络》课程设计报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **课程设计**  **名称** | 计算机网络课程设计 | | **学 院** | 计算机 | **指导教师** | 高占春 |
| **班 级** | **班内序号** | **学 号** | | **学生姓名** | **成绩** | |
| **2023211303** | **31** | **2023212872** | | **计子毅** |  | |
| **2023211303** | **28** | **2023210710** | | **陈子容** |  | |
| **2023211303** | **30** | **2023212219** | | **江宝金** |  | |
| **课**  **程**  **设**  **计**  **内**  **容** | DNS中继器可以查询本地”域名-IP地址“对照表，当出现”0.0.0.0“时，对响应域名进行  屏蔽，向客户端发送错误提示，此时客户端显示”Non-existent domain“的报错信息。  当在本地找到域名对应IP时，构造报文发送给客户端。  如果在本地没有该域名，就启用中继功能，发送给服务器端。  发送服务器端要注意IP转换逻辑。  一个域名对应多个IP，达到负载均衡。  DNS超时处理。  LRU缓存  成员分工：  陈子容:dataStruct部分  计子毅:dnsStruct部分  江宝金:server\system部分 | | | | | |
| **学生课程设计报告**  **(附页)** | （详见“实验报告和源程序”册） | | | | | |
| **课**  **程**  **设**  **计**  **成**  **绩**  **评**  **定** | **评语**:  **成绩**:  指导教师签名：    年 月 日 | | | | | |

注：评语要体现每个学生的工作情况，可以加页。

目录

[1. 实验环境 2](#_Toc200640855)

[2. 功能介绍 2](#_Toc200640856)

[2.1. 项目概述 2](#_Toc200640857)

[2.2. 核心功能 2](#_Toc200640858)

[2.2.1. DNS报文处理 2](#_Toc200640859)

[2.2.2. 本地域名解析 2](#_Toc200640860)

[2.2.3. 缓存管理 3](#_Toc200640861)

[2.2.4. 远程DNS代理 3](#_Toc200640862)

[2.2.5. 域名屏蔽 3](#_Toc200640863)

[2.2.6. 辅助功能 3](#_Toc200640864)

[2.3. 技术特点 4](#_Toc200640865)

[2.3.4.可配置性​​： 4](#_Toc200640866)

[2.4. 文件说明 4](#_Toc200640867)

[3. DNS中继器原理介绍 5](#_Toc200640868)

[3.1. DNS处理逻辑 5](#_Toc200640869)

[3.1.1. 域名结构 5](#_Toc200640870)

[3.1.2. DNS中继的作用 5](#_Toc200640871)

[3.1.3. DNS缓存 5](#_Toc200640872)

[3.1.4. DNS工作流程 5](#_Toc200640873)

[3.2. ID转换逻辑 6](#_Toc200640874)

[3.3. DNS报文分析 7](#_Toc200640875)

[3.3.1. DNS包头字段 7](#_Toc200640876)

[3.3.2. DNS报文内容与资源记录 11](#_Toc200640877)

[3.3.3. 指针压缩机制 14](#_Toc200640878)

[4. 项目目录分析 15](#_Toc200640879)

[4.1. 核心源码文件 15](#_Toc200640880)

[4.2. 配置和头文件 15](#_Toc200640881)

[4.3. 数据文件 15](#_Toc200640882)

[4.4. 项目构建文件 15](#_Toc200640883)

[4.5. 输出目录 16](#_Toc200640884)

[4.6. 文件依赖关系 16](#_Toc200640885)

[5. 模块划分与具体实现 16](#_Toc200640886)

[5.1. header模块 16](#_Toc200640887)

[5.1.1. 概述 16](#_Toc200640888)

[5.1.2. 包含的头文件 17](#_Toc200640889)

[5.1.3. 编译指令 17](#_Toc200640890)

[5.2. default模块 17](#_Toc200640891)

[5.2.1. 概述 17](#_Toc200640892)

[5.2.2. 常量定义 17](#_Toc200640893)

[5.2.3. 数据类型定义 18](#_Toc200640894)

[5.2.4. 资源记录类型 18](#_Toc200640895)

[5.3. dataStruct模块 18](#_Toc200640896)

[5.3.1. 概述 18](#_Toc200640897)

[5.3.2. IP链表 18](#_Toc200640898)

[5.3.3. ID转换 19](#_Toc200640899)

[5.3.4. Trie树 20](#_Toc200640900)

[5.3.5. LRU缓存 21](#_Toc200640901)

[5.4. dnsStruct模块 24](#_Toc200640902)

[5.4.1. DNS报文结构 24](#_Toc200640903)

[5.4.2. 标志位掩码定义 26](#_Toc200640904)

[5.4.3. 主要功能函数 26](#_Toc200640905)

[5.5. Socket模块 28](#_Toc200640906)

[5.5.1. Socket架构设计 28](#_Toc200640907)

[5.5.2. Socket初始化过程 (system.c: init\_socket函数) 28](#_Toc200640908)

[5.5.3. 非阻塞I/O配置 (server.c: poll函数) 29](#_Toc200640909)

[5.5.4. 事件驱动机制 29](#_Toc200640910)

[5.5.5. 数据收发操作 29](#_Toc200640911)

[5.5.6. Socket关闭 (server.c: closeServer函数) 30](#_Toc200640912)

[6. 主函数流程与流程图 30](#_Toc200640913)

[6.1. 流程图 30](#_Toc200640914)

[6.2. main 函数完整流程分析 31](#_Toc200640915)

[6.2.1. 初始化 32](#_Toc200640916)

[6.2.2. 主循环 32](#_Toc200640917)

[6.2.3. 处理数据包 33](#_Toc200640918)

[6.3. 程序启动流程 34](#_Toc200640919)

[6.3.1. main函数入口 34](#_Toc200640920)

[6.3.2. 系统初始化流程 (system.c: init函数) 34](#_Toc200640921)

[6.4. 主事件循环流程 (server.c: poll函数) 35](#_Toc200640922)

[6.4.1. 超时处理：timeout\_handle() 35](#_Toc200640923)

[6.4.2. 事件监听：WSAPoll() 35](#_Toc200640924)

[6.4.3. 事件分发： 35](#_Toc200640925)

[6.5. 客户端请求处理流程 (server.c: receiveClient函数) 35](#_Toc200640926)

[6.5.1. 接收DNS查询报文 35](#_Toc200640927)

[6.5.2. DNS报文解析 36](#_Toc200640928)

[6.5.3. 三级查找策略 36](#_Toc200640929)

[6.5.4. 本地响应处理 36](#_Toc200640930)

[6.6. 服务器响应处理流程 (server.c: receiveServer函数) 37](#_Toc200640931)

[6.6.1. 接收上游DNS响应 37](#_Toc200640932)

[6.6.2. DNS响应解析 37](#_Toc200640933)

[6.6.3. ID转换和路由 37](#_Toc200640934)

[6.6.4. 缓存更新 37](#_Toc200640935)

[6.6.5. 本地数据更新 37](#_Toc200640936)

[6.6.6. 响应转发 37](#_Toc200640937)

[6.7. 数据结构操作流程 38](#_Toc200640938)

[6.7.1. Trie树操作 (dataStruct.c) 38](#_Toc200640939)

[6.7.2. LRU缓存操作 (dataStruct.c) 38](#_Toc200640940)

[6.7.3. ID映射表操作 (dnsStruct.c) 38](#_Toc200640941)

[6.8. 错误处理和日志 38](#_Toc200640942)

[6.8.1. 网络错误处理 38](#_Toc200640943)

[6.8.2. 调试信息输出 38](#_Toc200640944)

[6.8.3. 日志记录 39](#_Toc200640945)

[7. 测试分析 39](#_Toc200640946)

[7.1. 准备工作 39](#_Toc200640947)

[7.2. 拦截 39](#_Toc200640948)

[7.3. 本地查询 41](#_Toc200640949)

[7.4. 中继 42](#_Toc200640950)

[7.5. 缓存 42](#_Toc200640951)

[7.6. 数据库更新 43](#_Toc200640952)

[8. 问题分析 44](#_Toc200640953)

[8.1. IPV6查询结果错误问题 44](#_Toc200640954)

[8.2. 文件描述符溢出 44](#_Toc200640955)

[8.3. IPV6应答数据报抓包结果 44](#_Toc200640956)

[8.4. 总结与收获 44](#_Toc200640957)

