

北京邮电大学课程设计报告

课程设计名称	计算机网络课程设计		学 院	计 算 机	指导教师	蒋砚军
班 级	班内序号	学 号		学生姓名	成绩	
2021211302	9	2021210949		王浩然		
2021211301	17	2021213586		郭栩源		
课程设计内容	一、教学目的：通过网络编程将在计算机网络这门课程中学到的 DNS 理论知识同其他网络知识综合运用起来。增强学生的获取知识、运用知识的能力，同时提高学生在网络层面上编程的水平，采用小组方式完成课程设计去培养学生的团队合作、分工、沟通的能力。					
	二、课程设计内容：设计一个 DNS 服务器程序，读入“域名-IP 地址”对照表，当客户端查询域名对应的 IP 地址时，用域名检索该对照表，三种检索结果：检索结果为 ip 地址 0.0.0.0，则向客户端返回“域名不存在”的报错消息，而不是返回 IP 地址为 0.0.0.0（不良网站拦截功能）					
	检索结果为普通 IP 地址，则向客户返回这个地址（服务器功能）表中未检测到该域名，则向因特网 DNS 服务器发出查询，并将结果返给客户端（中继功能）					
	考虑多个计算机上的客户端会同时查询，需要进行消息 ID 的转换。					
	三、实验方法：首先明确整个程序运行流程，进而根据流程去进行模块划分，确定号函数接口，从而明确成员分工、编写程序，最后测试、调试、完善程序。					
学生课程设计报告(附页)	四、成员分工					
	王浩然：主要负责编写字典树、包装换函数、ID 表，完成字典树的构建、查找功能、包装换函数中提取 URL、IP 等功能以及 ID 表的转换 ID，完善程序的主体结构，参与到主函数的搭建、调试过程。					
	郭栩源：主要负责编写 LRU 缓冲池-Cache，根据 LRU 算法完成 cache 的构造、添加、更新、删除功能，完善程序的主体结构，参与到主函数的搭建、调试过程。					
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

一：任务描述.....	2
二：开发环境.....	2
三：功能需求.....	2
四：模块划分.....	3
五：测试结果分析.....	24
六：用户使用指南.....	28
七：实验总结.....	30

一：任务描述

设计一个 DNS 服务器程序，读入“域名-IP 地址”对照表，当客户端查询域名对应的 IP 地址时，用域名检索该对照表，三种检索结果：

- 检索结果为 IP 地址 0.0.0.0，则向客户端返回“域名不存在”的报错消息（不良网站拦截功能）；
- 检索结果为普通 IP 地址，则向客户返回这个地址（服务器功能）；
- 表中未检测到该域名，则向因特网 DNS 服务器发出查询，并将结果返给客户端（中继功能）；

此外，考虑多个计算机上的客户端会同时查询，需要进行消息 ID 的转换。

二：开发环境

1. 操作环境：Windows、Ubuntu
2. 工程软件：VisualStudio
3. 编程语言：C

三：功能需求

1. 基本需求：

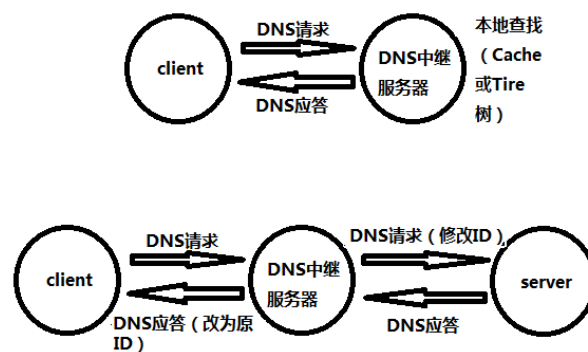
- 1) 解析并构建报文。
- 2) 加载并维护本地的“IP—域名”对照表，用于添加、删除、查询 DNS 记录。
- 3) 获取 DNS 请求；如果在本地能查询到 DNS 请求的对应信息，则将其发回主机。如果本地查询不到 DNS 请求的对应信息，则将其发到指定的外部服务器。

2. 额外需求：

基于老师提供的 PPT 中提到的额外功能，我们全都对其进行了实现。并且针对实用性与高性能的考虑，我们实现了额外的需求。

- 1) 在构建整个程序的时候我们就考虑到 Windows 与 Linux 兼容的问题，并在完成 Windows 版本后更新了代码中的头文件与数据类型就完成了在 Linux 版本的兼容。
- 2) 针对不同的调试等级完善了调试信息，仿照老师提供的测试程序与要求打印了各种调试信息并且能够输出 DNS 报文内容。
- 3) 使用事件驱动方式完成多并发，并针对多台主机同时发来 DNS 请求时重新分配序列号达到区分不同查询的目的。
- 4) 实现了使用 LRU 算法的 Cache，加快了查询效率。
- 5) 实现了命令行参数的解析，可以使得用户选择调试等级、远程服务器地址以及 host 文件。

3. DNS 查询功能的两种大体情况流程示意图



A. 本地查找成功

如果客户端请求的域名对应的 IP 地址在本地 DNS 服务器的 Cache 或 Tire 中，则经过查找后可以直接找到域名对应的 IP 地址并构造响应报文直接返回给客户端

B. 本地查找不成功

如果客户端请求的域名对应的 IP 地址在本地 DNS 服务器中找不到，则选择将对其进行 ID 转换并转发给远程服务器端，待远程服务器端返回报文后根据响应报文中的 ID 进行 ID 逆转换，从而返回给对应的客户端。

四：模块划分

(一) DNS 报文头模块—header.h

```

1 //
2 // Created by 27711 on 2023-06-22.
3 //
4 #ifndef HEADER_H
5 #define HEADER_H
6
7 #define u16 unsigned short
8
9 struct DNS_Header {
10     unsigned id : 16; /* query identification number */
11     unsigned rd : 1; /* recursion desired */
12     unsigned tc : 1; /* truncated message */
13     unsigned aa : 1; /* authoritative answer */
14     unsigned opcode : 4; /* purpose of message */
15     unsigned qr : 1; /* response flag */
16     unsigned rcode : 4; /* response code */
17     unsigned cd : 1; /* checking disabled by resolver */
18     unsigned ad : 1; /* authentic data from named */
19     unsigned z : 1; /* unused bits, must be ZERO */
20     unsigned ra : 1; /* recursion available */
21     u16 qdcount; /* number of question entries */
22     u16 ancount; /* number of answer entries */
23     u16 nscount; /* number of authority entries */
24     u16 arcount; /* number of resource entries */
25 };
26
27 #endif
28

```

这里直接采用老师 PPT 中提供的 DNS 报文头

(二) 全局变量模块—global.h、global.cpp

```

//
// Created by 27711 on 2023-06-22.
//
#ifndef DNS_SERVER_C_GLOBAL_H
#define DNS_SERVER_C_GLOBAL_H

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#ifdef _WIN32
#include <winsock2.h>
#pragma comment(lib, "wsock32.lib")
#endif

#define FLAG_MATCH 0x8180
#define FLAG_NO_MATCH 0x0000
#define FLAG_IP_FOUND 0x0001
#define NAME_OFFSET 0xc00c
#define TYPE_A 0x0001
#define CLASS_A 0x0001
#define TTL 0x10
#define IP_LENGTH 0x0004
#define IP_MAX_SIZE 16
#define URL_MAX_SIZE 200
#define ASCII_SIZE 128
#define LEN 512
#define ID_TABLE_SIZE 128
#define TIME_SIZE 1200
extern int globalTime;
extern int debug_level;
//extern int current_hour, current_minute, current_second, current_millisecond;
//extern SYSTEMTIME system_time;
#include <time.h>
unsigned int __stdcall timeRun(void* );
#endif //DNS_SERVER_C_GLOBAL_H

```

A. 宏定义:

- FLAG 类宏定义: 用于 DNS 报文的 Flag 字段
- NAME_OFFSET: 用于响应报文中的 Answer 的 Name
- TYPE_A: 用于 Answer 中的 Type
- CLASS_A: 用于 Answer 中的 Class
- TTL: 用于 Answer 中的 Time to live
- IP_LENGTH: 用于 Answer 中的 Data length
- IP_MAX_SIZE: IP 数组中的容量
- URL_MAX_SIZE: URL 数组的容量
- ASCII_SIZE: Ascii 码的数量
- LEN: 报文最大长度
- IP_TABLE_SIZE: ID 池的容量
- TIME_SIZE: 模拟时间的最大值

B. 全局变量:

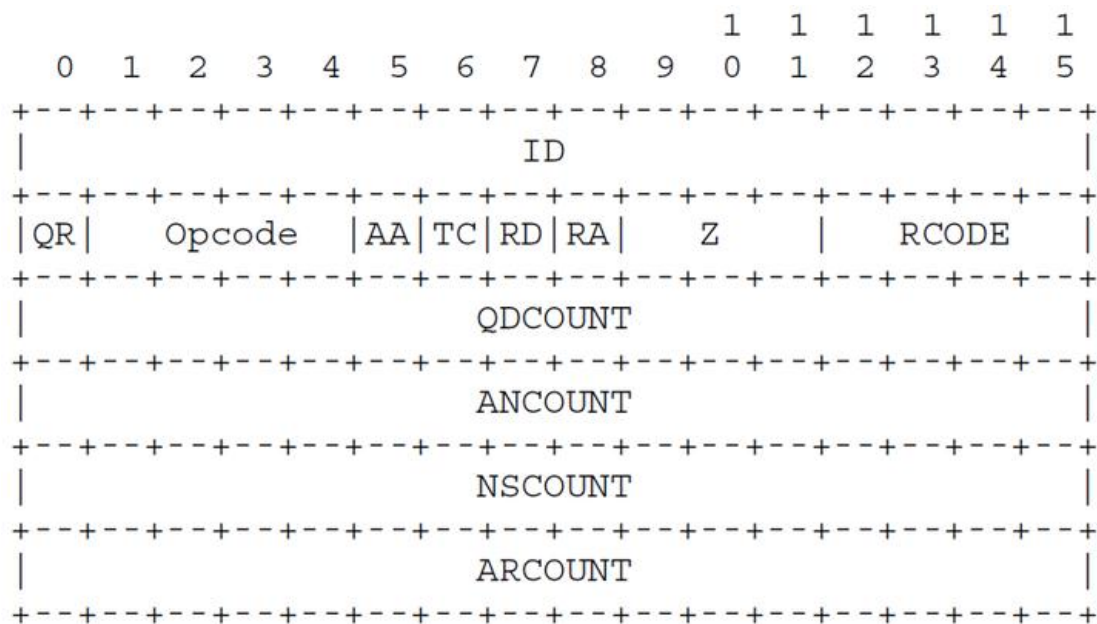
- globalTime: 模拟时间, 用于判定各种模块中的信息是否过期
- debug_level: 调试等级, 用于程序根据调试等级输出不同级别的信息

(三) 报文转换模块—package_Transformer.h、package_Transformer.c

1) 报文格式介绍

A. Header

报文头是不可缺失的, 它定义了报文是请求还是应答, 也定义了其他段是否需要存在, 以及是标准查询还是其他。报文头包含如下字段:



