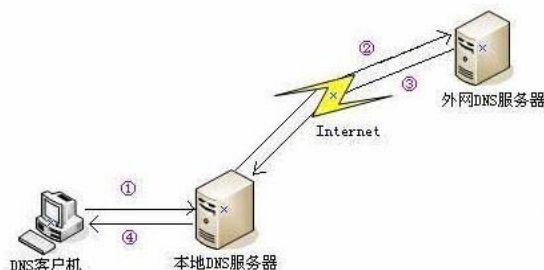


目录

1 设计总体概述.....	3
1.1 设计目标.....	3
1.2 整体分析及架构	3
1.3 编写环境	5
2 模块划分.....	6
3 程序算法	13
4 测试用例及运行结果.....	14
5 实验总结	16
4.1 开发问题总结	16
4.2 优点与不足	16
4.3 实验心得	19

1 设计总体概述

1.1 设计目标



设计一个 DNS 服务器程序，读入本地的“域名（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，存在三种检索结果：

- 1、检索结果为特殊 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）；
- 2、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）；
- 3、若本地列表中未检到该域名，则向外部 DNS 服务器发出查询请求，并将域名解析结果返回给客户端（中继功能）。

1.2 整体分析及架构

1.2.1 功能分析

本次课程设计所需的 DNS 中继服务器需要满足的功能有以下三点：

1. 正常 DNS 服务功能：

将计算机的 DNS 服务器地址指向本机，当用户输入域名时，DNS 服务器需要从本地存储的域名-IP 映射表以及高速缓存（Cache）中进行查询，将查询到的 IP 地址返回给用户进行正常的网址访问。

2. 不良网站拦截功能：

当用户想要访问某个网站输入其域名时，DNS 中继服务器需要获取其查询信息，将用户想要访问的域名与中继服务器中存储的域名-IP 对应表进行对比。将查询表中不良域名的 IP 设置为“0.0.0.0”，如果查到 IP 为全 0，向用户返回特殊的数据包，实现不良网站屏蔽功能。

3. DNS 中继功能

当 DNS 服务器本地存储的域名-IP 对照表中没有存储用户想要访问的域名，向外部 DNS 服务器发出查询请求，并将查询到的结果返回给客户端，同时将其加入到服务器本地的高速缓存（Cache）中。

1.2.2 需求分析

1. 多客户端并发处理能力

作为 DNS 服务器，可能会同时收到位于不同计算机上多个客户端的查询请求。即：允许第一个查询尚未得到答案前就启动处理另外一个客户端查询请求。所以，需要发挥 DNS 协议中 ID 字段的作用，设计 ID 转换表，对不同用户请求进行区分，将查询结果返回到正确的客户。

我们使用同一个 SOCKET 来完成和各个客户端和外界 DNS 服务器的通信，通过一个线程实现，对于不同客户端的消息，我们使用 HASH 对原 ID 和 IP 和转换后的 ID 进行存储，来实现程序的并发，并完成 SOCKET 的保护

2. 超时处理

由于 UDP 的不可靠性，考虑求助外部 DNS 服务器（中继）却不能得到应答或者收到迟到应答的情形。由于发出查询请求到收到应答报文之间存在一定时长，所以需要设定合适的阈值来判断是否超时。当某一用户请求因未得到响应而超时后，系统将不再保