# 实验环境

Windows11操作系统

Visual Studio 2022

C语言

# 功能介绍

## 项目概述

这是一个基于C语言实现的DNS代理服务器，主要功能包括：

* 本地DNS缓存和解析
* 远程DNS服务器代理
* IPv4和IPv6双栈支持
* 域名屏蔽功能
* LRU缓存管理
* 请求转发和ID转换

## 核心功能

### DNS报文处理

* 完整解析DNS请求和响应报文
* 支持A记录(IPv4)和AAAA记录(IPv6)查询
* 处理DNS报文头部、问题部分和资源记录部分
* 实现域名压缩指针解析

### 本地域名解析

* 使用Trie树数据结构高效存储和查询域名
* 从本地host文件加载域名-IP映射关系
* 支持IPv4和IPv6地址格式解析和存储
* 自动将远程查询结果保存到本地host文件

### 缓存管理

* 实现LRU(最近最少使用)缓存算法
* 缓存容量可配置(默认100条记录)
* 缓存命中时自动提升记录优先级
* 缓存满时自动淘汰最久未使用的记录

### 远程DNS代理

* 可配置远程DNS服务器地址
* 实现请求ID转换和映射
* 处理请求超时(默认1秒)
* 支持调试模式查看详细转发信息

### 域名屏蔽

检测到0.0.0.0或::0地址时返回域名错误

可扩展的屏蔽规则管理

### 辅助功能

* 支持调试模式(-d参数)
* 日志记录功能(-l参数)
* 可配置远程DNS服务器地址(-s参数)
* 帮助信息显示(-i参数)

## 技术特点

​​2.3.1．高效数据结构​​：

使用Trie树存储域名，实现快速查找

LRU链表管理缓存记录

​​2.3.2.网络通信​​：

非阻塞I/O模型

WSAPoll实现高效事件监听

UDP套接字通信

​​2.3.3.协议处理​​：

完整DNS协议解析和构造

处理网络字节序转换

支持域名压缩指针

### 2.3.4.可配置性​​：

通过命令行参数调整运行模式

可指定远程DNS服务器

缓存大小可配置

## 文件说明

host\_ipv4.txt - IPv4域名映射文件

host\_ipv6.txt - IPv6域名映射文件

log.txt - 运行日志文件

# DNS中继器原理介绍

## DNS处理逻辑

### 域名结构

域名（www.example.com）作为统一的网络标识符，方便用户记忆和访问网络资源。替代IP地址（255.255.255.255）来访问网络资源，解决了IP地址的难记忆性和不稳定性问题。

域名结构采用了层级结构，从右向左逐级表示更高级别的域名。

### DNS中继的作用

DNS（Domain Name System，域名系统）是互联网中的一种分布式数据库系统，它用于将域名与IP地址相互映射，以便于在网络中定位计算机和服务。

DNS就像是互联网的电话簿，它将人类可读的域名（如example.com）转换为计算机可识别的IP地址（如192.0.2.1），从而实现网络通信。

当用户在浏览器中输入一个域名时，比如"www.example.com"，DNS系统会启动域名解析过程：

**DNS客户端：**用户的计算机或网络设备，负责向DNS服务器发送查询请求。

**DNS服务器：**接收来自DNS客户端的查询请求，负责从根域名服务器开始递归地查询，直到找到所请求的域名的IP地址或其他相关信息，并将结果返回给DNS客户端。

### DNS缓存

DNS缓存是指DNS系统中的一种机制，用于临时存储已解析过的域名与IP地址之间的映射关系。当DNS服务器接收到查询请求并成功解析域名后，它会将解析结果存储在缓存中一段时间，以便在未来相同域名的查询请求中快速返回相应的解析结果。

### DNS工作流程

日程表

AI 生成的内容可能不正确。

1. 初始化数据结构，读入本地对照表

2. 创建套接字用于接收和发送DNS查询和响应报文，绑定到指定端口

3. 中继器接收来自客户端的DNS查询请求包。解析DNS请求包中的域名和查询类型。在LRU缓存中查找该域名对应的IP地址：

* 如果找到，则构造DNS响应包并发送给客户端。
* 如果未找到，则继续在本地哈希表中查找该域名对应的IP地址：
  + 如果在本地哈希表中找到，则构造DNS响应包并发送给客户端，同时将查询结果更新到LRU缓存中。
  + 如果在本地哈希表中也未找到，则将请求转发给上游DNS服务器。

4. 中继器接收上游DNS服务器返回的响应包，将响应包中的查询ID转换为客户端的查询ID。将DNS响应包发送给客户端。

5. 定时器线程周期性检查ID转换表中的记录，处理超时的DNS请求。

## ID转换逻辑

DNS查询请求和响应通过一个16位的ID字段关联。客户端发送请求时，会生成一个查询ID，上游DNS服务器在响应时会使用相同的ID。

由于中继器可能同时处理来自多个客户端的多个DNS查询请求，如果不进行ID转换，可能会出现多个请求使用相同的查询ID，导致响应无法正确匹配到对应的客户端请求。通过ID转换，中继器能够确保每个转发给上游DNS服务器的查询具有唯一的查询ID，从而避免ID冲突问题。

ID转换模块还可以记录每个查询请求的状态，包括请求的客户端地址、请求时间等信息，以便在需要时（例如超时处理）生成相应的错误响应并发送给客户端。

图示

AI 生成的内容可能不正确。

**如图分析：**

* Client与DNS relay的通信过程中，ID怎么来怎么回，即如果查询在本地找到结果了就直接返回
* 但在DNS relay与DNS server的通信过程中，需要进行ID的转换

**ID转换的实现:**

维护一个一维IP结构体列表，该列表即为ID转换表。当需要进行ID转换时，调用transfer\_id函数，遍历该列表，找到第一个未使用或被使用但超时的位置，将其替换，使用该位置的下标作为新的ID并向上层DNS服务器传输

## DNS报文分析

### DNS包头字段

**DNS报头总共12字节，包含以下字段：**

1. 事务id
2. 标志位flags
3. 问题数ques\_num
4. 回答数ans\_num
5. 授权数auth\_num
6. 附加数add\_num

DNS 报文的 ID 标识。对于请求报文和其对应的应答报文，该字段的值是相同的。通过它可以区分 DNS 应答报文是对哪个请求进行响应的。

**标志位细节如下：**

QR：1位，查询/响应标志位。0表示查询报文，1表示响应报文。

OPCODE：4位，操作码。通常取值为0，表示标准查询；取值为1，表示反向查询；取值为2，表示服务器状态查询。

AA：1位，授权回答。该位在响应报文中有效，表示响应的服务器是一个权威服务器。

TC：1位，截断。指示报文是否被截断。

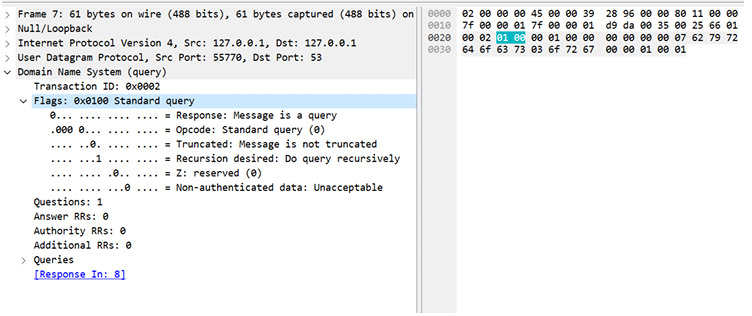
RD：1位，期望递归。表示发出查询的主机希望 DNS 服务器递归查询。

RA：1位，可用递归。表示 DNS 服务器支持递归查询。

Z：3位，保留字段。

RCODE：4位，返回码。表示响应的返回状态。通常取值为0，表示没有错误；取值为3，表示名字错误。

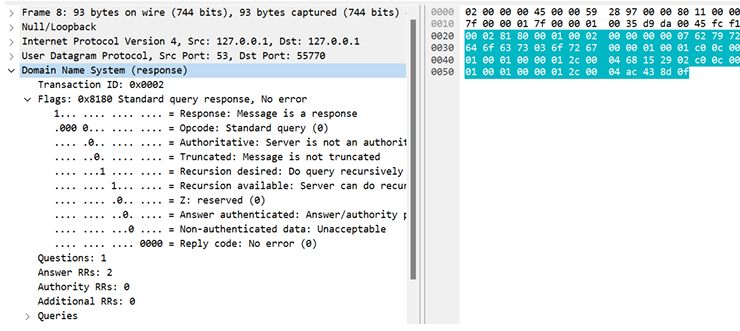
**查询报文：**

****

可提取如下：

|  |  |  |
| --- | --- | --- |
| 子字段名称 | 二进制值 | 说明 |
| QR | 0 | 查询 |
| Opcode | 0000 | 标准查询 |
| AA | 0 | 非授权回答 |
| TC | 0 | 未截断 |
| RD | 1 | 递归查询期望 |
| RA | 0 | 递归查询不可用 |
| Z | 000 | 保留字段，为0 |
| RCODE | 0000 | 无错误 |