报文头结构

- ID: 由客户程序设置并由服务器返回结果, 客户程序通过 ID 来确定查询与响应

是否匹配

- **QR:** 0 表示查询报文, 1 表示响应报文
- **OPCODE:** 0 - 标准查询 (通常) | 1 - 其他值 (反向查询) | 2 服务器状态请求
- **AA:** 权威答案 (Authoritative answer)
- **TC:** 截断的 (Truncated), 当应答长度超过限制时, 会发生截断, 应答的总长度超过 512 字节时, 只返回前 512 个字节。
- **RD:** 期望递归 (Recursion desired), 查询报中设置, 响应报中返回。用于告诉域名服务器处理递归查询, 如果该位为 0, 且被请求的域名服务器没有权威回答, 就返回一个能解答该查询的其他域名服务器。(之前提到过的迭代查询)
- **RA:** 递归可用 (Recursion Available), 如果域名服务器支持递归查询, 则在响应中将该比特设置为 1。
- **Z:** 保留字段, 为了防止未来为该字段赋予意义时产生错误, 应设为 0。
- **RCODE:** 响应码 (Response coded), 仅用于响应报, 值为 0 表示没有差错; 值为 3 表示域名差错, 从权威域名服务器返回, 表示在查询中指定域名不存在。
- **QDCOUNT:** 表示报文请求段中的问题记录数。
- **ANCOUNT:** 表示报文回答段中的回答记录数。
- **NSCOUNT:** 表示报文授权段中的授权记录数。
- **ARCOUNT:** 表示报文附加段中的附加记录数。

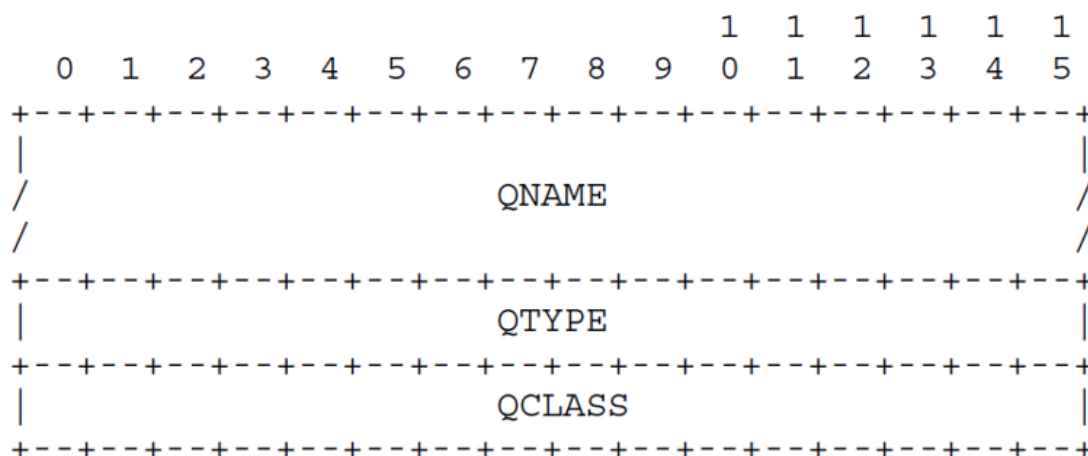
补充: 资源记录 (Resource Record) 用于在 DNS 中提供与查询相关的额外信息。

arcount 字段的存在是为了支持 DNS 的扩展功能和特性。在某些情况下, DNS 查询可能需要返回附加的资源记录, 以提供更多的上下文信息或满足特定的需求。例如, 它可以用于支持 DNSSEC (DNS Security Extensions) 中的数字签名和验证过程, 或者用于提供附加的授权和验证信息。

这些附加的资源记录可以包括与查询相关的附加信息、安全性相关的证书或密钥, 以及其他扩展功能所需的数据。

B. Question

问题段包含着问题, 包含着以下字段:



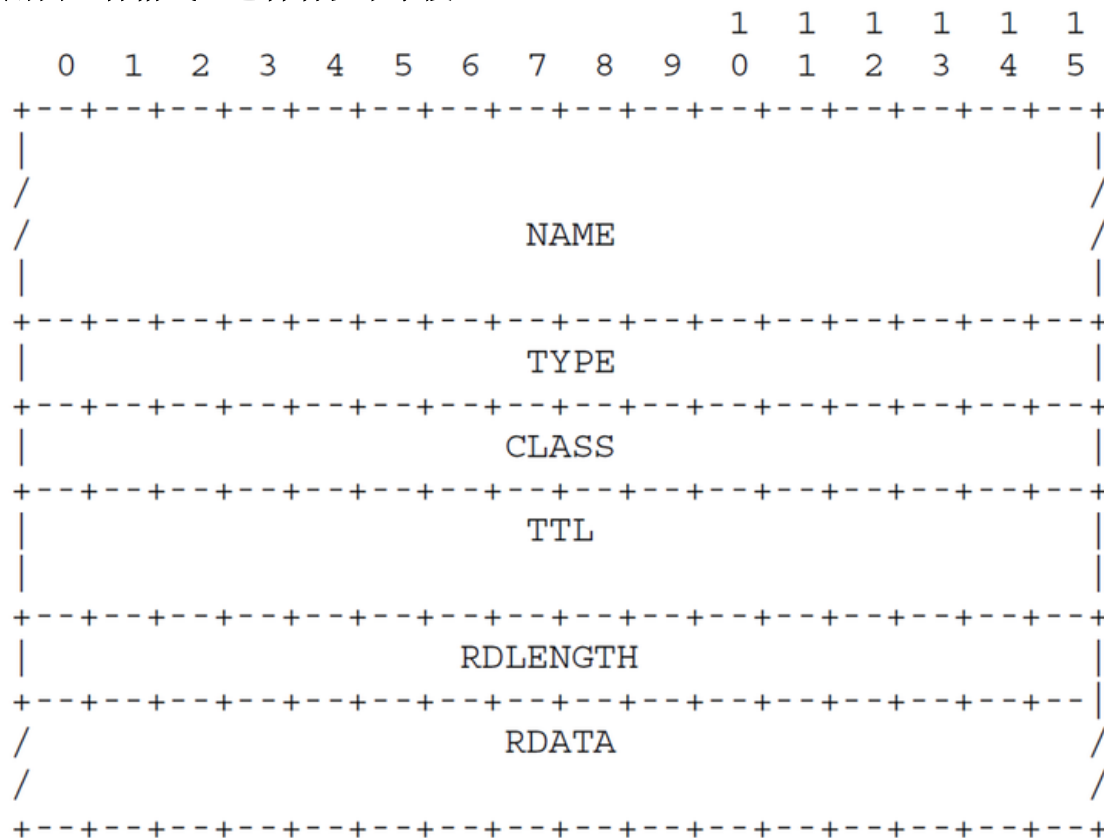
问题段格式

- **QNAME:** 域名, 比如: www.bupt.edu.cn。
- **QTYPE:** 查询类型, 由 2 个字节组成, 例如 A(1), MX(15), CNAME(5) 等。

- **QCLASS:** 查询的协议类型，比如 IN 表示 Internet。

C. Resource Record

资源记录，报文中的应答（Answer）、授权（Authority）、附加段（Additional）共用同一种格式，包含着以下字段：



资源记录格式

- **NAME:** 资源记录包含的域名。
- **TYPE:** 资源记录的类型码，指出 RDATA 数据的含义。
- **CLASS:** 查询的协议类型，通常为 IN，表示 Internet 数据
- **TTL:** 表示可以缓存的时间，数值为客户程序保留该资源记录的秒数，稳定的资源记录生存时间值可以为 2 天，它确定了客户端 DNS cache 可以缓存该记录多长时间。若值为 0 则表示只能被传输，但是不能被缓存。
- **RDLENGTH:** 资源数据（RDATA）长度，说明资源数据的字节数，对类型 1（TYPE A 记录）资源数据是 4 字节的 IP 地址。
- **RDATA:** 资源数据，以不定长字符串来表示记录，具体格式与 TYPE 和 CLASS 有关，例如：TYPE 为 A，CLASS 为 IN，则 RDATA 为 4 个字节的 ARPA 网络地址。

2) 报文处理函数介绍

A. 获取报文长度

```

int GetLengthOfDns(char *DnsInfo)
{
    int length = 12;
    while (DnsInfo[length] != 0) {
        length++;
    }
    length = length + 5;
    return length;
}

```

作用：此函数用于获取 DNS 报文的长度。通过解析报文的格式，从报文的第 12 个字节开始计算报文的总长度，并返回计算得到的长度值。

参数：DnsInfo——指向 DNS 报文的字符数组的指针。该报文用于计算长度。

B. IP 屏蔽

```

//实现屏蔽机制
void ShieldIP(char* DnsInfo, char* curIP) {
    if (curIP && !strcmp(curIP, "0.0.0.0")) {
        struct DNS_Header *error_Header = (struct DNS_Header *)DnsInfo;
        error_Header->qr = 1;
        error_Header->rcode = 3;
        if (debug_level) {
            printf("!!!IP is 0.0.0.0. Sent error message to client.!!!\n");
        }
    }
}

```

作用：此函数用于实现屏蔽机制。如果 IP 地址为"0.0.0.0"，则将响应报文的 qr 字段设置为 1，rcode 字段设置为 3，以发送错误消息给客户端，完成不良网站拦截功能。

参数：

- **DnsInfo:** 指向 DNS 报文的字符数组的指针。该报文用于判断是否需要发送错误消息。
- **curIP:** 当前的 IP 地址，用于判断是否为"0.0.0.0"，进而决定是否发送错误消息。

C. URL 提取

```

//从DNS报文中提取URL
int UrlInDns(char *DnsInfo, char *UrlInDns)
{
    int i = 12, k = 0;
    //DNF报头长12字节，从第12个字节处开始进行读取域名

    while (DnsInfo[i] != 0) {
        int labelLength = DnsInfo[i];
        memcpy(UrlInDns + k, DnsInfo + i + 1, labelLength);
        k += labelLength;
        UrlInDns[k] = '.';
        k++;
        i += (labelLength + 1);
    }

    //设置字符串结束符
    UrlInDns[k - 1] = 0;
    //协议为IPv4则返回1，为IPv6则返回0。这里可以修改为单独返回各个协议对应的数值
    return (!(DnsInfo[i + 2] - 1)) && (!DnsInfo[i + 1]);
}

```

作用：该函数的功能是从 DNS 报文中提取 URL 信息。通过解析 DNS 报文的格式，从报文的第 12 个字节开始读取域名，并将提取的 URL 存储在 UrlInDns 数组中。函数内部使用了循环和指针操作来读取报文中的域名，并在每个标签之间添加.作为分

隔符。

参数:

- **DnsInfo:** 指向 DNS 报文的字符数组的指针。该报文包含了要提取 URL 信息的数据。
- **UrlInDns:** 用于存储提取的 URL 信息的字符数组。在函数执行完毕后，该数组将包含提取的 URL。