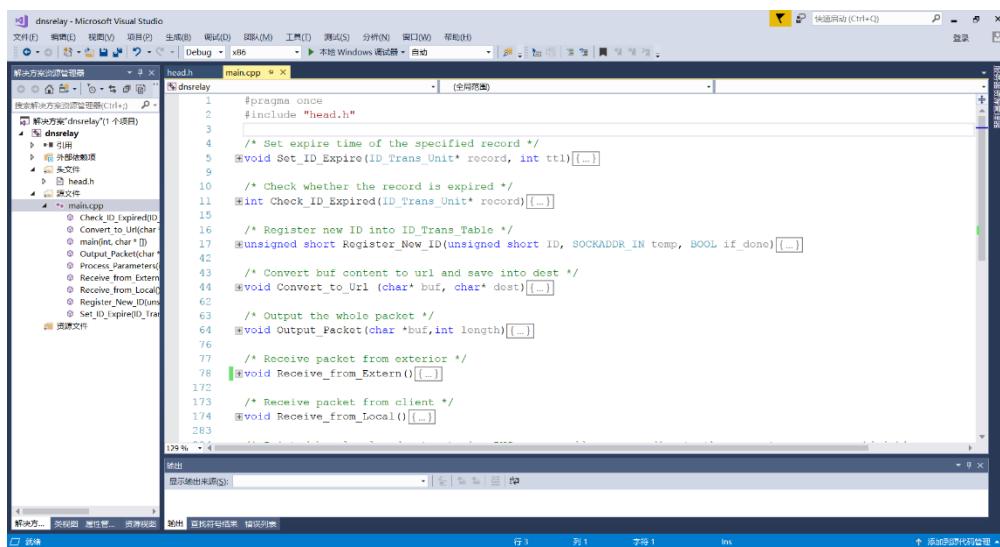
障其在 ID 映射表中的内存资源，当有新的用户请求需要注册操作时，服务器有可能会将其覆盖，避免了服务器资源被长期占用而造成浪费。

1.3 编写环境

在 DNS 中继服务器的设计与开发上，我们采用了团队所有成员比较熟悉的 C 语言中的 Socket 等标准库进行 DNS 服务器的编写工作并在搭载 Windows10 的计算机上进行测试工作。