**响应报文：**



|  |  |  |
| --- | --- | --- |
| 子字段名称 | 二进制值 | 说明 |
| QR | 1 | 响应 |
| Opcode | 0000 | 标准查询 |
| AA | 0 | 非授权回答 |
| TC | 0 | 未截断 |
| RD | 1 | 递归查询期望 |
| RA | 1 | 递归查询可用 |
| Z | 000 | 保留字段，为0 |
| RCODE | 0000 | 无错误 |

**错误响应报文：**

**图形用户界面, 文本, 应用程序

AI 生成的内容可能不正确。**

|  |  |  |
| --- | --- | --- |
| 子字段名称 | 二进制值 | 说明 |
| QR | 1 | 响应 |
| Opcode | 0000 | 标准查询 |
| AA | 0 | 非授权回答 |
| TC | 0 | 未截断 |
| RD | 1 | 递归查询期望 |
| RA | 1 | 递归查询可用 |
| Z | 000 | 保留字段，为0 |
| RCODE | 0011 | 名字错误 |

### DNS报文内容与资源记录

在 DNS 报文头部之后，报文包含四个主要部分：Questions、Answers、Authority、Additional。

#### Questions 部分

1. QName（域名）

* 长度：可变
* 功能：要查询的域名，使用点分割的每个部分前都有一个长度字节（标签）。

例子：www.bilibili.com 被编码为：3www8bilibili3com0

2. QType（查询类型）

* 长度：2 字节
* 功能：指定查询的记录类型，如 A、CNAME、MX、TXT 等。
* 例子：HTTPS (65)

3. QClass（查询类）

* 长度：2 字节
* 功能：指定查询的类，通常为 IN（Internet）。
* 例子：IN (0x0001)

#### Answers 部分

1. Name（域名）

* 长度：可变
* 功能：被查询的域名。
* 例子：www.bilibili.com

2. Type（资源记录类型）

* 长度：2 字节
* 功能：资源记录的类型，如 A、CNAME、MX、TXT 等。
* 例子：CNAME (5)

3. Class（资源记录类）

* 长度：2 字节
* 功能：资源记录的类，通常为 IN（Internet）。
* 例子：IN (0x0001)

4. TTL（生存时间）

* 长度：4 字节
* 功能：资源记录的生存时间，以秒为单位。
* 例子：227（3 分钟 47 秒）

5. Data Length（数据长度）

* 长度：4 字节
* 功能：RData 字段的长度。
* 例子：18

6. RData（资源数据）

* 长度：可变
* 功能：资源记录的具体数据。
* 例子：172.67.141.15

#### Authority 部分

1. Name（域名）

* 长度：可变
* 功能：被查询的域名。
* 例子：bilicdn1.com

2. Type（资源记录类型）

* 长度：2 字节
* 功能：资源记录的类型，如 SOA、NS 等。
* 例子：SOA (6)

3. Class（资源记录类）

* 长度：2 字节
* 功能：资源记录的类，通常为 IN（Internet）。
* 例子：IN (0x0001)

4. TTL（生存时间）

* 长度：4 字节
* 功能：资源记录的生存时间，以秒为单位。
* 例子：17（17 秒）

5. Data Length（数据长度）

* 长度：2 字节
* 功能：RData 字段的长度。
* 例子：67

6. RData（资源数据）

* 长度：可变
* 功能：资源记录的具体数据，具体数据结构因记录类型而异。
* 例子：ns3.dnsv5.com...

#### Additional 部分

Additional 部分的字段和 Answers、Authority 部分相同，具体字段和功能可以参照上述两部分的描述。Additional 部分通常包含额外的资源记录，这些记录可能用于进一步的查询或提供附加信息。

### 指针压缩机制

在回答报文中，可能需要出现多次域名，但重复的存放域名会极大的浪费空间，所以采用指针压缩，使用2字节的指针就可以指向前面已经记录的域名，通过读取位置的跳转即可取到域名，避免了重复储存。

在 DNS 报文中，c00c 表示一个压缩指针，它是域名压缩的一部分。压缩指针用于减少 DNS 报文的长度，避免重复存储域名。指针指向前面报文中已经出现过的域名位置。具体来说，指针格式为两个字节，高两位为 11，表示这是一个压缩指针，其余 14 位表示偏移量。

假设 DNS 报文的前 12 字节是头部，从第 12 字节开始是查询部分：

Header (12 bytes) | 3www7example3com0 | Type | Class | TTL | Data Length | Address |

在这个示例中：

假设 currentPosition 初始为 12，指向3www

域名 www.example.com 编码为 3www7example3com0

查询部分结束后，假设在回答部分 name 字段出现 c00c

c00c的偏移量为 12，指向第 12 字节处。

第 12 字节处的内容是 3www7example3com0 ，表示 www.example.com

因此，c00c 表示的域名是 www.example.com

# 项目目录分析

## 核心源码文件

main.c - 程序入口，调用初始化和主循环

server.c/server.h - DNS服务器核心逻辑，事件循环和请求处理

system.c/system.h - 系统初始化，Socket配置，参数处理

dataStruct.c/dataStruct.h - 数据结构实现，Trie树和LRU缓存

dnsStruct.c/dnsStruct.h - DNS协议实现，报文解析和组装

## 配置和头文件

header.h - 主头文件，包含所有必要的系统头文件

default.h - 常量定义，如缓存大小、超时时间等

## 数据文件

host\_ipv4.txt - IPv4域名-IP映射文件

host\_ipv6.txt - IPv6域名-IP映射文件

host.txt - 通用host文件

log.txt - 日志文件(运行时生成)

## 项目构建文件

DNS.sln - Visual Studio解决方案文件

DNS.vcxproj - Visual Studio项目文件

DNS.vcxproj.filters - 项目文件过滤器

DNS.vcxproj.user - 用户特定设置

## 输出目录

x64/Debug/ - 64位调试版本输出目录

DNS/x64/ - 项目特定输出目录

## 文件依赖关系

main.c

├── header.h (系统头文件)

├── system.h (系统初始化)

└── server.h (服务器逻辑)

server.c

├── dataStruct.h (数据结构)

├── dnsStruct.h (DNS协议)

└── system.h (系统配置)

system.c

├── dataStruct.h (Trie树操作)

└── default.h (常量定义)

# 模块划分与具体实现

## header模块

### 概述

header负责封装网络编程相关头文件，主要包含 Windows 平台下 Socket 编程所需的基本头文件引用和库链接指令。

### 包含的头文件

该部分包含了C语言中常用的头文件，用于输入输出、内存管理、字符串处理、标准整数类型定义、Windows网络编程以及时间处理功能。

#include <stdio.h> // 标准输入输出函数

#include <stdlib.h> // 标准库函数，如内存分配等

#include <string.h> // 字符串处理函数

#include <stdint.h> // 标准整数类型定义

#include <WinSock2.h> // Windows Socket 2.0 主要头文件

#include <time.h> // 时间处理函数

### 编译指令

该部分用于在 Windows 环境下链接 ws2\_32.lib 网络库，并禁用 C4996（不安全函数）的编译器警告。

#pragma comment(lib, "ws2\_32.lib") // 指示链接器链接 Windows Socket 2.0 库

#pragma warning(disable:4996) // 禁用 C4996 警告

## default模块

### 概述

default定义了 DNS 代理服务器中使用的常量、宏和基本数据类型，为整个项目提供统一的配置参数。

### 常量定义

该部分定义了一个DNS代理服务器的核心参数，包括字典树节点数、域名长度限制、缓存大小、监听端口、报文尺寸、ID映射表大小、过期时间以及最大存储的host-IP对数量。