D. IP 提取

```
//对于含ipv4地址的应答报文, 提取其ipv4地址
int IpInDns(char *DnsInfo, char *ipInDns)
{
    int i = 12;
    int QuestionNum = DnsInfo[5];
    int AnswerNum = DnsInfo[7];

    // 跳过问题部分的条目
    for (int queryNum = 0; queryNum < QuestionNum; queryNum++) {
        while (DnsInfo[i] != 0) {
            int length = DnsInfo[i];
            i += length + 1;
        }
        i += 5; // 跳过QTYPE和QCLASS字段
    }

    // 计算IP地址的偏移量
    //12是Name(2字节)+类型(2字节)+类(2字节)+TTL(4字节)+数据长度(2字节)
    for (int answerNum = 0; answerNum < AnswerNum; answerNum++) {
        //如果是ipv4地址
        if ((unsigned short)(DnsInfo[i + 3]) == 1) {
            NumIP_To_CharIp(&DnsInfo[i + 12], ipInDns);
            return 1;
        } else {
            i = i + 12 + DnsInfo[i + 11];
        }
    }

    //回应报文中不包含ipv4地址
    ipInDns = NULL;
    return 0;
}
```

作用: 此函数用于从 DNS 报文中提取 IPv4 地址信息。通过解析报文的格式, 跳过问题部分的条目, 并计算出 IPv4 地址的偏移量。然后, 检查偏移量处的字段, 如果是 IPv4 地址, 则将其转换为点分十进制形式的字符串, 并存储在 ipInDns 数组中。

参数:

- **DnsInfo:** 指向 DNS 报文的字符数组的指针。该报文包含了要提取 IP 地址信息的数据。
- **ipInDns:** 用于存储提取的 IPv4 地址的字符数组。在函数执行完毕后, 该数组将包含提取的 IPv4 地址。

E. 地址类型变换

```

void NumIP_To_CharIp(char *NumStart, char *ipInDns)
{
    int i, k = 0;
    unsigned char IpAddress;
    for (i = 0; i < 4; i++) {
        IpAddress = NumStart[i];
        //将无符号整数按照十进制输出
        k += sprintf(&ipInDns[k], "%u", IpAddress);
        ipInDns[k] = '.';
        k++;
    }
    ipInDns[k - 1] = 0;
}

```

作用：该函数用于将数值形式的 IP 地址转换为点分十进制形式的字符串。函数接收一个指向数值形式 IP 地址的指针，并使用 `sprintf` 函数按照十进制格式将每个字节转换为字符串，并在每个字节后添加.作为分隔符。

参数：

- **NumStart:** 指向数值形式的 IP 地址的字符数组的指针。该地址是一个无符号整数。
- **ipInDns:** 用于存储转换后的点分十进制形式的 IP 地址的字符数组。在函数执行完毕后，该数组将包含转换后的 IP 地址

F. 创建响应报文

```

74 int CreateResponse(char *DnsInfo, int DnsLength, char *FindIp,
75                  char *DnsResponse)
76 {
77     //按照客户端发来的查询报文去构建响应报文
78     int LengthOfDns = GetLengthOfDns(DnsInfo);
79     memcpy(DnsResponse, DnsInfo, LengthOfDns);
80
81     //这里设置报文的标志位为0x8180或0x0000, 前者代表正确找到IP地址并返回, 后者代表没找到IP地址返回一个空回答
82     unsigned short a = (DnsLength == LengthOfDns) ? htons(FLAG_MATCH)
83                       : htons(FLAG_NO_MATCH);
84     memcpy(&DnsResponse[2], &a, sizeof(unsigned short)); // 修改标志位
85
86     //找到的IP为0代表屏蔽, 设置回答数为0
87     if (strcmp(FindIp, "0.0.0.0") == 0) {
88         a = htons(FLAG_NO_MATCH);
89     } else {
90         a = htons(FLAG_IP_FOUND);
91     }
92     memcpy(&DnsResponse[6], &a, sizeof(unsigned short));
93
94     int curLen = 0;
95     char answer[16];
96
97     //构造host响应的各个字段
98     unsigned short Name = htons(NAME_OFFSET);
99     unsigned short TypeA = htons(TYPE_A);
100    unsigned short ClassA = htons(CLASS_A);
101    unsigned long timeLive = htons(TTL);
102    unsigned short IPLen = htons(IP_LENGTH);
103    unsigned long IP = (unsigned long)inet_addr(FindIp);
104
105    memcpy(answer + curLen, &Name, sizeof(unsigned short));
106    curLen += sizeof(unsigned short);
107    memcpy(answer + curLen, &TypeA, sizeof(unsigned short));
108    curLen += sizeof(unsigned short);
109    memcpy(answer + curLen, &ClassA, sizeof(unsigned short));
110    curLen += sizeof(unsigned short);
111    memcpy(answer + curLen, &timeLive, sizeof(unsigned long));
112    curLen += sizeof(unsigned long);
113    memcpy(answer + curLen, &IPLen, sizeof(unsigned short));
114    curLen += sizeof(unsigned short);
115    memcpy(answer + curLen, &IP, sizeof(unsigned long));
116    curLen += sizeof(unsigned long);
117
118    memcpy(DnsResponse + LengthOfDns, answer, sizeof(answer));
119
120    if (debug_level == 2) {
121        printf("CreateResponse Success---");
122        //打印各个字段的值想不到除了挨个printf之外有什么好办法完成自动化打印, 索性就不显示名称了
123        /*printf("answer: ");
124        for (int i = 0; i < sizeof(answer); i++) {
125            printf("%02X ", answer[i]);
126        }
127        printf("\n");*/
128    }
129
130    return curLen + LengthOfDns;
131 }

```

作用：此函数用于根据客户端发来的查询报文构建响应报文。首先根据查询报文的长度复制一部分报文到响应报文中，并根据匹配结果设置标志位。然后，根据找到的 IP 地址情况，设置回答数和构建回答部分。最后，将构建好的响应报文存储在 DnsResponse 数组中。

参数：

- **DnsInfo:** 指向客户端发来的查询报文的字符数组的指针。该报文用于构建响应报文。
- **DnsLength:** 查询报文的长度。
- **FindIp:** 要添加到响应报文中的 IPv4 地址的点分十进制形式的字符串。
- **DnsResponse:** 用于存储构建的响应报文的字符数组。在函数执行完毕后，该数组将包含构建的响应报文。

(四) ID 转换模块—ID_Transformer.h、ID_Transformer.c

1) ID 表的初始化

```
//初始化ID表
void init_Table(table_ID_Ptr tableId) {
    tableId->records = (record_Ptr)malloc(sizeof(record) * ID_TABLE_SIZE);
    tableId->size = 0;
    for (int i = 0; i < ID_TABLE_SIZE; i++) {
        tableId->records[i].url = (char*)malloc(sizeof(char) * URL_MAX_SIZE);
        tableId->records[i].Len = 0;
        tableId->records[i].question_Id = 0;
        tableId->records[i].finished = TRUE;
        tableId->records[i].time = -1;
        memset(&(tableId->records[i].client_address), 0, sizeof(SOCKADDR_IN));
    }
}
```

功能：该函数用于初始化 ID 表。

用途：在使用 ID 转换功能前，需要先初始化 ID 表，该表用于存储 ID 转换相关的记录和信息。初始化时，分配内存空间并将各个记录的初始值设置为默认值。

参数：tableId——指向 ID 表的指针。该表用于存储 ID 转换相关的记录和信息

2) ID 逆转换

```
//从服务器端收到报文后进行ID转换
record_Ptr search_ID(table_ID_Ptr tableId, unsigned short ID)
{
    //out_Of_Time(tableId);

    if (tableId->records[ID - 1].finished) {
        return NULL;
    } else {
        if (debug_level == 2) {
            printf("search_ID: ID is %d\n", ID);
        }

        tableId->records[ID - 1].finished = TRUE;
        return &tableId->records[ID - 1];
    }
}
```

功能：该函数用于根据给定的 ID 在 ID 表中查找相应的记录。

用途：在服务器端接收到客户端发送的报文后，需要通过该函数找到对应的 ID 记录，以便进行 ID 转换或获取报文信息

参数：

- tableId: 指向 ID 表的指针。该表用于存储 ID 转换相关的记录和信息。
- ID: 要搜索的 ID 值。

3) ID 转换、保存

```
//从客户端收到报文后进行ID转换并保存
unsigned short save_ID(table_ID_Ptr tableId, char *buf, unsigned short ID,
                      SOCKADDR_IN client_addr)
{
    unsigned short transID = 0;

    // 寻找空闲的记录或插入新记录

    int curIndex = tableId->size;

    /*if (curIndex ≥ ID_TABLE_SIZE) {
        return -1;
    }*/

    do {
        //超时就将ID释放去给其他报文使用, 找到一个能放ID的位置就直接放。没必要遍历, 在找的时候稍带刷新
        if ((globalTime - tableId->records[curIndex].time + TIME_SIZE) % TIME_SIZE > TTL) {
            tableId->records[curIndex].finished = TRUE;
        }

        if (tableId->records[curIndex].finished) {
            // 更新记录的值(根据RFC文档可知ID为0被视为无效或保留的值, 有特殊目的, 所以transID必须从1开始)
            transID = (unsigned short)(curIndex + 1);

            if (debug_level == 2) {
                printf("Client_save_ID: old ID is:%d, new ID is:%d\n",
                      ID, transID);
            }

            // 更新转发包的 ID
            memcpy(buf, &transID, sizeof(unsigned short));
            record_Ptr record_ptr = &tableId->records[curIndex];

            //只处理IPv4信息
            if (UrlInDns(buf, record_ptr->url))
            {
                record_ptr->Len =
                    strlen(record_ptr->url);
            } else{
                record_ptr->Len = 0;
            }

            //把老旧的信息刷新掉, 找到可放的index后就直接退出循环, 没必要继续找位置了
            record_ptr->question_Id = ID;
            record_ptr->client_address = client_addr;
            record_ptr->finished = FALSE;
            record_ptr->time = globalTime;
            curIndex = (curIndex + 1 + ID_TABLE_SIZE) % ID_TABLE_SIZE;
            break;
        }

        curIndex = (curIndex + 1 + ID_TABLE_SIZE) % ID_TABLE_SIZE;
    } while (curIndex ≠ tableId->size);

    //从当前位置继续给接下来要到来的ID进行ID转换
    tableId->size = curIndex;

    return transID;
}
```

功能: 该函数用于进行 ID 转换并保存相关信息。

用途: 在客户端接收到服务器端返回的报文后, 客户端需要对其中的 ID 进行转换, 并保存相关信息到 ID 表中。然后客户端可以使用转换后的 ID 和 ID 表中的信息来处理接收到的报文

参数:

- **tableId:** 指向 ID 表的指针。该表用于存储 ID 转换相关的记录和信息。
- **buf:** 指向接收到的报文数据的字符数组的指针。该报文数据用于进行 ID 转换并更新。
- **ID:** 接收到的报文的 ID 值。
- **client_addr:** 报文的客户端地址

(五) 字典接口模块——tire.h、tire.c

1) 查找指定域名

```
#include "tire.h"
TirePtr searchWord(TireNode *root, const char *word)
{
    TirePtr current = (TireNode *)malloc(sizeof(TireNode));
    current = root;
    int len = strlen(word);

    for (int i = 0; i < len; i++) {
        int index = word[i];
        if (current->children[index] == NULL) {
            if (debug_level == 2) {
                printf("searchWord: can't search the word\n");
            }
            return NULL;
        }
        current = current->children[index];
    }
    return current;
}
```