本次实验的编写环境可分为硬件和软件环境，它们分别如下所示：

- ① 硬件环境：搭载了微软 Windows10 操作系统的笔记本电脑；
- ② 软件环境：Visual Studio 2017 集成开发环境。



Visual Studio 2017 集成开发环境

2 模块划分

2.0 概述：

本程序有两个线程一个 SOCKET，对于客户端与外界 DNS 的通信，我们使用一个 SOCKET 进行通信，然后进行条件判断来选择操作。

两个进程则一个负责通信，一个负责刷新。

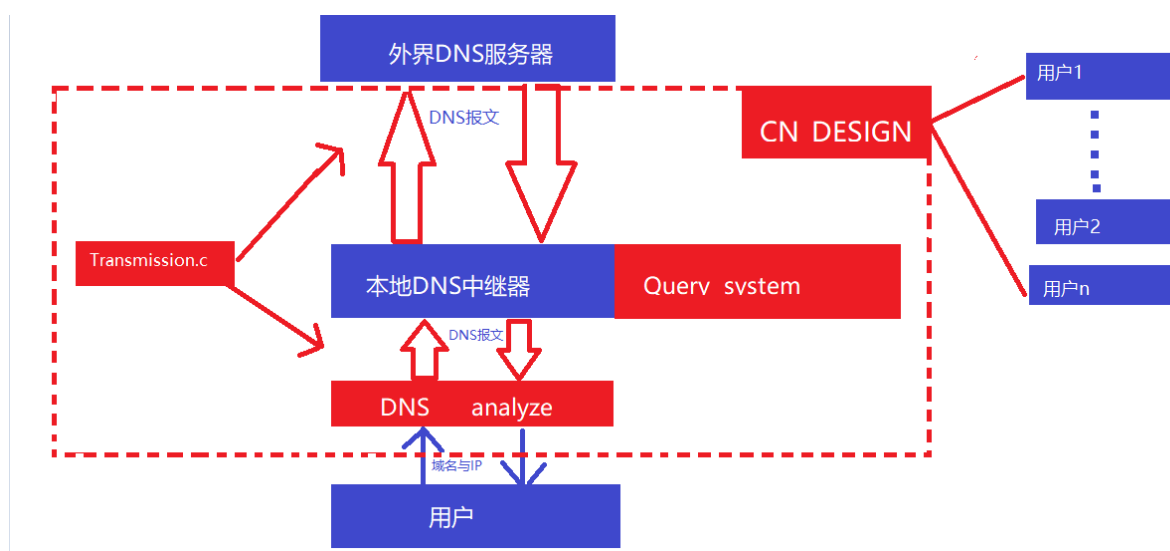
本程序有四个模块：

一个缓存资源维护的模块，主要将外界 DNS 资源添加加入缓存，并对查询包把内存资源加入。

一个 DNS 报文解析模块，将 DNS 报文与 char* 进行相互转换，方便通信，并且在转换时，要改变 DI。

一个传输模块，主要封装通信函数来实现通信。

一个主程序模块：负责命令输入的规定调试等级，还有两个线程的创建。



2.1全局数据结构：

```
struct FLAG {
    unsigned char qr;
```

```
    unsigned char opcode;
    unsigned char aa;
    unsigned char tc;
    unsigned char rd;
    unsigned char ra;
    unsigned char z;
    unsigned char rcode;
};

struct QUE {
    unsigned char* domain;
    unsigned short type;
    unsigned short class;
};

struct RR {
    struct QUE que;
    unsigned int ttl;
    unsigned short len;
    unsigned char* resource;
    unsigned char* resource_string; /*此区域仅在class=1, type=5时有效, 代表点分形式的cname回答,
此时resource无效*/
};

struct DNS {
    unsigned short id;
    struct FLAG flag;
    unsigned short question_num;
    unsigned short answer_num;
    unsigned short authority_num;
    unsigned short additional_num;
    struct QUE* que;
    struct RR* ans;
    struct RR* aut;
    struct RR* add;
};
```

这都是与 DNS 报文相关的, 用以解析 DNS 报文和封装 DNS 报文的数据结构

2.2 模块一：缓存资源模块

模块名称：Querysystem

该模块主要负责维护缓存资源, 一方面是将外界 DNS 的报文中的资源添加到本地缓存资源, 另一方面是, 对于来自客户端的查询包, 如果能在本地缓

存中找到结果 IP 就把这些答案资源添加到 DNS 相应报文中。还有就是刷新内存资源，对过期的资源进行清理。

模块特有数据结构：

```
struct DOMAIN_CNAME_RECORD {
    struct DOMAIN_CNAME_RECORD* next;
    unsigned char* domain;
    unsigned char* cname;
    int ttl;
};

struct CNAME_IP_RECORD {
    struct CNAME_IP_RECORD* next;
    unsigned char* cname;
    unsigned char** ip;
    int* ip_ttl;
    int flag;
    int ip_num;
};
```

这是两个链表节点的数据结构，用来构成记录缓存 IP-CNAME，DOMAIN-CNAME 这些资源的链表。

其中需要说明的是，CAME_IP_RECORD 用以构成 CAME_IP 的储存链表，一个 CNAME 可能对应多个 IP，我们会存下所有的 IP 记录而且对于 DOMAIN-IP 这样的 A 记录，我们会将 DOMAIN-IP 记录进入这个链表，把 DOMAIN 当成 CNAME。

而 DOMAIN_CNAME_RECORD 构成的 DOMAIN_CNAME 存储链表，其中一个 DOMAIN 对应一个 CNAME，但是根据实际情况一个 DOMAIN 可能会有多个 CNAME，所以，其实我们这是一个域名对域名的数据结构，还记录了 CNAME-CNAME 的转化过程。

所以总的来说，如果有一个 CNAME 记录，则会将 DOMAIN-CANME, CNAME-CNAME 多重别名全部计入 DOMAIN-CNAME 链表中，并且将 DOMAIN 和每一个 CNAME 对应的 IP 都计入 CNAME-IP 组中。