#define NODE\_NUM 65535 // 字典树的最大结点数

#define MAX\_SIZE 300 // 域名最大长度

#define MAX\_CACHE 100 // 缓存最大容量

#define DEFAULT\_PORT 53 // 监听端口号

#define BUFFER\_SIZE 1500 // DNS报文的最大尺寸

#define ID\_LIST\_SIZE 128 // ID映射表大小

#define ID\_EXPIRE\_TIME 1 // ID过期时间

#define MAX\_HOST\_SAVE 15000 // 最大保存的host-ip对数

### 数据类型定义

该部分定义 dns\_addr 为指向 uint8\_t 的指针类型，用于表示 DNS 数据地址。

typedef uint8\_t\* dns\_addr; // 定义DNS地址指针类型

### 资源记录类型

该部分宏定义列举了常见的 DNS 资源记录类型，用于在 DNS 报文解析和构造时标识不同的记录类型。

#define RR\_A 1 // IPv4 地址

#define RR\_CNAME 5 // 别名

#define RR\_SOA 6 // 授权起始

#define RR\_PTR 12 // 反向解析

#define RR\_MX 15 // 邮件交换

#define RR\_TXT 16 // 文本信息

#define RR\_AAAA 28 // IPv6 地址

## dataStruct模块

### 概述

dataStruct定义了 DNS 代理服务器中使用的核心数据结构及相关操作函数，包括 IP 地址链表、ID 转换结构、Trie 树和 LRU 缓存机制。

### IP链表

IP链表模块实现了一个支持多协议（IPv4/IPv6）的地址存储结构，主要功能包括：

​**双栈IP存储**​：通过联合体(union)实现IPv4（4字节）和IPv6（16字节）的兼容存储

​**动态管理**​：采用链表结构灵活管理多个IP地址记录

​**版本标识**​：通过version字段（4/6）明确区分地址类型

#### IP地址结构体

该部分定义了支持 IPv4/IPv6 的链表节点结构，包含地址数据（联合体存储）、IP 版本标识和指向下一个节点的指针。

typedef struct ip\_list\_node {

union {

uint8\_t ipv4[4]; // IPv4地址

uint16\_t ipv6[8]; // IPv6地址

} addr;

int version; // 4 or 6

struct ip\_list\_node\* next; // 下一条IP地址记录

} ip\_list\_node, \* ip\_list;

#### **init\_ip\_list()**

init\_ip\_list 函数参数为链表头指针，用于准备存储域名解析结果的IP地址列表。

void init\_ip\_list(ip\_list head); // 初始化IP链表

#### ipv4\_to\_bytes()

ipv4\_to\_bytes 函数将点分十进制格式的 IPv4 字符串（如 "192.168.1.1"）转换为 4 字节的二进制数组（uint8\_t[4]），用于网络传输。

void ipv4\_to\_bytes(const char\* ipv4\_str, uint8\_t\* ipv4\_bytes); // IPv4字符串转字节数组

#### Ipv6\_to\_bytes()

ipv6\_to\_bytes 函数将标准IPv6字符串（如"2001:db8::1"）转换为8组16位整型数组（uint16\_t[8]），用于网络传输。

void ipv6\_to\_bytes(const char\* ipv6\_str, uint16\_t\* ipv6\_bytes); // IPv6字符串转字节数组

#### hex\_to\_int()

hex\_to\_int 函数将单个十六进制字符（0-9、a-f、A-F）转换为对应的整数值（0-15），用于地址解析。

int hex\_to\_int(char c); // 十六进制字符转整型

### ID转换

ID转换模块用于在DNS代理服务器中维护客户端请求与响应的映射关系，主要功能包括：

​**请求-响应关联**​：存储客户端原始请求的DNS报文、ID和地址信息，确保响应能正确返回给对应客户端。

​**超时管理**​：通过expire\_time字段实现请求记录的自动过期清理，避免内存泄漏。

​**快速查询**​：基于固定大小的数组（ID\_LIST\_SIZE = 128）实现O(1)复杂度的ID查找。

#### ID转换结构体

该部分定义了 DNS 代理的 ID 映射结构，用于关联客户端请求 ID、过期时间、DNS 报文及客户端地址。

typedef struct {

uint16\_t client\_ID; // 客户端ID

int expire\_time; // 过期时间

struct dns\_message\* msg; // DNS报文

int msg\_size; // 报文大小

struct sockaddr\_in client\_addr; // 客户端地址

} ID\_conversion;

#### ID转换结构体

该部分定义了一个固定大小的ID映射表数组，用于存储DNS请求的客户端ID与报文信息的映射关系。

ID\_conversion ID\_list[ID\_LIST\_SIZE]; // ID转换列表

### Trie树

Trie树模块用于高效存储和检索域名，支持快速插入、查询和删除操作，主要功能包括：

​**域名存储**​：将域名按字符拆解为树状结构，支持通配符和子域名匹配。

​**IP地址关联**​：每个有效域名节点（is\_end = 1）关联一个IP链表（IPv4/IPv6）。

​**动态扩展**​：通过预分配的节点池（node\_list[NODE\_NUM]）管理内存，避免频繁动态分配。

#### Trie树结构体

该部分定义了Trie树节点结构，用于高效存储和查询域名，包含结束标志、子节点索引数组和关联的IP地址链表。

typedef struct trie\_node {

int is\_end; // 是否是单词的结尾

uint16\_t next[38]; // 指向下一个节点的指针数组

ip\_list ip\_list; // IP地址链表

} trie\_node, \* trie;

#### Trie树全局变量

该部分定义了一个固定大小的字典树节点数组和当前使用量统计变量，用于高效存储和检索域名数据。

trie\_node node\_list[NODE\_NUM]; // 节点列表

size\_t list\_size; // 节点列表大小

#### init\_trie()

init\_trie 函数初始化字典树结构，参数为根节点指针和节点数组长度，用于构建域名快速检索的基础数据结构。

void init\_trie(trie\_node\* root, int len); // 初始化Trie树

#### get\_index()

get\_index 函数将输入字符映射为字典树中的索引位置（0-37），支持处理域名常用字符（字母数字和连字符）。

int get\_index(char c); // 获取字符在Trie中的索引

#### insert4()

insert4 函数将域名及其对应的IPv4地址（4字节格式）插入字典树，建立快速查询映射关系。

void insert4(char\* domain, uint8\_t\* ip); // 插入IPv4记录

#### insert6()

insert6 函数将域名及其对应的IPv6地址（8组16位格式）插入字典树，扩展支持IPv6的快速检索能力。

void insert6(char\* domain, uint16\_t\* ip); // 插入IPv6记录

#### search()

search 函数在字典树中检索指定域名，若存在则将其关联的IP地址链表挂载到head节点，返回是否查找成功。

int search(char\* domain, ip\_list head); // 搜索域名

### LRU缓存

LRU（Least Recently Used）缓存模块用于高效管理热点域名-IP映射，通过淘汰最久未使用的记录保持缓存新鲜度，主要功能包括：

​**缓存自动更新**​：插入或查询时自动将记录移至链表头部（标记为最新使用）

​**容量控制**​：当缓存达到MAX\_CACHE限制时，自动淘汰链表尾部（最久未使用）节点

​**双协议支持**​：统一存储IPv4（version=4）和IPv6（version=6）记录

#### LRU缓存节点

该部分定义了LRU缓存节点结构，用于存储域名和对应IP地址，并维护双向链表结构以实现LRU淘汰机制。

typedef struct lru\_node {

uint16\_t version; // 4 or 6

char domain[MAX\_SIZE]; // 域名

ip\_list ip\_list; // 域名对应的IP地址链表

struct lru\_node\* prev; // 前驱节点

struct lru\_node\* next; // 后继节点

} lru\_node;

#### LRU缓存全局变量

该部分定义LRU缓存的全局管理变量，包括链表头尾指针和当前缓存数量统计，用于实现LRU淘汰策略。

lru\_node\* lru\_head; // LRU链表头

lru\_node\* lru\_tail; // LRU链表尾

size\_t cache\_size; // 缓存大小

#### init\_cache()

init\_cache 函数初始化LRU缓存结构，重置头尾指针和缓存计数器，准备缓存数据存储。

/\*

+------+ +------+

head---->| |---->| |---->NULL

| head | | tail |

NULL<----| |<----| |<----tail

+------+ +------+

\*/

void init\_cache(); // 初始化LRU缓存

#### update\_cache()

update\_cache 函数插入或更新指定域名的IP记录到LRU缓存头部，并维护链表结构。

/\*

+------+ +----------+ +-----+

| |---->| |---->| |

| head | | new\_node | | ... |

| |<----| |<----| |

+------+ +----------+ +-----+

\*/

void update\_cache(char\* domain, ip\_list head); // 更新缓存

#### search\_cache()

search\_cache 函数在LRU缓存中检索指定域名，若命中则将IP链表挂载到head并返回成功状态。

/\*

+------+ +-----+ +-----+ +-----+ +-----+

| |---->| |---->| |---->| |---->| |

| head | | ... | | cur | | tar | | ... |

| |<----| |<----| |<----| |<----| |

+------+ +-----+ +-----+ +-----+ +-----+

\*/

int search\_cache(char\* domain, ip\_list head); // 查询缓存

#### delete\_cache()

delete\_cache 函数自动移除LRU链表尾部的节点（最久未使用记录），维护缓存容量限制。

/\*

+-----+ +--------+ +------+

| |---->| |---->| |

| ... | | target | | tail |

| |<----| |<----| |

+-----+ +--------+ +------+

\*/

void delete\_cache(); // 删除缓存节点(自动删除最久未使用的)