参数:

- **root:** 指向 Trie 树根节点的指针。
- **word:** 要搜索的单词。

功能: 在 Trie 树中搜索指定的单词。

用途: 在 Trie 树中搜索指定的单词, 用于查找给定单词是否存在于 Trie 树中。

2) 根据域名寻找 IP 地址

```

//由域名寻找IP
char *findIP(char *domain, TirePtr dnsRoot)
{
    if (dnsRoot == NULL)
        return NULL;

    TirePtr curNode = searchWord(dnsRoot, domain);

    if (curNode != NULL) {
        if (debug_level == 2) {
            printf("searchWord: domain is%s, IP_Address is%s\n",
                domain, curNode->IP_Address);
        }
        return curNode->IP_Address;
    }
    return NULL;
}

```

参数:

- **domain:** 要查找 IP 的域名。
- **dnsRoot:** 指向 Trie 树根节点的指针。

功能: 根据域名在 Trie 树中查找相应的 IP 地址。

用途: 通过给定的域名, 在 Trie 树中查找相应的 IP 地址。该函数可用于 DNS 代理中, 将域名映射为 IP 地址。

3) 构建字典树

```

TirePtr bulidTire(FILE *file)
{
    TirePtr dnsRoot = (TirePtr)malloc(sizeof(TireNode));
    dnsRoot->IP_Address = NULL;
    dnsRoot->children = (TirePtr *)malloc(sizeof(TirePtr) * ASCII_SIZE);
    memset(dnsRoot->children, 0, sizeof(TirePtr) * ASCII_SIZE);

    TirePtr curNode = (TireNode *)malloc(sizeof(TireNode));
    char url[URL_MAX_SIZE];
    char ip[IP_MAX_SIZE];

    while (fscanf(file, "%s %s", ip, url) > 0) {
        //构造字符串路径
        //curNode = insertWord(dnsRoot, url);
        curNode = dnsRoot;
        int len = strlen(url);

        for (int i = 0; i < len; i++) {
            //这里不能使用url[i] - 'a', 因为有的域名里面含有数字、特定字符等
            //使用字符的ascii码值作为下标
            int index = url[i];

            //如果字符串路径不存在在此基础上继续构造路径
            if (curNode->children[index] == NULL) {
                curNode->children[index] =
                    (TirePtr)malloc(sizeof(TireNode));
                curNode->children[index]->IP_Address = NULL;
                curNode->children[index]->children =
                    (TirePtr *)malloc(sizeof(TirePtr) *
                        ASCII_SIZE);
                memset(curNode->children[index]-
                    >children,
                    0, sizeof(TirePtr) * ASCII_SIZE);
            }
            curNode = curNode->children[index];
        }

        //给每个字符串路径与IP地址之间建立映射
        if (curNode->IP_Address == NULL) {
            curNode->IP_Address =
                (char *)malloc(sizeof(char) * (strlen(ip) + 1));
            memcpy(curNode->IP_Address, ip, strlen(ip) + 1);
            if (debug_level == 2) {
                printf("Building Tire: URL is %s, ip is %s\n",
                    url, ip);
            }
        }
        else {
            free(curNode->IP_Address);
            curNode->IP_Address =
                (char *)malloc(sizeof(char) * (strlen(ip) + 1));
            memcpy(curNode->IP_Address, ip, strlen(ip) + 1);

            if (debug_level == 2) {
                printf("The URL is: %s, old IP_Address: %s\n",
                    url, curNode->IP_Address, ip);
                printf("\n\treplace successfully.\n");
            }
        }
    }
}

```

参数:

- **file:** 指向包含 IP 地址和域名对应关系的文件指针。

功能: 构建 Trie 树, 并将 IP 地址与域名之间建立映射关系。

用途: 根据给定的文件内容, 构建 Trie 树, 并将域名与 IP 地址之间建立映射关系。
该函数在 DNS 代理中用于构建域名与 IP 地址的映射关系。

4) 创建字典树的结点

```
TireNode *createNode()
{
    TireNode *node = (TireNode *)malloc(sizeof(TireNode));
    if (node) {
        node->ascii = '\0';
        node->isEndOfWord = false;
        for (int i = 0; i < ASCII_SIZE; i++) {
            node->children[i] = NULL;
        }
    }
    return node;
}
```

功能: 创建一个 Trie 节点。

用途: 用于创建一个 Trie 节点, 分配内存并初始化节点的相关字段。

5) 插入结点

```
TireNode *insertWord(TireNode *root, const char *word)
{
    TireNode *current = root;
    int len = strlen(word);
    for (int i = 0; i < len; i++) {
        int index = word[i];
        if (current->children[index] == NULL) {
            current->children[index] = createNode();
            current->children[index]->ascii = word[i];
        }
        current = current->children[index];
    }
    current->isEndOfWord = true;
    return current;
}
```

参数:

- **root:** 指向 Trie 树根节点的指针。
- **word:** 要插入的单词。

功能: 将单词插入到 Trie 树中。

用途: 将指定的单词插入到 Trie 树中, 并返回最后一个字符所在的节点。该函数构建 Trie 树时使用。

6) 释放字典树

```

void freeTrie(TireNode *root)
{
    if (root) {
        for (int i = 0; i < ASCII_SIZE; i++) {
            freeTrie(root->children[i]);
        }
        free(root);
    }
}

```

参数：

- **root**: 指向 Trie 树根节点的指针。

功能：释放 Trie 树的内存空间。

用途：在程序结束时，调用该函数来释放 Trie 树所占用的内存空间，防止内存泄漏。

7) 优缺点总结

优点：

1. **高效的字符串检索**：Trie 树以字符串的前缀作为路径进行存储和检索，因此在查找字符串时具有较高的效率，通常可以在 $O(m)$ 的时间复杂度内完成，其中 m 是要查找的字符串的长度。相比于线性搜索或哈希表等其他数据结构，Trie 树通常具有更快的检索速度。
2. **前缀匹配**：Trie 树可以方便地进行前缀匹配操作。通过遍历以给定前缀开头的路径，可以快速找到具有相同前缀的所有字符串。
3. **空间效率**：尽管 Trie 树需要额外的空间来存储节点和指针，但在实际应用中，由于共享相同前缀的字符串可以共享相同的节点，因此 Trie 树在存储具有重复前缀的字符串时可以节省空间。

缺点：

1. **空间消耗**：Trie 树使用了大量的指针和节点来存储字符串，这导致了较高的空间消耗。尤其是对于大型字母表和长字符串的情况下，Trie 树的空间占用可能会很大。
2. **构建时间复杂度**：构建 Trie 树需要遍历每个字符串，并按照字符逐层创建节点，因此构建的时间复杂度较高。

(六) 缓存模块——cache.cpp, cache.h

1) DNSCache 功能介绍

DNS 域名系统给应用访问带来了额外的时延，另外由于 DNS 域名解析采用不可靠的 UDP 协议通讯，受内外部网络环境的影响较大，特别是在有丢包的情况下，导致的时延可能达到数秒。为缓解此问题，DNS 解析采用了缓存机制。

在客户第一次访问之后，递归服务器和客户端都会缓存到该域名的解析记录，并设置相应的缓存生存时间（TTL），在 TTL 有效期内，客户再次对同域名发起访问时，直接通过客户端本地和本地 DNS 服务器高速缓存解析，不再需要经过迭代查询过程。DNS 缓存可极大提升 DNS 域名解析的效率，一定程度上减少了客户端到用户之间环境对 DNS 域名解析的影响。

2) DNSCache 数据结构

```

#define cache_capacity 30

typedef struct Cache_record {
    char *url; // 域名
    char *ip;
    int time_stamp;
    struct Cache_record *next;
} cache_record;

typedef struct Cache {
    cache_record *head;
    cache_record *tail;
    int size;
} cache, *CachePtr;

```

参数:

cache_capacity: DNS 缓存容量。可以通过修改宏来修改。

struct Cache_record: DNS 缓存表项。DNS 表项包括一组相对应的域名和 IP 地址、时间戳、指向下一表项的指针。

struct Cache: DNS 缓存表。DNS 缓存表以带有头节点的链表形式存储，同时存储尾指针和表的大小。根据 LRU 算法的思想链表头部存储的是最近最少使用的表项，尾部存储的最新使用的表项。

3) DNScache 函数

DNScache 需主要实现的功能有：查找表项，插入表项，刷新。在 DNS.c 中分别定义了 **cache_search**，**cache_insert** 和 **cache_flash** 这三个函数来实现上述功能。

cache_search:

功能：根据输入的 url 在 dns 缓存表中寻找对应的 IP

参数:

- url——要查找的表项的 url
- Cache——执行查找操作的 Cache

返回值：与 url 对应的 IP


```

// 在DNS缓存中查找指定URL对应的IP地址（采用LRU算法）
char *cache_search(char *url, cache *Cache)
{
    // 刷新缓存，清除过期的记录
    cache_flash(Cache);

    char *ip = NULL;
    cache_record *pre, *cur;
    pre = Cache->head;
    cur = Cache->head->next;
    while (cur != NULL) {
        // 如果URL匹配，则返回对应的IP地址
        if (strcmp(url, cur->url) == 0) {
            ip = cur->ip;

            // 将当前记录移到链表末尾（LRU算法）
            if (cur->next != NULL) {
                pre->next = cur->next;
                Cache->tail->next = cur;
                Cache->tail = Cache->tail->next;
                cur->next = NULL;
            }
            break;
        }
        pre = cur;
        cur = cur->next;
    }
    return ip;
}

```

思想：查找前首先刷新一遍 **cache**，清除过期表项。然后从 **DNScache** 表头开始查找，如果找到 **url** 和对应对应 **IP**，则将该表项移至链表尾部，然后返回 **IP**。若未查询到，则返回 **NULL**。

性能分析：由于每次从链表头部开始查询，所以理论时间复杂度是 $O(N)$ ，但是，由于每次查询前先执行刷新操作，所以实际上在 $O(N)$ 的时间内能够执行刷新和查找操作，程序效率高于 $O(N)$ 。

cache_insert:

功能：在 dns 缓存表中插入一个<url,ip>的键值对

参数：

- url——要插入的表项的 url
- IP——要插入的表项的 IP
- Cache 执行插入操作的 cache

返回值：NULL

```

// 向DNS缓存中插入新的记录 (采用LRU算法)
char *cache_insert(char *url, char *ip, cache *Cache)
{
    // 创建新的缓存记录, 并复制URL和IP地址
    cache_record *tmp_record = (cache_record *)malloc(sizeof(cache_record));
    tmp_record->url = (char *)malloc(sizeof(char) * URL_MAX_SIZE);
    memcpy(tmp_record->url, url, strlen(url) + 1);
    tmp_record->ip = (char *)malloc(sizeof(char) * IP_MAX_SIZE);
    memcpy(tmp_record->ip, ip, strlen(ip) + 1);
    tmp_record->time_stamp = globalTime;

    // 将新记录插入到链表末尾, 并更新尾指针 (LRU算法)
    Cache->tail->next = tmp_record;
    Cache->tail = Cache->tail->next;
    tmp_record->next = NULL;

    // 输出调试信息
    if (debug_level == 2) {
        printf("Record is inserted to cache:url = %s, ip = %s, timestamp = %d\n",
            tmp_record->url, tmp_record->ip, tmp_record->time_stamp);
    }

    // 增加缓存大小并刷新缓存, 以清除过期的记录
    Cache->size++;
    cache_flash(Cache);

    return NULL;
}