模块函数说明：

```
int queryDomainIp(struct DNS* dst, struct DNS src);

/*对于 src 对应 DNS 中 query 部分，在本地进行查询，通过调用先查询 DOMAIN-CNAME, 再查询每个 CNAME 对应的 IP 得到结果，将结果写入 dst 处 DNS 中去，查询失败则返回 0，成功返回 1*/

static struct DOMAIN_CNAME_RECORD* queryCname(unsigned char* src) /*查询 src 对应字符串的 cname，查询成功则返回记录对应数据的地址，如果失败（无记录或过期）则返回 NULL*/

这里需要说明的是，由于一个 DOMAIN 引申出来的 CNAME 有多个，但是这个函数只会返回其中最直接的一个，所以在 queryDomainIp 需要循环树状查询。

static struct CNAME_IP_RECORD* queryIp(unsigned char* src) /*查询 src 对应字符串的 ip，查询成功则返回记录对应数据的地址，如果失败（无记录或过期）则返回 NULL*/

void domainIpRecordFlush(void) 用于缓存资源的刷新，因为 cache 需要替换，所以根据 TTL 来进行刷新。

static void addAnswer(struct DNS* dns, struct RR add) /*将 add 处资源区域数据写入 dns 的 ans 区域中，包括更改资源区域数量以及内存申请*/

static int hashCalculate(unsigned char* src, int hashnum) /*计算 src 处对应字符串在 domain-ip 转换过程中对应哈希值，用于定位记录的位置*/

static void freeIpRecord(struct CNAME_IP_RECORD* record, int mode) /*清空 CNAME-ip 中一条记录，mode 代表模式，如果为 0 代表完全释放，否则不释放 cname 部分*/

static void addCnameIp(struct DNS dns) /*将 dns 报文中所有 cname-ip 的回答添加入转换的记录中*/

static void addDomainCname(struct DNS dns) /*将 dns 报文中所有 domain-cname 的回答添加入转换的记录中*/ （循环添加）
```

对于从外界 DNS 的包：会先调用 addCnameIP 和 addDomainCname 两个函数将资源记录入缓存，其中对于选择缓存的位置，会用到 hashcalculate 函数来进行位置计算。后再将包转发给客户端

对于客户端的请求包：会先进行 queryDomainIp，在函数会调用 hashcalculate，然后再调用 queryip 和 querycame 进行查询，最终返回包含结果的包。

2.3.DNS报文解析模块：

模块名称：DNSanalyze

该模块主要负责将收到的char*区域字符串重新构建为一个DNS报文，也可以将要发送的DNS报文转化为char*

1.特有数据结构：

```
struct COMPRESS_INFO {
    unsigned char* name;
    unsigned short offset;
    short next;
```

};/用以解析域名中的 c0 跳转所用的中间结构

```
static void myMemcpy(unsigned char* dst, unsigned char* src, unsigned int len) {/*实现大小端不同的两块内存区域的数据拷贝，比如 src 字节为 01 02 03，长度为 3，则以 03 02 01 的顺序写入 dst*/
```

```
static unsigned char* domainAnalyze(unsigned char* buff, unsigned char* dns) {/*函数用于解析 buff 地址上的域名，以 00 或 0c 跳转指令结尾，dns 为报文的初始地址，用于处理 0c 跳转后的域名解析*/
```

```
static unsigned int domainInverseAnalyze(unsigned char* dst, unsigned char* src) {/*把 src 处点分域名字符串转换为 dns 报文中的格式，写入 dst，如 www.baidu.com\0 转换为\03www\05baidu\03com，返回转换后有效长度*/
```

```
void freeDns(struct DNS* dnsptr) {
    /*函数实现对 dnsptr 指向的 DNS 数据块进行清除操作，释放占用的内存，并将整个 DNS 数据块清零*/
```