## dnsStruct模块

dnsStruct模块定义了DNS协议相关的数据结构及操作函数，包括DNS报文结构、资源记录类型、报文解析和构造等功能。该模块是DNS代理服务器的核心协议处理部分。

### DNS报文结构

#### DNS报文整体结构

该部分定义了DNS报文的每个大部分，其中后三个字段一起存放在资源字段中

typedef struct dns\_message {

struct dns\_header\* header; // 报文头

struct dns\_question\* question; // 查询问题

struct dns\_resource\_record\* answer; // 回答

struct dns\_resource\_record\* authority; // 授权

struct dns\_resource\_record\* additional; // 附加信息

} dns\_message;

#### DNS报文头结构

该部分定义了DNS头部信息的具体格式，这个部分是DNS能否正确到达目的地址的关键

typedef struct dns\_header {

uint16\_t id; // 事务id

uint8\_t qr : 1; // 查询/响应标志位

uint8\_t opcode : 4; // 操作码

uint8\_t aa : 1; // 授权回答

uint8\_t tc : 1; // 截断

uint8\_t rd : 1; // 期望递归

uint8\_t ra : 1; // 可用递归

uint8\_t z : 3; // 保留

uint8\_t rcode : 4; // 返回码

uint16\_t ques\_num; // 问题数

uint16\_t ans\_num; // 回答数(一般为3)

uint16\_t auth\_num; // 授权数

uint16\_t add\_num; // 附加数

} dns\_header;

#### DNS查询问题结构

该部分定义了查询报文的具体格式，因每次查询可能包含ipv4、ipv6等多次查询所以需要用链表存放

typedef struct dns\_question {

char\* q\_name; // 查询的域名

uint16\_t q\_type; // 查询类型

uint16\_t q\_class; // 查询类

struct dns\_question\* next; // 下一个问题

} dns\_question;

#### DNS资源记录结构

该部分定义了资源记录的具体格式，因每次查询可能包含ipv4、ipv6等多次资源所以需要用链表存放

typedef struct dns\_resource\_record {

char\* name; // 域名

uint16\_t type; // 类型

uint16\_t class; // 类

uint32\_t ttl; // 生存时间

uint16\_t data\_len; // 数据长度

union ResourceData r\_data; // 资源内容

struct dns\_resource\_record\* next; // 下一个资源记录

} dns\_resource\_record;

#### 资源数据联合体

union ResourceData {

/\* IPv4 \*/

struct {

uint8\_t IP\_addr[4];

} a\_record;

/\* IPv6 \*/

struct {

uint8\_t IP\_addr[16];

} aaaa\_record;

/\* SOA：权威记录的起始 \*/

struct {

char\* MName; // 主服务器域名

char\* RName; // 管理员邮箱

uint32\_t serial; // 版本号

uint32\_t refresh; // 刷新数据间隔

uint32\_t retry; // 重试间隔

uint32\_t expire; // 超时重传时间

uint32\_t minimum; // 默认生存时间

} soa\_record;

/\* cname规范名称记录 \*/

struct {

char\* name;

} cname\_record;

};

### 标志位掩码定义

采用掩码，在每次取相应标志位字段时将其与掩码相与即可取得相应位的标志信息

static const uint16\_t QR\_MASK = 0x8000;

static const uint16\_t OPCODE\_MASK = 0x7800;

static const uint16\_t AA\_MASK = 0x0400;

static const uint16\_t TC\_MASK = 0x0200;

static const uint16\_t RD\_MASK = 0x0100;

static const uint16\_t RA\_MASK = 0x0080;

static const uint16\_t RCODE\_MASK = 0x000F;

### 主要功能函数

#### DNS报文解析函数

获取完整的DNS报文

dns\_addr get\_dns\_message(struct dns\_message\* msg, dns\_addr buffer, dns\_addr start);

获取DNS报文

dns\_addr get\_header(struct dns\_message\* msg, dns\_addr buffer, dns\_addr start);

获取DNS查询问题部分

dns\_addr get\_question(struct dns\_message\* msg, dns\_addr buffer, dns\_addr start);

获取DNS回答部分

dns\_addr get\_answer(struct dns\_message\* msg, dns\_addr buffer, dns\_addr start);

获取域名(支持压缩指针)

dns\_addr get\_domain\_name(dns\_addr buffer, char\* name, dns\_addr start);

#### DNS报文构造函数

构造完整的DNS报文

dns\_addr set\_dns\_message(struct dns\_message\* msg, dns\_addr buffer, struct ip\_list\_node\*\* head, int cnt);

构造DNS报文头

dns\_addr set\_header(struct dns\_message\* msg, dns\_addr buffer, struct ip\_list\_node\*\* head, int cnt);

构造DNS查询问题部分

dns\_addr set\_question(struct dns\_message\* msg, dns\_addr buffer);

构造DNS回答部分

dns\_addr set\_answer(dns\_message\* msg, dns\_addr buffer, ip\_list\_node\*\* head, int cnt);

构造域名(支持压缩指针)

uint8\_t\* set\_domain\_name(dns\_addr buffer, char\* name);

#### 辅助功能函数

从缓冲区读取指定位数的数据并转换字节序

size\_t get\_bits(dns\_addr\* buffer, int num);

向缓冲区写入指定位数的数据并转换字节序

void set\_bits(dns\_addr\* buffer, int len, int value);

判断是否为指针压缩格式

int is\_ptr(uint8\_t\* ptr);

释放DNS报文内存

void free\_message(struct dns\_message\* msg);

## Socket模块

### Socket架构设计

本项目采用UDP Socket实现DNS中继服务器，使用双Socket架构：

• client\_sock：监听客户端DNS查询请求 (端口53)

• server\_sock：与上游DNS服务器通信 (端口53)

### Socket初始化过程 (system.c: init\_socket函数)

#### WSAStartup初始化：

- 版本：Winsock 2.2

- 初始化Windows Socket库

#### Socket创建：

- 协议族：AF\_INET (IPv4)

- 类型：SOCK\_DGRAM (UDP协议)

- 协议：0 (默认UDP协议)

#### 地址结构配置：

client\_addr:

- sin\_family: AF\_INET

- sin\_addr: INADDR\_ANY (监听所有网络接口)

- sin\_port: htons(53) (DNS标准端口)

server\_addr:

- sin\_family: AF\_INET

- sin\_addr: inet\_addr(remote\_dns) (上游DNS服务器IP)

- sin\_port: htons(53)

#### Socket选项设置：

- SO\_REUSEADDR：端口复用，避免"Address already in use"错误

#### Socket绑定：

- bind(client\_sock, client\_addr)：绑定到本地53端口

### 非阻塞I/O配置 (server.c: poll函数)

使用ioctlsocket设置非阻塞模式：

• FIONBIO命令：设置Socket为非阻塞模式

• 优势：单线程处理多个并发连接，提高性能

### 事件驱动机制

使用WSAPoll函数实现事件驱动：

• 监听事件：POLLIN (数据可读)