```

思想：首先根据传入的 url 和 IP 创建新的 cache 表项，然后将该表项插入链表末尾，并增加链表大小，最后刷新 cache，删除过期表项与超出 cache 容量的表项。

性能分析：由于采用尾插法，只需 $O(1)$ 的时间即可进行插入操作。

cache_flash:

功能：刷新 cache 里面过期内容

参数：

- Cache：被刷新的 cache

```

// 刷新DNS缓存, 清除过期的记录
void cache_flash(cache *Cache)
{
    cache_record *tmp_record;
    while (Cache->size > 0 && (globalTime - Cache->head->next->time_stamp +
        TIME_SIZE) % TIME_SIZE >
        TTL) {
        // 移除过期的记录并释放相关内存
        tmp_record = Cache->head->next;
        Cache->head->next = Cache->head->next->next;
        free(tmp_record->url);
        free(tmp_record->ip);
        free(tmp_record);
        Cache->size--;

        // 如果链表变为空, 则更新尾指针
        if (Cache->head->next == NULL) {
            Cache->tail = Cache->head;
        }
    }

    int i = 0;
    // 如果缓存大小超过限制, 则逐个移除记录直到缓存大小符合要求 (LRU算法)
    while (Cache->size > cache_capacity) {
        i++;
        tmp_record = Cache->head->next;
        Cache->head->next = Cache->head->next->next;
        free(tmp_record->url);
        free(tmp_record->ip);
        free(tmp_record);
        Cache->size--;
    }

    // 打印缓存中的记录 (用于调试)
    if (debug_level == 2) {
        printf("Cache is flashed:\n");
        printCache(Cache);
    }
}

```

思想: **cache** 刷新的流程分两步: 第一步是删除所有 **TTL** 到期的表项, 第二步是在所有到期表项被删除后, 假设剩余表项的数目仍大于 **cache** 容量, 则删除最近最少使用的 (位于表头部分的表项。这两个步骤用两个循环实现。在第一步中, 由于在链表中, 表项按照生存期从旧到新排序, 所以只需要在链表头部一次检验每个表项是否到期, 到期则删去; 若表头的表项未到期, 则后续表项也不可能到期, 所以跳出循环。在第二步中, 只需比较 **cache** 中表项数目和 **cache** 容量, 若表项数目大于容量, 则从表头删除一个表项; 否则跳出循环。

性能分析: 对任意一个表项, 使用头删法, 以 $O(1)$ 的时间复杂度即可删除它; 假设在一次刷新过程中有 M 个表项需要删除, 则进行一次刷新的所需的时间为 $O(M)$ 。

(七) 主函数模块——Main.c

A. 整体流程:

1. 初始化变量和数据结构：
 - 初始化套接字和地址变量。
 - 定义并初始化其他变量，如套接字大小、服务器 IP 地址、调试等级。
 - 初始化 ID_Table。
 - 获取系统时间。
2. 解析命令行参数，根据参数设置调试等级、服务器 IP 地址和构建字典树所需的文件。
3. 打开字典树文件，如果成功打开，则读取文件内容并构建字典树。
4. 初始化缓存。
5. 创建一个线程来定时清理缓存。
6. 初始化 Socket 库。
7. 进入主循环，接收来自客户端或服务器的数据包，并根据包的类型进行处理：
 - 如果是响应报文 (qr=1)，表示从服务器接收到的数据包，进行以下操作：
 - 检查包中的 URL 是否在字典树中存在，如果存在则替换为字典树中的 IP 地址。
 - 根据 ID 在 ID 表中查找对应的记录。
 - 如果找到对应的记录，则将包的 ID 替换为记录中的 ID，并发送回客户端。
 - 根据需要将 IP 地址屏蔽或添加到缓存中。
 - 如果是查询报文 (qr=0)，表示从客户端接收到的数据包，进行以下操作：
 - 检查包中的 URL 是否在缓存中存在，如果存在则直接从缓存中获取 IP 地址。
 - 如果缓存中不存在，则在字典树中查找 URL 对应的 IP 地址。
 - 如果在字典树中找到 IP 地址，则进行以下操作：
 - 将包中的 IP 地址替换为字典树中的 IP 地址。
 - 将 URL 和 IP 地址添加到缓存中。
 - 构建响应报文，将 IP 地址发送回客户端。
 - 如果在字典树中没有找到 IP 地址，则将查询报文转发给服务器端，并在 ID 表中保存 ID 和客户端地址的对应关系。
8. 打印调试信息和发送的数据包内容。
9. 循环结束后，关闭套接字，释放内存，退出程序。

B. 命令行参数解析:

```
void set_Command_Option(int argc, char *argv[], char *server_IP,
                        char *file_name)
{
    // 解析命令行参数
    for (int i = 1; i < argc; i++) {
        if (strcmp(argv[i], "-d") == 0) {
            debug_level = 1;
        } else if (strcmp(argv[i], "-dd") == 0) {
            debug_level = 2;
        } else if (i == 2) {
            strcpy(server_IP, argv[i]);
        } else if (i == 3) {
            strcpy(file_name, argv[i]);
        }
    }

    // 输出设置结果
    printf("Set debug_level: %d\n", debug_level);
    if (server_IP != NULL) {
        printf("Set DNS server: %s\n", server_IP);
    }
    if (file_name != NULL) {
        printf("Set File Name: %s\n", file_name);
    }
}
```

参数:

- **argc:** 表示命令行参数的数量。
- **argv:** 一个字符串数组, 包含命令行参数的值。
- **server_IP:** 一个字符串指针, 用于存储解析出的服务器 IP 地址。
- **file_name:** 一个字符串指针, 用于存储解析出的文件名。

功能: 根据命令行参数设置调试等级、服务器 IP 地址和文件名。

流程: 函数通过遍历命令行参数数组来解析参数。如果参数为 `-d`, 则将调试等级设置为 1; 如果参数为 `-dd`, 则将调试等级设置为 2。如果参数的位置是第二个 (`i == 2`), 则将其解析为服务器 IP 地址; 如果参数的位置是第三个 (`i == 3`), 则将其解析为文件名。

C. 套接字初始化:

```
//创建套接字
int create_socket()
{
    int socket_fd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (socket_fd < 0) {
        printf("Failed to create socket.");
    }
    return socket_fd;
}
```

```
//绑定套接字到指定地址与端口号
int bind_socket(int socket_fd, const char *ip, int port,
                struct sockaddr_in *address)
{
    memset(address, 0, sizeof(struct sockaddr_in));
    address->sin_family = AF_INET;
    address->sin_addr.s_addr = htonl(INADDR_ANY); // 绑定到任意可用地址
    address->sin_port = htons(port);

    // 设置套接字选项, 禁止地址复用
    int reuse = 0;
    if (setsockopt(socket_fd, SOL_SOCKET, SO_REUSEADDR,
                  (const char *)&reuse, sizeof(reuse)) < 0) {
        printf("Failed to set socket options.\n");
        return -1;
    }

    // 绑定套接字到指定地址与端口
    if (bind(socket_fd, (struct sockaddr *)address,
            sizeof(struct sockaddr_in)) < 0) {
        printf("Failed to bind socket port.\n");
        return -1;
    }

    return 0;
}
```

```
void init_Socket(char *server_ip)
{
    // 创建套接字
    my_socket = create_socket();

    // 绑定套接字到本地地址与端口
    int bind_result = bind_socket(my_socket, NULL, 53, &client_address);

    // 设置服务器地址信息
    server_address.sin_family = AF_INET;
    inet_pton(AF_INET, server_ip, &(server_address.sin_addr));
    server_address.sin_port = htons(53);

    if (my_socket ≥ 0 && bind_result == 0) {
        printf("Socket build and bind successfully!\n");
    }
}
```

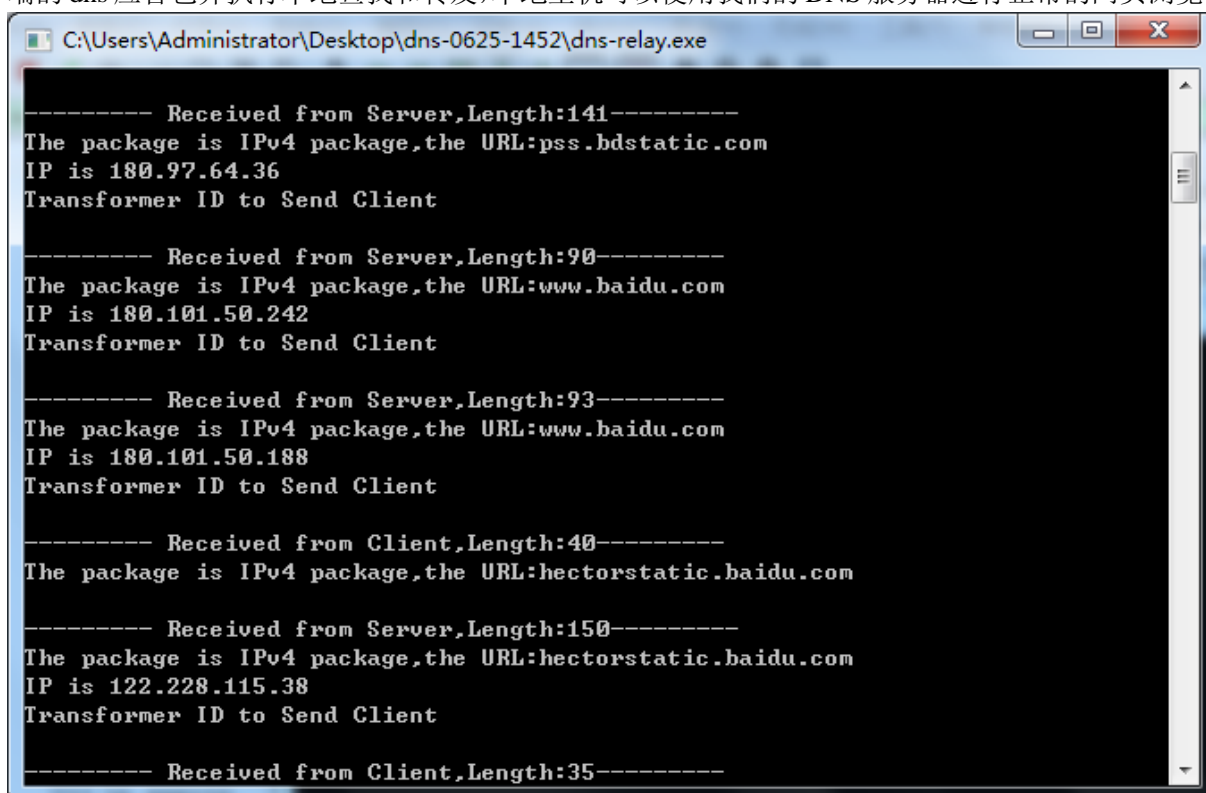

流程:

- a) 首先, `create_socket` 函数创建一个套接字, 使用 `socket` 函数指定地址族为 IPv4 (AF_INET), 类型为数据报套接字 (SOCK_DGRAM), 协议为 UDP (IPPROTO_UDP)。如果创建套接字失败, 将输出一条错误消息, 并返回一个小于 0 的值。
- b) 接下来, `bind_socket` 函数绑定套接字到指定的地址和端口。它接受套接字文件描述符 `socket_fd`、要绑定的 IP 地址字符串 `ip`、端口号 `port`, 以及一个指向 `struct sockaddr_in` 类型的地址结构体 `address` 的指针。
- c) 在函数中, 首先将 `address` 结构体清零, 并设置地址族为 IPv4 (AF_INET), IP 地址为任意可用地址 (INADDR_ANY), 端口号为网络字节序的 `port`。
- d) 接下来, 通过 `setsockopt` 函数设置套接字选项, 禁止地址复用。这样可以确保在套接字关闭后, 重新启动时可以立即绑定到相同的地址和端口, 避免出现 "地址已被使用" 的错误。
- e) 最后, 使用 `bind` 函数将套接字绑定到指定的地址和端口。如果绑定失败, 将输出一条错误消息, 并返回一个小于 0 的值。

五：测试结果分析

A. 基础功能测试

经测试, 我们编写的 dns 服务器能够正确运行, 能够正确接收来自 Client 端的 dns 请求包和 Server 端的 dns 应答包并执行本地查找和转发, 本地主机可以使用我们的 DNS 服务器进行正常的网页浏览:



```
C:\Users\Administrator\Desktop\dns-0625-1452\dns-relay.exe

----- Received from Server,Length:141-----
The package is IPv4 package,the URL:pss.bdstatic.com
IP is 180.97.64.36
Transformer ID to Send Client

----- Received from Server,Length:90-----
The package is IPv4 package,the URL:www.baidu.com
IP is 180.101.50.242
Transformer ID to Send Client

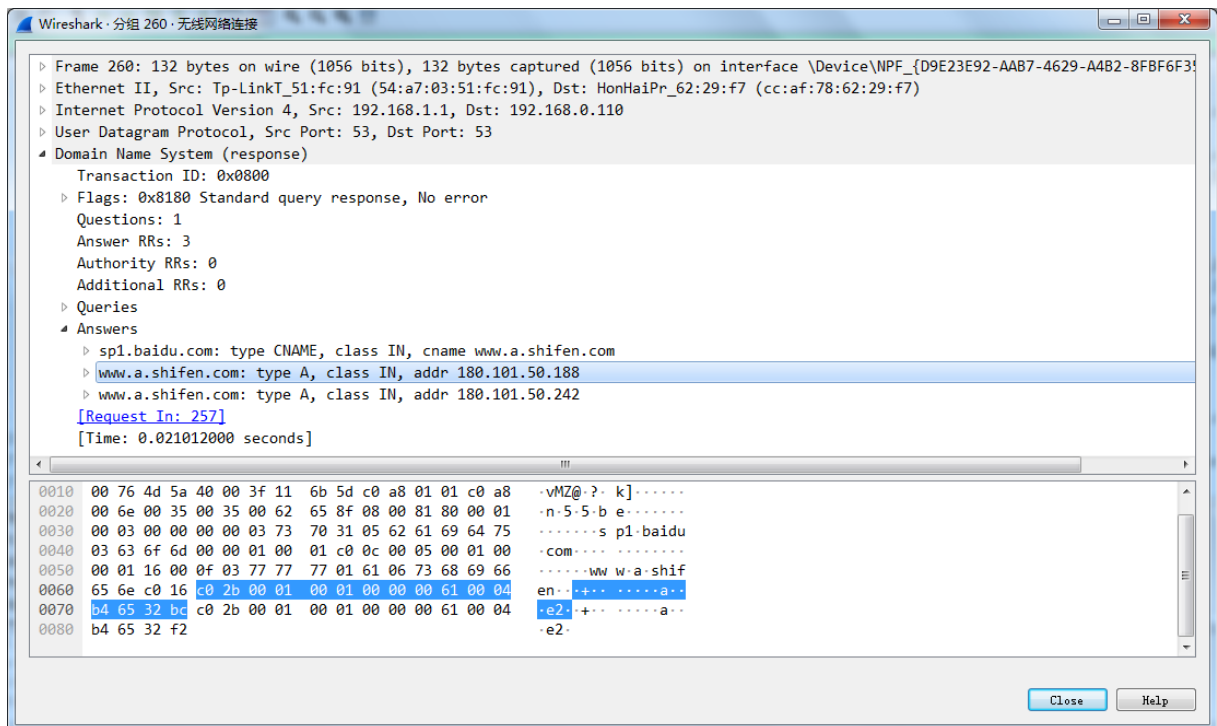
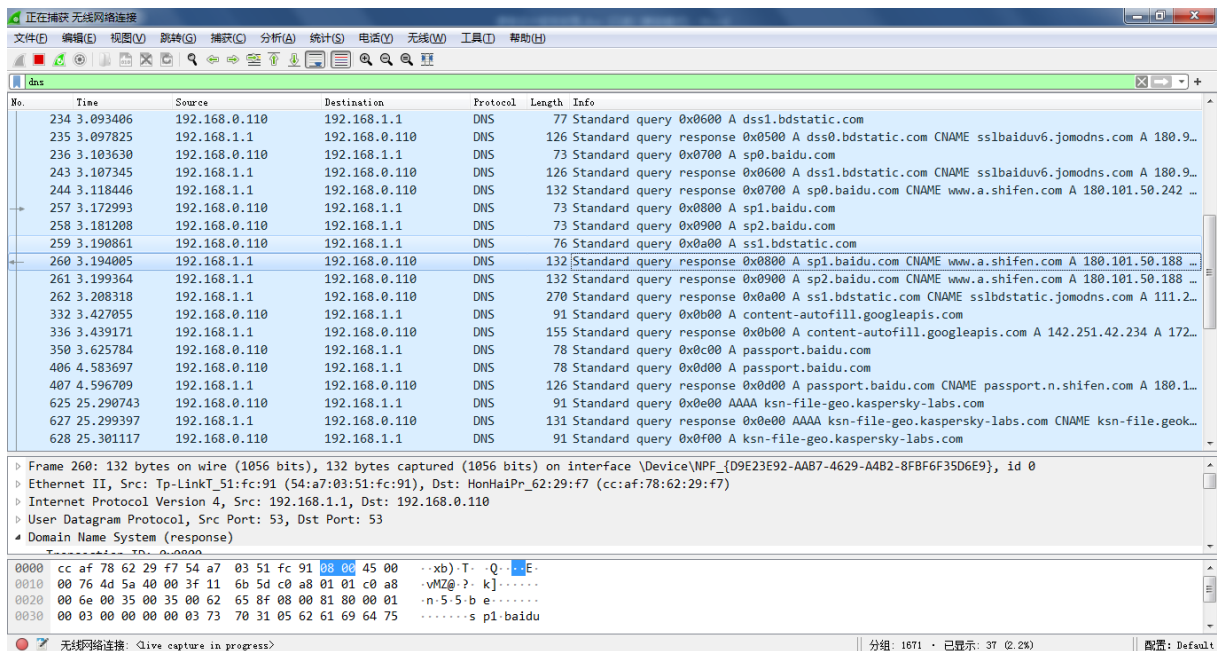
----- Received from Server,Length:93-----
The package is IPv4 package,the URL:www.baidu.com
IP is 180.101.50.188
Transformer ID to Send Client

----- Received from Client,Length:40-----
The package is IPv4 package,the URL:hectorstatic.baidu.com

----- Received from Server,Length:150-----
The package is IPv4 package,the URL:hectorstatic.baidu.com
IP is 122.228.115.38
Transformer ID to Send Client

----- Received from Client,Length:35-----
```

同时通过 Wireshark 抓包我们也可以看到 DNS 中继服务器能够正确接收来自 Client 端的 dns 请求包和 Server 端的 dns 应答包:



B. 三类报文测试


1. 查询常规域名

```
C:\Users\Administrator>nslookup www.baidu.com
```

```
非权威应答:
名称:      www.a.shifen.com
Addresses: 180.101.50.242
           180.101.50.188
Aliases:   www.baidu.com
```

2. 查询 IP 地址为 0.0.0.0 的域名(屏蔽机制)

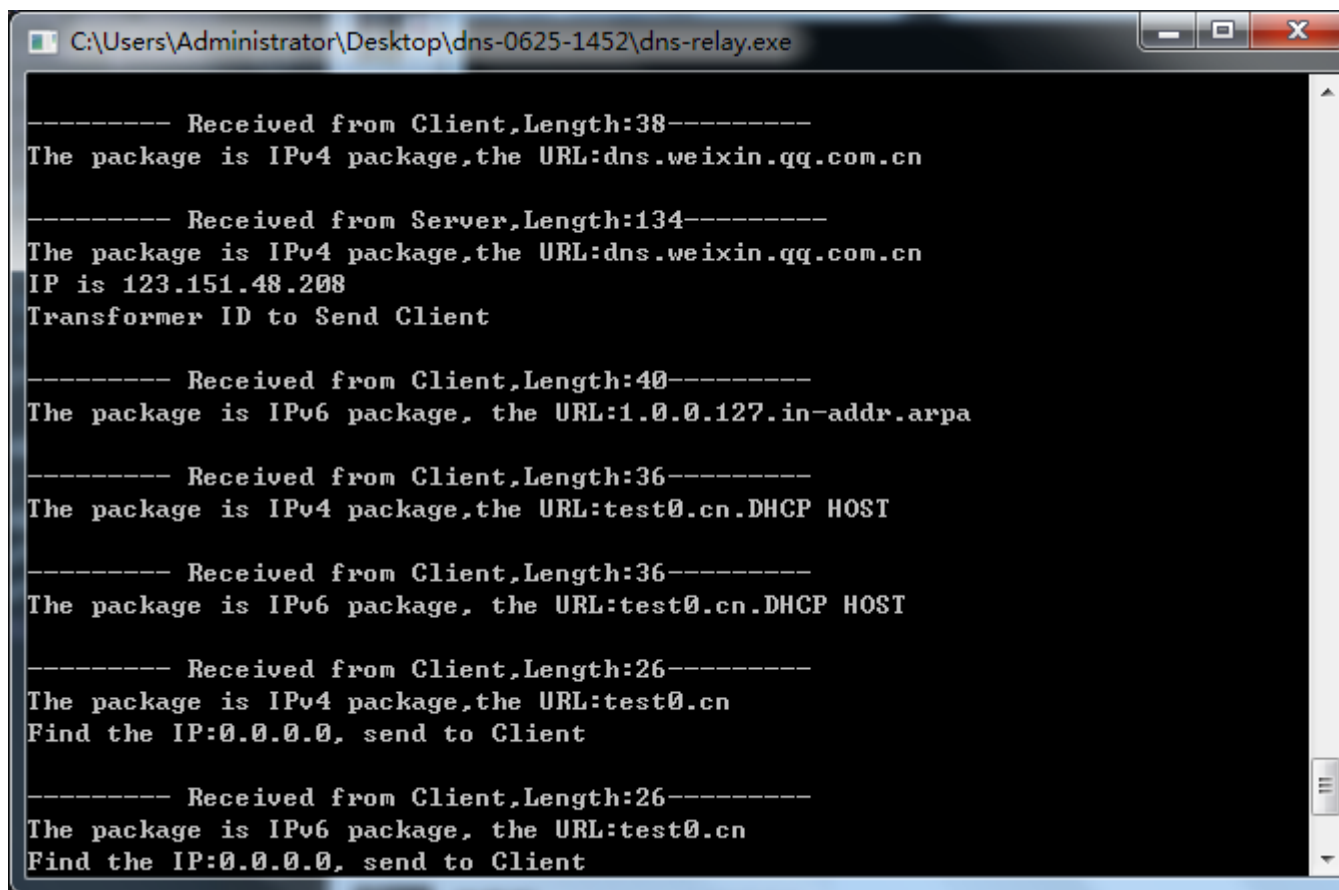
这里使用老师提供的 dnsrelay.txt 文件中的 test0.cn 进行测试

 dnsrelay.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
0.0.0.0 test0.cn
11.111.11.111 test1.cn
22.22.222.222 test2.cn
0.0.0.0 008.cn
0.0.0.0 2qq.cn
0.0.0.0 555.265.com
0.0.0.0 abc.265.com
```

查看下图的最后几条信息可以得知主机请求 test0.cn 的 IP 地址