以上四个函数是功能函数是被调用的，最终这个模块被使用的是一下两个封装好的函数，来完成模块简介中的两大功能：

```
unsigned char* dnsToData(struct DNS* dns, unsigned int* len) {
    unsigned int dataToDns(unsigned char* buff, struct DNS* dns) {//函数用于把 buff 指针所指向的原始 dns 报文转换为本程序中可识别的格式，变得容易读取，包括域名部分的解压缩（c0 跳转指令）。返回原始 dns 报文长度
```

2.4 传输模块：

模块名称：Trassmission.c

该模块主要负责与客户端/外界服务器通信，接收和发送报文，用来封装 recv()和 send()函数，还有比较重要的 ID 转换。

需要说明的是，对于通信和 socket，我们采取了单线程和单 SOCKET 来避免 socket 套接字被复用，于是在 recvdns()函数采取了 if-else 的判断，来根据收到包的类型选择操作。

特有数据结构：

```
struct ID_IP_TRANS_INFO {
    struct ID_IP_TRANS_INFO* next; /*相同hash的下一个记录*/
    unsigned short id; /*从客户端来的时候的id*/
    unsigned short trans_id; /*和服务端通信时使用的id*/
    struct sockaddr addr; /*客户端的地址*/
    int ttl; /*过期时间*/
}trans_info[256];
```

用于记录 ID 转换的链表

函数分析：

通信类：发送,接收,初始化

```
int recieveDns(unsigned char* buff)
void sendDns(unsigned char* buff, int len)
void networkConnectionInit(void)
```

ID 转换类：

```
static void addTransInfo(unsigned short id, struct sockaddr addr, unsigned short trans_id) { /*将
id 转换记录添加入缓存，其中 id 为原始 id，addr 为客户端地址，trans_id 为转换后 id*/
static int findTransInfo(unsigned short trans_id, struct sockaddr* addr, unsigned short* id) { /*
根据 id 寻找存储的有效记录，trans_id 为要查找的 id，查找结果中客户端地址写入 addr 指向内存，对应原
始 id 写入 id 指向内存中，如果查找成功则返回 1，失败则返回 0*/
void transInfoFlush(void) /刷新
void transInfoInit(void) {
```

2.5 主函数模块：

该模块主要就是来进行线程操作，在我们的程序中有两个线程，其中一个线程负责通信，另外一个线程则负责内存资源的刷新。

```
DWORD WINAPI recieveProcessDns(void) /通信线程函数
DWORD WINAPI localRecordFlush(void) { /刷新线程函数
void argvProcess(int argc, char* argv[]) { //由命令行设置的调试等级函数
```