• 超时设置：5毫秒

• 同时监听：client\_sock和server\_sock

### 数据收发操作

#### 接收数据：recvfrom函数

- 获取数据内容和发送方地址信息

- 用于客户端请求接收和服务器响应接收

#### 发送数据：sendto函数

- 指定目标地址发送数据

- 用于转发请求到上游DNS和响应客户端

### Socket关闭 (server.c: closeServer函数)

• closesocket：关闭Socket连接

• WSACleanup：释放Winsock资源

# 主函数流程与流程图

## 流程图

图示

AI 生成的内容可能不正确。

图示

AI 生成的内容可能不正确。图示

AI 生成的内容可能不正确。图示

AI 生成的内容可能不正确。

## main 函数完整流程分析

### 初始化

**1. 定义默认参数：**

初始化默认的DNS服务器IP server\_ip 为 "10.3.9.6"。

初始化默认的文件名 file\_name 为

"D:\File\Cworks2\dns\_relay\_2022211181\dnsrelay.txt"。

**2. 初始化组件：**

调用 initialize\_all 函数，完成以下步骤：

**设置命令行参数**：

根据命令行参数，设置服务器IP和文件名。

设置调试级别（默认为0，支持-d和-dd）。

**初始化ID表**：

分配内存并初始化ID表，用于存储DNS查询记录。

**初始化缓存**：

创建并初始化LRU缓存，用于存储最近查询的结果。

**初始化本地文件和哈希表**：

尝试打开本地IP-URL映射文件。如果文件存在，从文件中读取IP-URL映射并构建哈

希表。

**初始化套接字**：

创建用于通信的UDP套接字，绑定本地地址和端口。

**启动定时器线程**：

根据操作系统（Windows或Linux），启动定时器线程，用于定期更新时间。

### 主循环

**3. 进入主循环：**

程序进入一个无限循环，以处理接收到的数据包。

**4. 处理超时请求：**

定期调用 findOutOfTime 函数，检查并处理超时请求，清理ID表中已超时的记录。

**5. 清空缓冲区：**

清空用于接收数据的缓冲区 buf ，准备接收新的数据包。

**6. 接收数据包：**

使用 recvfrom 函数接收来自客户端或服务器的数据包，并将发送方的地址信息存储在

tmp\_sockaddr 中。

通过检查DNS报文头中的 qr 字段，判断接收到的是查询请求还是响应包。

### 处理数据包

**7. 处理服务器响应包：**

如果 qr 字段为1，表示这是一个响应包，调用 handle\_server\_packet 函数：

**提取ID并解析域名**：

从响应包中提取ID，并解析出查询的域名。

**查找ID表并转换ID**：

在ID表中查找对应的记录，转换为客户端的ID。如果找不到对应的记录，则丢弃该

响应包并发出警告。

**发送响应包**：

将转换后的响应包发送给客户端。

**更新缓存和本地文件**：

如果是A记录查询，提取IP地址，更新缓存和本地文件中的映射。

**8. 处理客户端查询包：**

如果 qr 字段为0，表示这是一个查询包，调用 handle\_client\_packet 函数：

**提取ID并解析域名**：

从查询包中提取ID，并解析出查询的域名。

**本地查找IP地址**：

在本地哈希表和缓存中查找IP地址，检查是否为禁止URL。

**处理禁止URL**：

如果是禁止URL，生成并发送错误响应包。

**生成并发送响应包**：

如果找到IP地址，生成响应包并发送给客户端。

**转发查询包**：

如果在本地缓存和哈希表中未找到，将查询包转发给DNS服务器，并更新ID表。

**9. 关闭套接字：**

当循环终止时，关闭套接字，释放系统资源（在代码中没有显示，但通常在程序退出时需要执

行）。

## 程序启动流程

### main函数入口

- 调用init(argc, argv)进行系统初始化

- 调用poll()进入主事件循环

- 程序结束时调用closeServer()清理资源

### 系统初始化流程 (system.c: init函数)

a) 设置默认上游DNS服务器：208.67.220.220

b) 解析命令行参数：-d(调试) -l(日志) -i(信息) -s(设置DNS服务器)

c) 打印帮助信息

d) 初始化Socket (端口53)

e) 初始化Trie树节点池

f) 初始化节点链表

g) 初始化ID映射表

h) 初始化LRU缓存

i) 读取host文件到Trie树

## 主事件循环流程 (server.c: poll函数)

无限循环执行以下步骤：

### 超时处理：timeout\_handle()

- 检查ID映射表中的过期请求

- 发送超时响应给客户端

- 清理过期的映射表项

### 事件监听：WSAPoll()

- 同时监听client\_sock和server\_sock

- 5毫秒超时，保证系统响应性

### 事件分发：

- 客户端数据到达 → receiveClient()

- 服务器响应到达 → receiveServer()