```
C:\Users\Administrator\Desktop\dns-0625-1452\dns-relay.exe

----- Received from Client,Length:38-----
The package is IPv4 package,the URL:dns.weixin.qq.com.cn

----- Received from Server,Length:134-----
The package is IPv4 package,the URL:dns.weixin.qq.com.cn
IP is 123.151.48.208
Transformer ID to Send Client

----- Received from Client,Length:40-----
The package is IPv6 package, the URL:1.0.0.127.in-addr.arpa

----- Received from Client,Length:36-----
The package is IPv4 package,the URL:test0.cn.DHCP HOST

----- Received from Client,Length:36-----
The package is IPv6 package, the URL:test0.cn.DHCP HOST

----- Received from Client,Length:26-----
The package is IPv4 package,the URL:test0.cn
Find the IP:0.0.0.0, send to Client

----- Received from Client,Length:26-----
The package is IPv6 package, the URL:test0.cn
Find the IP:0.0.0.0, send to Client
```

经过 nslookup 可以得知 test0.cn 得不到 IP 地址，说明 IP 屏蔽效果成功

```
C:\Users\Administrator>nslookup test0.cn
```

```
*** 没有 test0.cn 可以使用的 internal type for both IPv4 and IPv6 Addresses (A+AAAA)记录
```

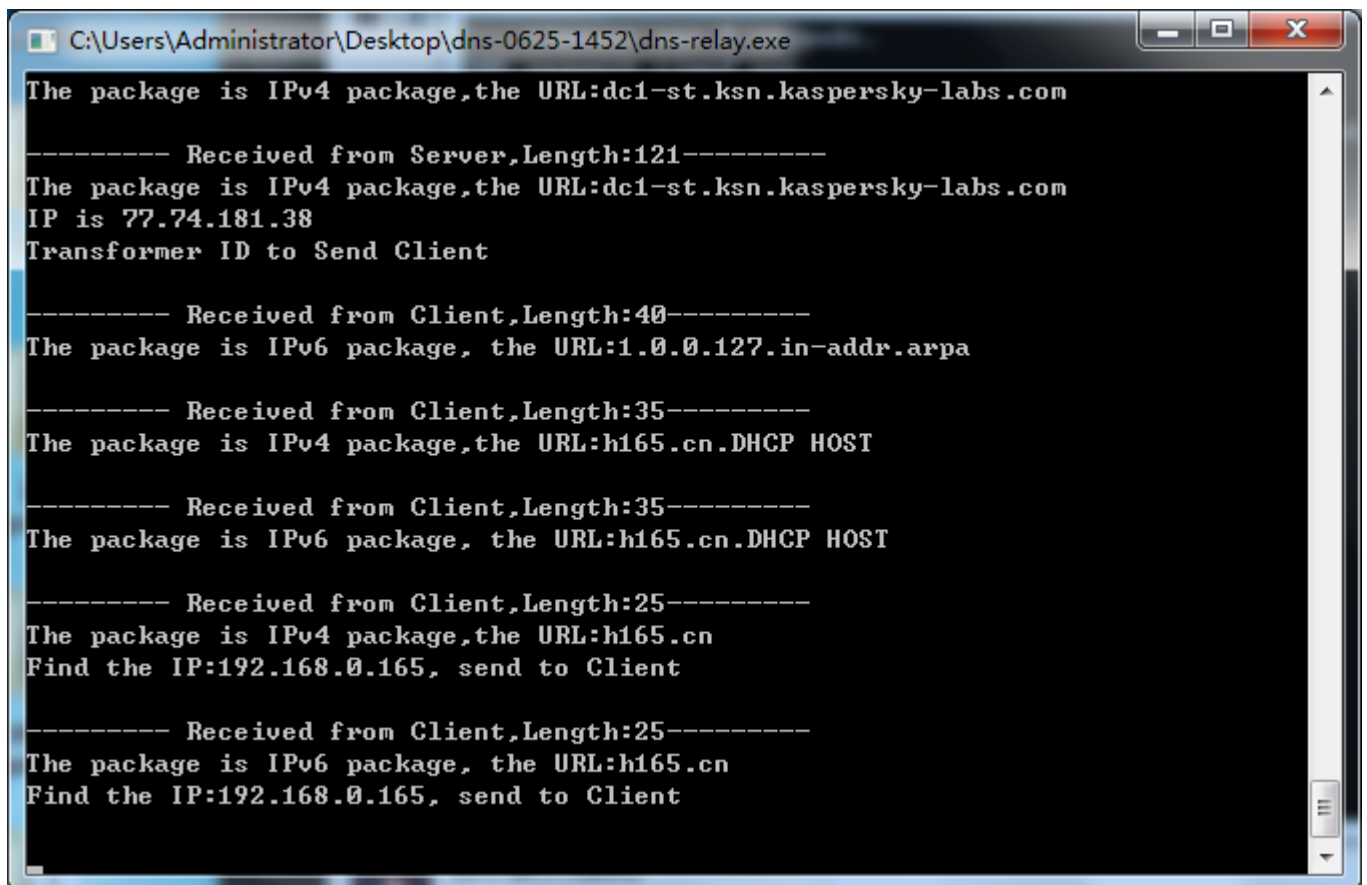
3. 查询 IP 地址为 192.168.0.165 的域名

这里依旧选择使用老师提供的 dnsrelay.txt 文件中的信息—h165.cn 进行测试。



```
dnsrelay.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
192.168.0.162 h162.cn
192.168.0.163 h163.cn
192.168.0.164 h164.cn
192.168.0.165 h165.cn
192.168.0.166 h166.cn
192.168.0.167 h167.cn
```

查看下图的中间部分信息可得知主机请求 h165.cn 的 IP 地址



```
C:\Users\Administrator\Desktop\dns-0625-1452\dns-relay.exe
The package is IPv4 package,the URL:dc1-st.ksn.kaspersky-labs.com
----- Received from Server,Length:121-----
The package is IPv4 package,the URL:dc1-st.ksn.kaspersky-labs.com
IP is 77.74.181.38
Transformer ID to Send Client
----- Received from Client,Length:40-----
The package is IPv6 package, the URL:1.0.0.127.in-addr.arpa
----- Received from Client,Length:35-----
The package is IPv4 package,the URL:h165.cn.DHCP HOST
----- Received from Client,Length:35-----
The package is IPv6 package, the URL:h165.cn.DHCP HOST
----- Received from Client,Length:25-----
The package is IPv4 package,the URL:h165.cn
Find the IP:192.168.0.165, send to Client
----- Received from Client,Length:25-----
The package is IPv6 package, the URL:h165.cn
Find the IP:192.168.0.165, send to Client
```

经过 nslookup 可以得知 test0.cn 的 IP 地址，结合文档中的实际地址可以得知说明字典接口成功实现。



```
C:\Users\Administrator>nslookup h165.cn
```

```
非权威应答:
名称:      h165.cn
Addresses: 192.168.0.165
           192.168.0.165
```

六：用户使用指南

我们小组完成了 dns 程序在 Windos 与 Linux 的兼容,以下主要介绍在虚拟机的 Ubuntu 中运行该程序, Windos 倘若遇到编译器给出报错也可采用同样的方法进行解决。

A. 安装依赖

1. 首先确保系统的软件包列表是最新的

```
harmoon@ubuntu:~/Desktop$ sudo apt update
[sudo] password for harmoon:
Err:1 http://us.archive.ubuntu.com/ubuntu focal InRelease
Temporary failure resolving 'us.archive.ubuntu.com'
Err:2 http://security.ubuntu.com/ubuntu focal-security InRelease
Temporary failure resolving 'security.ubuntu.com'
Err:3 http://us.archive.ubuntu.com/ubuntu focal-updates InRelease
Temporary failure resolving 'us.archive.ubuntu.com'
Err:4 http://us.archive.ubuntu.com/ubuntu focal-backports InRelease
Temporary failure resolving 'us.archive.ubuntu.com'
Reading package lists... Done
Building dependency tree
Reading state information... Done
116 packages can be upgraded. Run 'apt list --upgradable' to see them.
W: Failed to fetch http://us.archive.ubuntu.com/ubuntu/dists/focal/InRelease Temporary failure resolving 'us.archive.ubuntu.com'
W: Failed to fetch http://us.archive.ubuntu.com/ubuntu/dists/focal-updates/InRelease Temporary failure resolving 'us.archive.ubuntu.com'
W: Failed to fetch http://us.archive.ubuntu.com/ubuntu/dists/focal-backports/InRelease Temporary failure resolving 'us.archive.ubuntu.com'
W: Failed to fetch http://security.ubuntu.com/ubuntu/dists/focal-security/InRelease Temporary failure resolving 'security.ubuntu.com'
W: Some index files failed to download. They have been ignored, or old ones used instead.
```

2. 安装 GCC 编译器

```
harmoon@ubuntu:~/Desktop$ sudo apt install build-essential
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  dpkg-dev fakeroot libalgorithm-diff-perl libalgorithm-diff-xs-perl
  libalgorithm-merge-perl libfakeroot
Suggested packages:
  debian-keyring
The following NEW packages will be installed:
  build-essential dpkg-dev fakeroot libalgorithm-diff-perl
  libalgorithm-diff-xs-perl libalgorithm-merge-perl libfakeroot
0 upgraded, 7 newly installed, 0 to remove and 116 not upgraded.
Need to get 842 kB of archives.
After this operation, 2,738 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
```

3. 查看 GCC 编译器是否安装成功

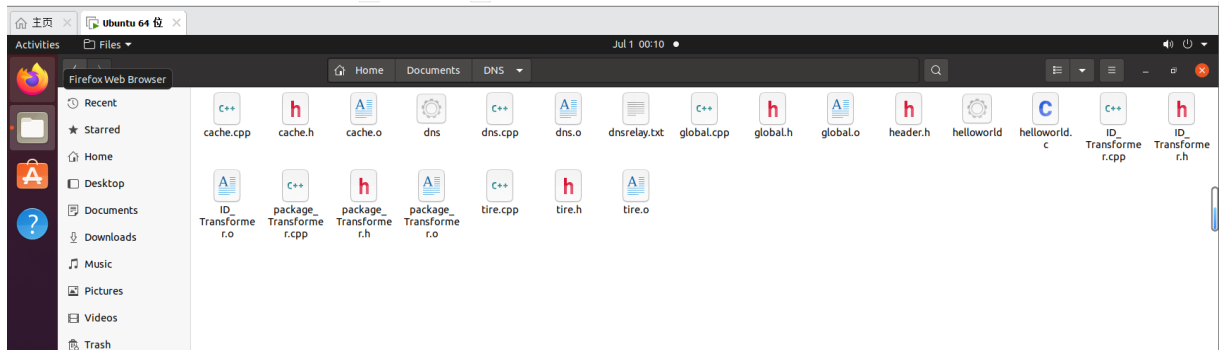