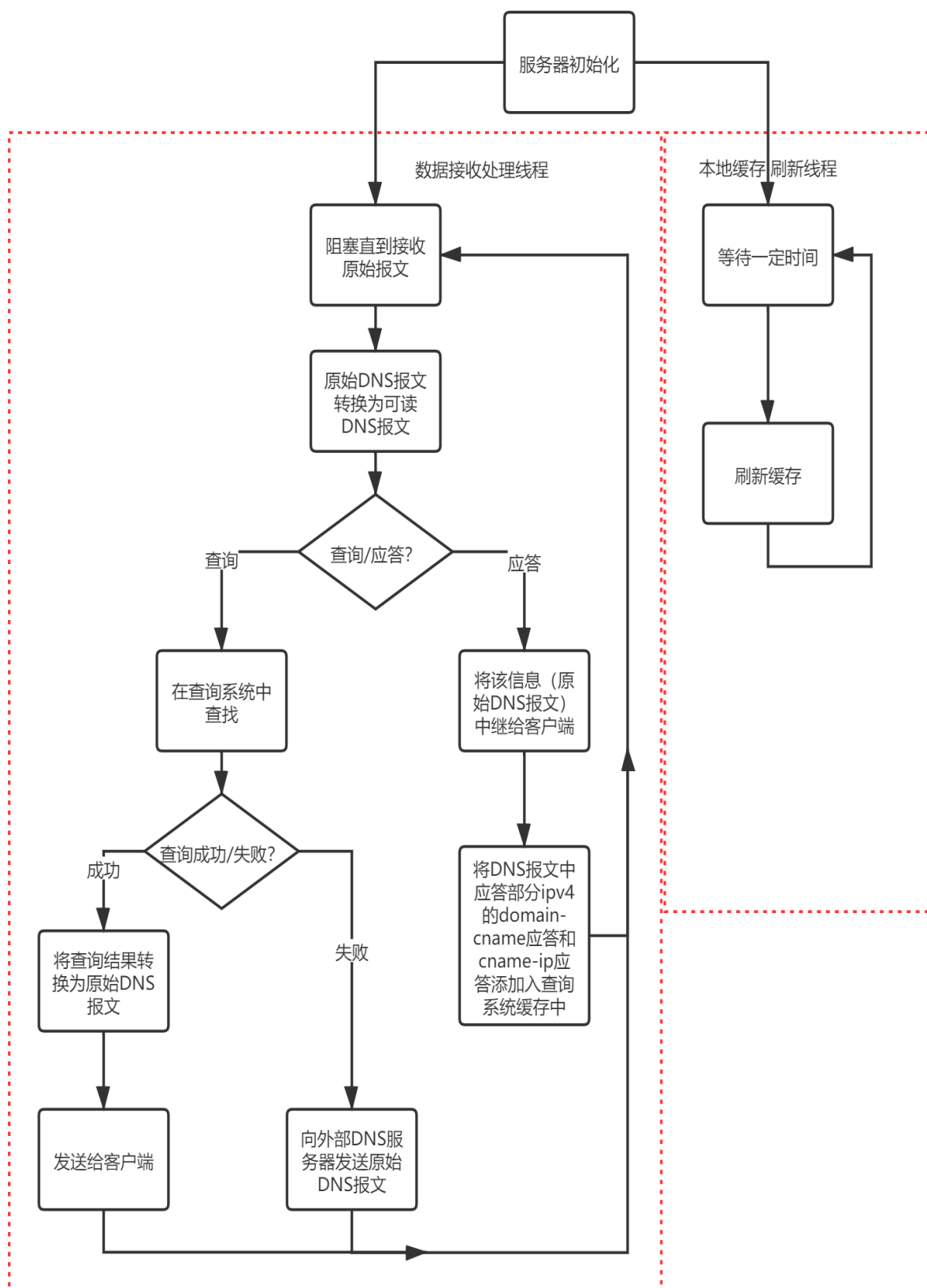
2.6 debug 模块：

负责输出输入信息的规定与解读，还有调试信息输出：

```
void debugDns2(unsigned char* dns, int len) {  
void debugDns(struct DNS dns) {
```

都是输出 DNS 报文，但是输出详细不一样，根据命令行调试等级决定。

3 程序算法流程：




4.测试用例及截图：

其中拦截中继查询功能截图如下：

1、有预设的网站

预设文件如下：

 sets.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V)

www.bing.com 1.1.1.1

www.tencent.com 0.0.0.0

www.bupt.edu.cn 1.2.3.4

测试结果：

```
C:\Users\99314>nslookup
默认服务器: UnKnown
Address: 127.0.0.1

> www.bing.com
服务器: UnKnown
Address: 127.0.0.1

非权威应答:
名称: www.bing.com
Address: 1.1.1.1

> www.tencent.com
服务器: UnKnown
Address: 127.0.0.1

*** UnKnown 找不到 www.tencent.com: Non-existent domain
> www.bupt.edu.cn
服务器: UnKnown
Address: 127.0.0.1

非权威应答:
名称: vn46.bupt.edu.cn
Addresses: 2001:da8:215:4038::161
1.2.3.4
Aliases: www.bupt.edu.cn

>
```

服务器运行情况：

```

=====
DNS-----ID: bec4 ----- Q, A, A, A:1 0 0 0
QUESTION 0:
www.bupt.edu.cn
type:1 class:1
=====