## 客户端请求处理流程 (server.c: receiveClient函数)

### 接收DNS查询报文

- recvfrom接收客户端UDP数据包

- 记录客户端地址信息

### DNS报文解析

- 调用get\_dns\_message解析二进制报文

- 提取查询域名、类型、类别等信息

### 三级查找策略

a) 第一级：缓存查找

- 调用search\_cache在LRU缓存中查找

- 命中则直接返回结果

b) 第二级：本地host文件查找

- 调用search在Trie树中查找

- 命中则返回本地配置的IP地址

c) 第三级：外部DNS查询

- 分配新的ID映射表项

- 替换报文ID为代理服务器ID

- 转发请求到上游DNS服务器

### 本地响应处理

- 如果本地查找成功，调用set\_dns\_message组装响应

- 发送响应给客户端

## 服务器响应处理流程 (server.c: receiveServer函数)

### 接收上游DNS响应

- recvfrom接收上游DNS服务器的UDP响应

### DNS响应解析

- 调用get\_dns\_message解析响应报文

### ID转换和路由

- 根据响应中的ID查找ID映射表

- 恢复原始客户端ID和地址信息

- 修改响应报文的ID字段

### 缓存更新

- 如果响应包含A或AAAA记录，更新LRU缓存

- 构建IP地址链表并调用update\_cache

### 本地数据更新

- 检查域名是否已存在于Trie树

- 如果不存在，调用insert\_host添加到本地数据

### 响应转发

- 将修改后的响应发送给原始客户端

- 清理ID映射表项

## 数据结构操作流程

### Trie树操作 (dataStruct.c)

- insert4/insert6：插入IPv4/IPv6域名记录

- search：查找域名对应的IP地址

- 支持域名的前缀匹配和精确查找

### LRU缓存操作 (dataStruct.c)

- search\_cache：在缓存中查找域名

- update\_cache：更新缓存，实现LRU算法

- delete\_cache：删除最久未使用的缓存项

### ID映射表操作 (dnsStruct.c)

- add\_list\_id：分配新的ID映射项

- 超时检查：定期清理过期的映射项

## 错误处理和日志

### 网络错误处理

- Socket操作失败时打印错误信息

- 使用strerror获取详细错误描述

### 调试信息输出

- debug\_mode控制调试信息的显示

- 包括请求来源、目标域名、处理结果等

### 日志记录

- log\_mode控制日志记录功能

- 记录查询历史到log.txt文件

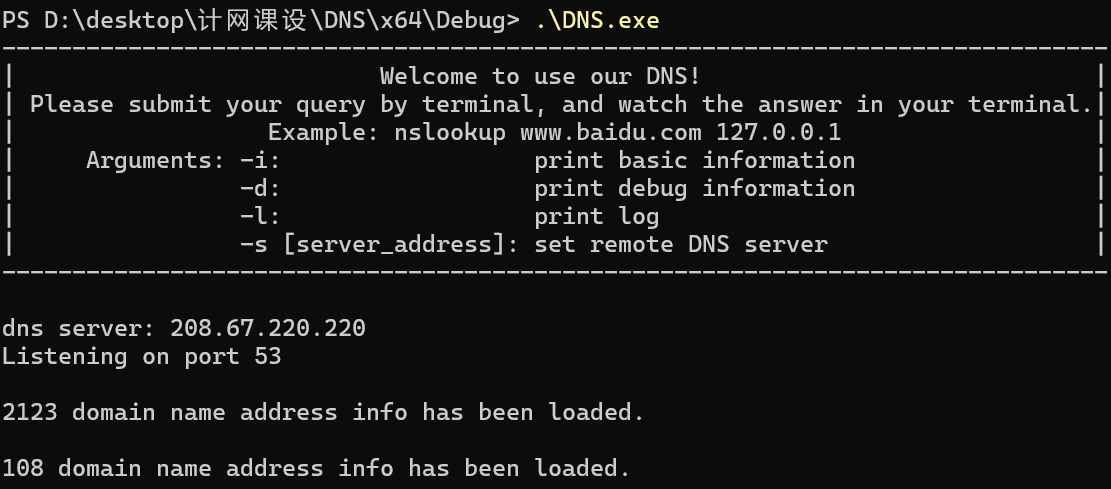
# 测试分析

## 准备工作

设置本机DNS服务器为127.0.0.1如图所示



启动中继器

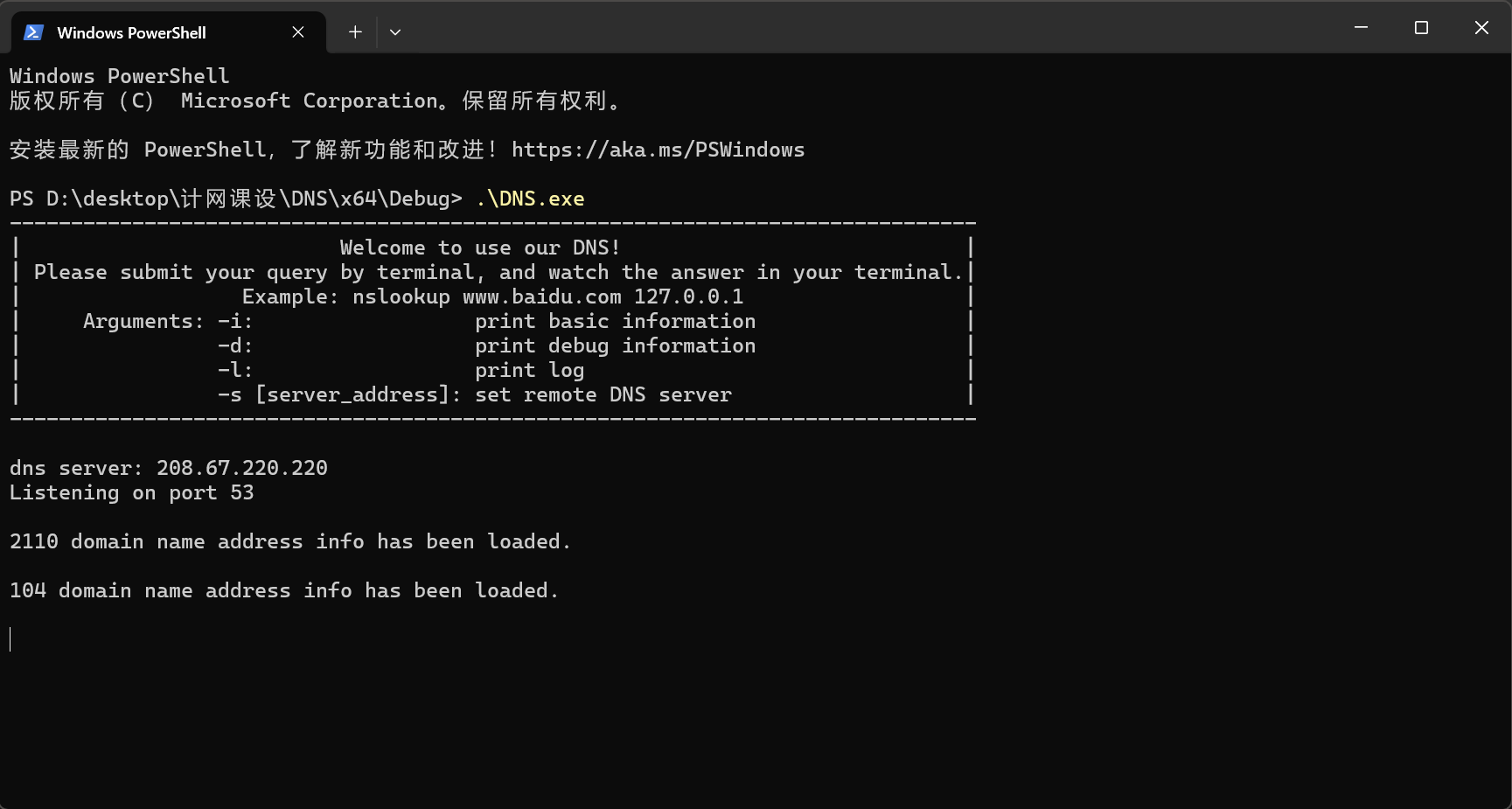


## 拦截

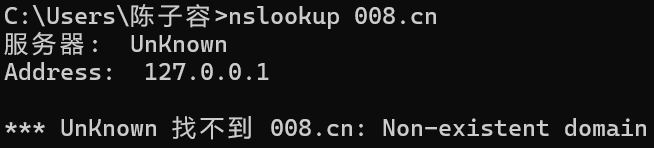
假设本地转换表有如下内容



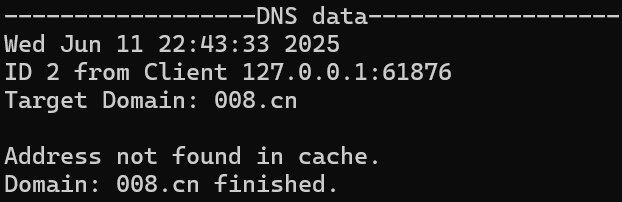
打开中继器程序



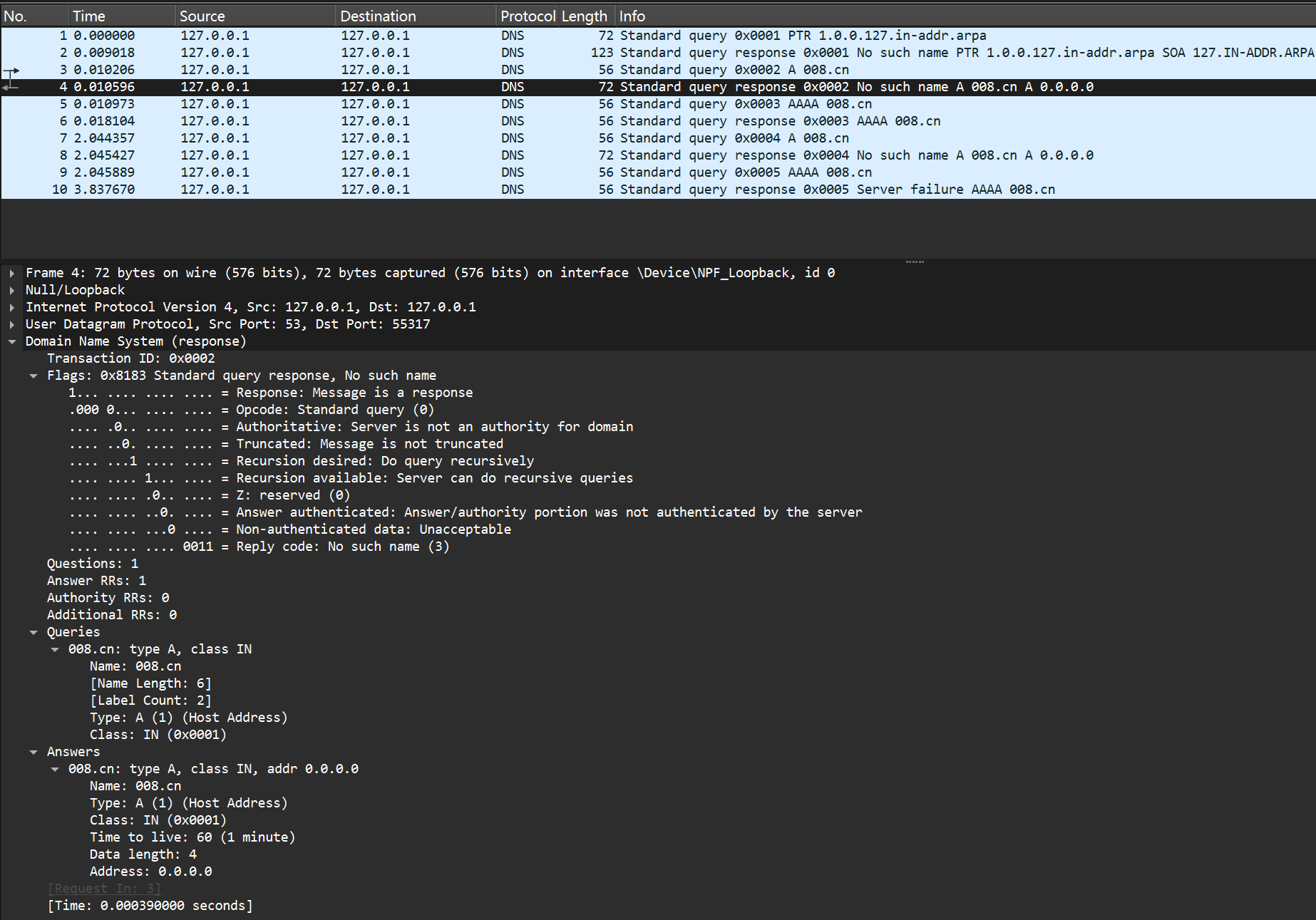
查询008.cn如下



中继器输出如下



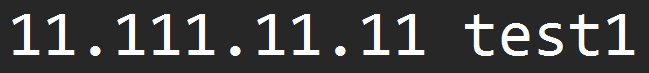
Wireshark监听udp port 53，捕捉本地环路DNS包如下



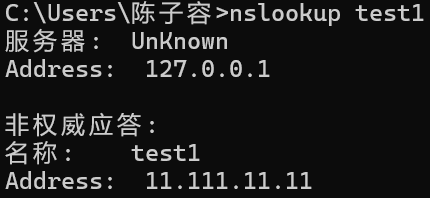
符合预期

## 本地查询

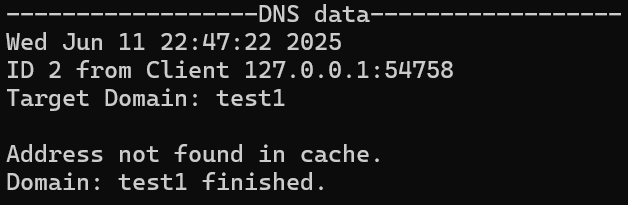
假设本地转换表有如下内容



查询test1如下



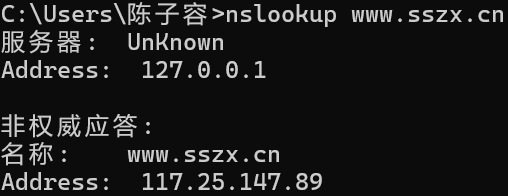