```

harmoon@ubuntu:~/Desktop$ gcc --version
gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

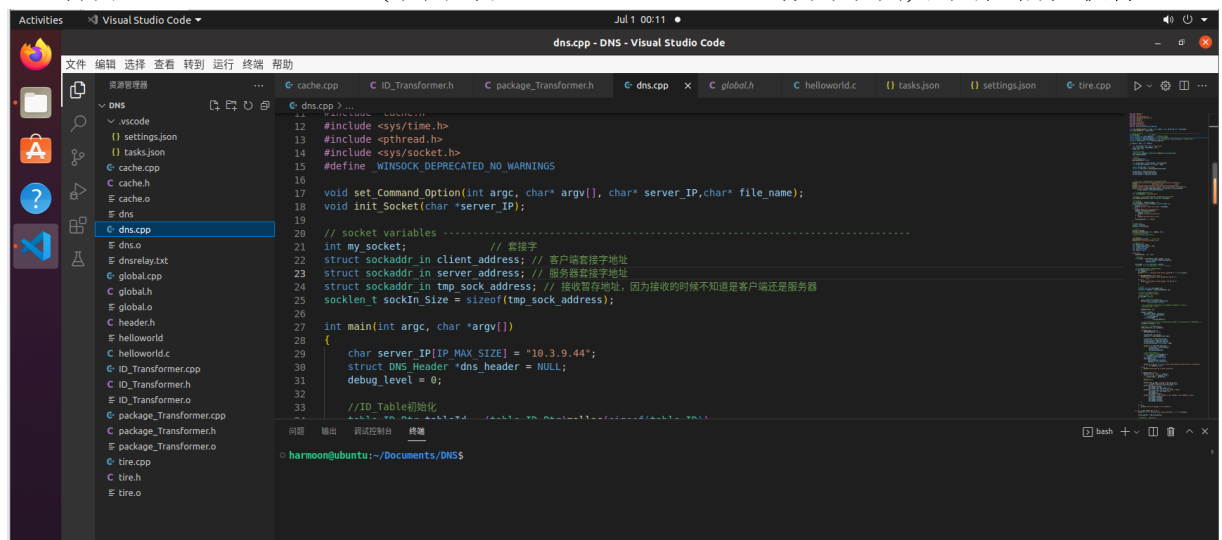
```

B. 编译、执行

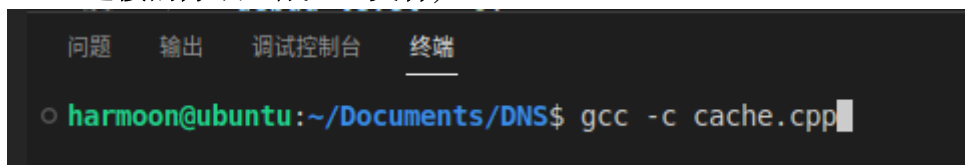
1. 将所有源文件以及 `dnsrelay.txt` 转移到 Ubuntu 系统中，如下图



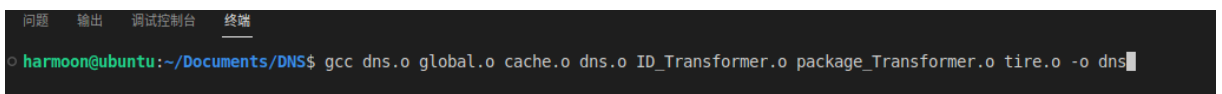
2. 打开 VisualStudioCode(下面均以 VisualStudioCode 作为示例)或其他编程软件



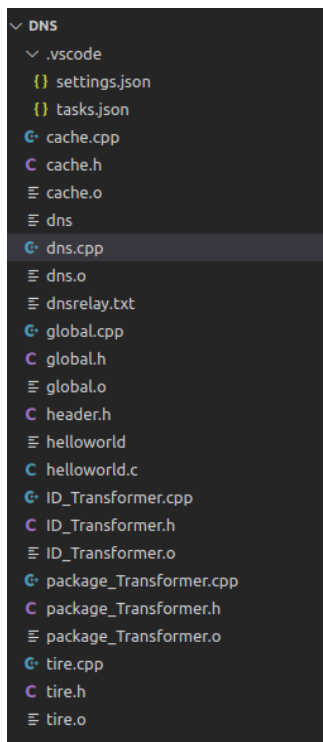
3. 在 VisualStudioCode 中新建终端并输入下图的指令(由于我的虚拟机中的 VSCode 直接进行编译会给出我的 setting.json 有问题, 所以采用手动编译与连接的办法生成 exe 文件)



4. 将生成的.o 文件进行连接



5. 最终得到可执行文件 `dns`



6. 在终端中输入./dns 即可运行程序

```
harmoon@ubuntu:~/Documents/DNS$ ./dns

*****
*                               *
*               DNS Relay Server               *
*                               *
*****
command format: dnsrelay [-d|-dd] [dns-server-ipaddr] [filename]
Start Time:00:22:16.690
Set debug_level: 0
Set DNS server: 10.3.9.44
Set File Name: dnsrelay.txt
Open file is success!
Build tireTree Successfully!
```

七：实验总结

A. 实验中遇到的问题

在程序调试过程中，我们遇到了三个主要问题：

- 1) package_transformer 中的 IPinDNS 函数，一开始我们采用的方法是无论如何我们都读取 Resource Record 字段中的最后一条字段中的 IP 地址。但后来通过 Wireshark 抓包分析后我们发现在 DNS 包的 Resource Record 字段可能存在特殊情况：RR 字段不包含 IP 地址而是只包含一个昵称（CNAME），而我们的初版程序无法处理这种情况。于是我们后来改为先读取 RR 字段的 TYPE，根据 TYPE 的不同，我们采取不同的处理方法：如果 TYPE 为 1，则读取一个 IPv4 地址；若为 28 则读取一个 IPv6 地址；若为其他，则不读取地址。
- 2) 在程序的调试过程中，cache 部分遇到了很多指针相关的问题，比如：在进行链

表头删的过程中，如果删除的是链表中最后一个元素（`head->next==NULL`），则需要更新尾指针指向表头；在将链表中的某元素移动至链表尾时，如果它已经位于尾部，则不需要移动，否则将造成节点的丢失；在进行 cache 刷新时，要先检查 cache 是否为空，否则在程序运行时会出现空指针的问题。

- 3) 在调试等级为-dd 时打印调试信息时，我们遇到了两个问题：在以十六进制打印包内容时，数据部分前出现多个 f，后检查发现是打印时使用 signed 的数据类型所致，后来改为 unsigned 则解决了问题；在输出部分包数据时，我们还遇到了数据类型大小端表示的问题，x86 下数据以小端法表示，而数据报中字节序与 x86 中相反，导致在输出调试信息时存在明显错误。后来我们使用字节转换函数 `ntohl` 和 `htonl` 等解决了问题。

B. 实验耗时

此次课程设计总共耗费我们接近一周的时间去完成。基本上从端午节当天就开始进行准备工作，搜集各种资料，去理解并分析整个 DNS 程序的整个工作流程。在后续的几天基本从早到晚一直在进行代码的编写、程序测试等。

6.22: 详细研究了 DNS 协议的内容，仔细阅读了 RFC 1034 和 1035 文档的关键部分，并参考了一些博客，设计了报文的基本格式。在这个过程中，我们理解并理清了整个程序的大致工作流程，对于构建整个程序有了初步思路。

6.23: 完成了基础的报文转发功能，能够对程序进行初步测试。

6.24: 完成了字典接口与命令行参数解析功能。

6.25: 完成基于 LRU 算法的高速缓存并将其融入到整个框架中。

6.26: 完善了调试信息的输出。针对部分代码进行了优化，对各种样例进行了测试。完成了 Windows 与 Linux 版本的兼容。

6.29-7.1: 撰写实验报告。

C. 收获总结

在本次课程设计中我们主要有如下几个收获：

1. **DNS 解析：**通过实验，我们深入了解了 DNS 解析的过程。我们学会了解析 DNS 报文中的字段，如域名和 IP 地址的对应关系，以及如何进行缓存和刷新操作。这使我们能够实现 DNS 解析功能，并将域名转换为相应的 IP 地址。
2. **网络编程：**实验中涉及了网络编程的基本概念和技巧。我们学习了使用套接字进行网络通信的方法。通过创建套接字、绑定端口、发送和接收数据等操作，我们能够与远程服务器进行通信，实现了 DNS 查询和响应的功能。
3. **套接字函数：**我们掌握了一些重要的套接字函数，如 `socket()`、`bind()`、`sendto()`、`recvfrom()` 等。这些函数使我们能够建立和管理网络连接，并在网络中发送和接收数据。通过正确使用这些函数，能够进行可靠的网络通信，实现了与 DNS 服务器的交互。
4. **错误处理和调试：**在网络编程中，我们学会了分析错误信息和调试代码，以解决网络连接问题、数据传输错误等。
5. **跨平台编程：**通过将代码从 Windows 环境迁移到 Linux 环境，我们学习

了如何处理平台相关的差异和调整代码以适应不同的操作系统。我们修改了一些与操作系统相关的函数和数据类型，使代码能够在 Linux 环境下正确编译和运行。

6. **异步操作和多线程编程：**我们掌握了在程序中创建异步线程并进行异步操作的技巧。学习了如何在主线程和异步线程之间进行数据交互和同步，以实现并行处理和提高程序的效率。

D. 工作总结

在本次课程设计中，从课设初期我们就讨论了程序模块的划分，并明确了各自的任务分工。

在开始这个课程设计之前，小组成员对套接字的应用、报文的结构和含义等方面的了解只停留在表面。因此，在编写程序时，我们花费了很多时间来深入理解和梳理。在梳理清除之后我们研究了跨平台的套接字编程以及高并发的实现。在花费大量的时间后也确实完成了程序的基本功能以及高并发的处理。通过在编写代码时应用理论知识，我们更加深入地理解了 DNS 服务器的工作机制。

总的来说，小组成员通过共同合作编写了 DNS 中继器，进一步明确了服务机制和过程的认识，并锻炼了我们的团队合作的意识和能力，也使得我们能够把在课上学习的计算机网络相关知识运用到现实中去，增强了我们将理论知识转化为实践技巧的能力。