```

```

=====
Time:6.960 Query Ip Succ! responding...
=====

```

```

=====
DNS-----ID: 70b1 ----- Q, A, A, A:1 0 0 0
QUESTION 0:
www.bupt.edu.cn
type:28 class:1
=====

```

```

=====
Time:6.966 Query Ip Failed ,relaying...
=====

```

Wireshark 抓包:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	DNS	93	Standard query 0x000c A www.bupt.edu.cn
2	0.001840	127.0.0.1	127.0.0.1	DNS	109	Standard query response 0x000c A www.bupt.edu.cn A 1.2.3.4
3	0.002247	127.0.0.1	127.0.0.1	DNS	93	Standard query 0x000d AAAA www.bupt.edu.cn
4	0.077353	127.0.0.1	127.0.0.1	DNS	140	Standard query response 0x000d AAAA www.bupt.edu.cn CNAME vn46.bupt.edu.cn AAAA 2001:da8:215:4038::161

2、对没有预设的网站查询：(baidu)

```

> www.baidu.com
服务器: UnKnown
Address: 127.0.0.1

非权威应答:
名称: www.wshifen.com
Addresses: 104.193.88.123
          104.193.88.77
Aliases: www.baidu.com
         www.a.shifen.com

```

Wireshark 抓包

7	56.131582	127.0.0.1	127.0.0.1	DNS	91	Standard query 0x000e A www.baidu.com
8	56.206981	127.0.0.1	127.0.0.1	DNS	176	Standard query response 0x000e A www.baidu.com CNAME www.a.shifen.com CNAME www.wshifen.com A 104.193.88.123 A 104.193.88.77
9	56.207310	127.0.0.1	127.0.0.1	DNS	91	Standard query 0x000f AAAA www.baidu.com
10	56.286710	127.0.0.1	127.0.0.1	DNS	201	Standard query response 0x000f AAAA www.baidu.com CNAME www.a.shifen.com CNAME www.wshifen.com SOA ns1.wshifen.com

5 实验总结

5.1 开发问题总结

经过反复测试，服务器的运行性能及健壮性均十分良好，因此通过结果及分析可以看出，我们对整个 DNS 服务器的设计与实现是非常成功的。但开发过程是充满艰辛与坎坷的，并且团队曾经遇到若干比较棘手的问题。下面本报告对开发过程中所遇到的比较典型的问题进行总结与归纳。

5.1.1 线程与 SOCKET 资源的保护

在一开始编写程序时，我们对于客户端和服务端的通信采用了两个 SOCKET 使用双线程来进行通信。其中对于两个线程共同使用的缓存模块，我们采用了线程锁来进行保护，而且采用了阻塞模式接收两端的消息。我们一开始认为这样的形式非常好，这样可以互不冲突然后各自独立自主的进行通信。但是经过老师的验收后我们发现了问题所在：

由于 `recv()` 我们采取了阻塞模式，那么是不可以将 `recv()` 函数放入线程锁里对 SOCKET 进行保护的，因为这么做会导致线程锁一直被阻塞函数占用，另一个线程无法实现对缓存的读写。

那么这时 SOCKET 会出现问题了，考虑这样的场景，当外界服务器发包要转发给客户端时，那么根据我们的设计，要用与客户端通信的 SOCKET 来进行转发，但这个动作是在与服务器通信的线程中实现的，如果这是与客户端通信的线程也要恰好 `recv`，那么两个线程都要使用客户端通信 SOCKET 这个资源，将会导致冲突。

于是我们改变策略，采用单线程单 SOCKET 进行通信，对受到的报文进行标志位分析，再进行发送，采用 `if-else` 模式，完成了解决方案。

5.1.2 IPV6 包与内存资源的处理

在验收时，老师还提出了我们的另一个问题，就是当查询时，返回 IPV6 包的问题，其实当时也差不多实现了这个功能，只是我们维护了一个缓存表，IPV6 地址和 IPV4 地址格式不一样，所以会导致我们的缓存表出问题，所以当时就会禁止 IPV6。