中继器输出如下



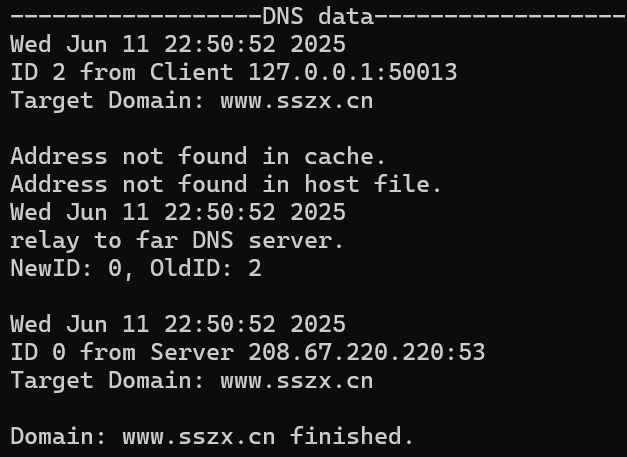
可以看出是从本地查找到的，本地查询功能符合预期

## 中继

查询www.sszx.cn如下



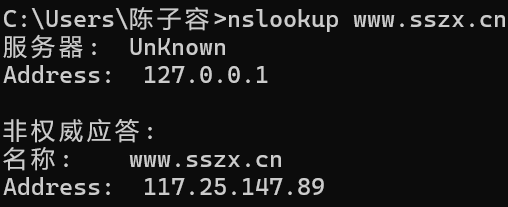
中继器输出如下



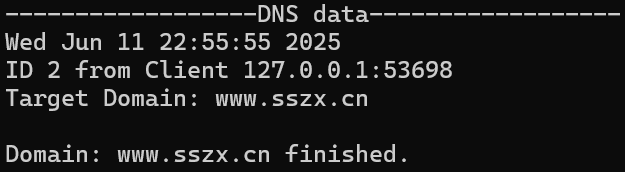
可以看到未从缓存或数据库中查找到该域名，中继功能符合预期

## 缓存

再次查询www.sszx.cn如下



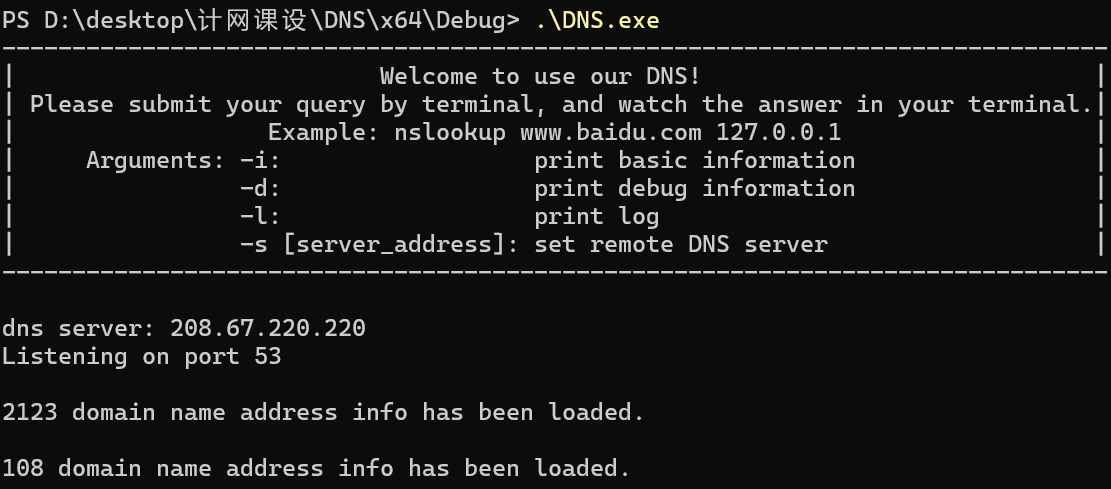
中继器输出如下



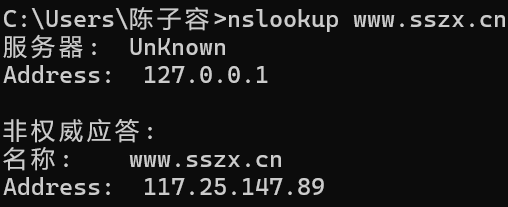
可以看出是从缓存中查找到的，缓存功能符合预期

## 数据库更新

重新启动中继器



再次查询www.sszx.cn如下



中继器输出如下



可以看出是从数据库中查找到的，数据库更新功能符合预期

# 问题分析

## IPV6查询结果错误问题

在测试过程中，我们发现程序输出的IPv6地址有时会出现端序错误，例如期望输出规范的IPv6地址段（如2001），实际却输出了字节错位的错误结果（如120）。经过分析，这是由于程序在处理IPv6地址时未正确进行网络字节序与主机字节序的转换。IPv6地址在DNS报文中以网络字节序（大端序）存储，而主机环境可能采用小端序，直接读取会导致地址段错位。通过调用ntohs/htons等函数对地址字段进行端序转换后，问题得到解决。这一过程让我们深刻认识到网络协议中字节序处理的重要性，尤其是在跨平台开发时需严格遵循RFC规范。

## 文件描述符溢出

在进行长稳测试的过程中，我们发现在不能在本地的查到的域名越来越多时程序会崩溃。在寻找原因后发现是因为打开相应txt文件记录后未关闭文件导致的。

当程序通过fopen()打开文件时，操作系统会为该文件分配一个文件描述符，而文件描述符是有限资源，操作系统对单个进程能同时打开的文件描述符数量有严格限制，默认值通常为1024，如果程序不断打开文件而不关闭，文件描述符就会被耗尽，此时程序就会崩溃。

## IPV6应答数据报抓包结果

在抓包分析AAAA类查询（IPv6地址查询）时，我们发现部分远端DNS服务器会返回CNAME类型的应答报文（即域名别名记录），而当前中继器仅支持直接解析AAAA记录，尚未实现CNAME记录的递归查询功能。例如，当查询example.com的IPv6地址时，服务器可能返回example.com → cname.example.net → 2001:db8::1的链式应答，但中继器会因无法处理CNAME记录从而无法记录该IPv6地址。未来计划扩展协议支持范围，从而支持CNAME记录。

## 总结与收获

通过这次实验，我们团队对DNS协议及其实现机制有了系统而深入的认识。作为计算机网络课程的重要实践环节，我们共同完成了一个功能完善的DNS中继服务器开发项目。在为期三周的开发过程中，我们经历了需求分析、模块设计、编码实现、测试优化等完整的软件开发流程，收获颇丰。

在技术层面，我们不仅掌握了DNS报文的结构解析和构造方法，还深入理解了域名解析的完整流程。通过实现Trie树数据结构，我们优化了域名查询效率；采用LRU缓存机制，显著提升了系统响应速度。在解决IPv6查询异常问题时，我们通过反复分析RFC文档和抓包数据，最终准确定位了字节序转换的错误，这个过程让我们深刻体会到协议标准的重要性。文件描述符泄漏问题的排查则教会我们系统资源管理的必要性。

团队协作方面，我们采用模块化分工方式，通过定期会议和代码评审确保项目进度。陈子容同学负责的数据结构模块为系统提供了高效的数据存储方案；计子毅同学实现的DNS协议解析功能准确可靠；江宝金同学搭建的系统框架稳定易扩展。这种分工合作模式充分发挥了每个成员的优势，也培养了我们的团队协作能力。

这次实验让我们将课堂理论转化为实际工程能力，特别是在网络编程、协议分析和性能优化等方面收获显著。我们不仅掌握了技术知识，更培养了严谨的工程思维和问题解决能力。这些经验将为我们未来的学习和工作奠定坚实基础，也增强了我们解决复杂工程问题的信心。