回去我们立刻将判断条件改了一下，设置为当收到外界 DNS 的 IPV6 报文时，会将该报文转发给客户端但并不记入缓存，便解决了这一问题。

5.1.3 其他的问题

(1) 压缩报文的问题，在处理压缩报文时，会发现有循环跳转等各种问题，在详细了解资源区域名的压缩规律后，最终写出了适用于各种压缩方式的域名解析函数。

(2) 返回错误包问题，当时没有注意到，只是将被屏蔽的网站以 0.0.0.0 转发了回去，但是后来发现并不是这样，经过老师的提醒后，我们将屏蔽的 DNS 报文中的标志位改变，实现了功能。

(3) 资源存储问题，我们对于很多的资源，如 CNAME-IP 表，DOMAIN-CNAME 表和 ID 表，对于这些表的存储方式查询方式和 TTL 维护我们尝试了很多手段，最后解决了安全高效存储的问题。

5.2 优点与不足

5.2.1 系统的优点

1. 服务器功能完善、性能良好

根据 DNS 中继服务器设计要求，我们的服务器实现了要求的所有功能，同时在此基础上，我们还为 ID 转换表的表项设计了超时机制。经过大量测试，该系统满足设计需求，运行情况良好，能够很好的完成以下功能：

- **正常 DNS 服务功能：**将计算机的 DNS 服务器地址指向本机，当用户输入域名时，DNS

服务器需要从本地存储的域名-IP 映射表以及高速缓存（Cache）中进行查询，将查询到的 IP 地址返回给用户进行正常的网址访问；

·**不良网站拦截功能：**当用户想要访问某个网站输入其域名时，DNS 中继服务器需要获取其查询信息，将用户想要访问的域名与中继服务器中存储的域名-IP 对应表进行对比。将查询表中不良域名的 IP 设置为“0.0.0.0”，如果查到 IP 为全 0，向用户返回特殊的数据包，实现不良网站屏蔽功能；

·**DNS 中继功能：**当 DNS 服务器本地存储的域名-IP 对照表中没有存储用户想要访问的域名，向外部 DNS 服务器发出查询请求，并将查询到的结果返回给客户端，同时将其加入到服务器本地的高速缓存（Cache）中；

·**生存时间的设定：**由于 UDP 报文传输的不可靠性，在发出查询到收到应答报文之间存在一定时延，所以需要设定合适的阈值来判断是否超时。将超时的 ID 从转换表中释放，避免无用 ID 占用转换表，造成资源浪费的问题；

·**多用户端并发处理：**我们使用了一个线程一个套接字，然后来完成通信，如上所说，解决了 SOCKET 资源保护问题

2. 扩展功能模块：Cache

当用户查询的域名在服务器中并没有存储时，此时中继服务器需要向外部 DNS 服务器发出请求，并将查询到的结果返回给客户端。在这个过程中，因为涉及到向外请求连接的过程大大降低了 DNS 服务器的处理速度。

于是我们想到“计算机组成原理”课程中学到高速缓存的原理。我们向外部 DNS 服务器查询 IP 的过程就类似 CPU 从主存中取数据的过程，如果我们将查询过的 IP 地址放进 Cache 中，那么之后的访问便可以大大加快访问速度。因此，我们设计当 DNS 中继服务器接收到外部服务器返回的 DNS 报文时，将这个报文的关键信息存储下来，如果之后再有相同的访问，便可以立即读取 IP 地址并组装数据包后将其返回给客户端。

Cache 存储结构的选择：在访问记录的缓存上，我们使用了链表实现的哈希，这种方式查询速度很快，可以保证加快查询速率。

5.2.2 存在的不足

1. 程序用户交互性较弱

在 DNS 中继服务器的设计上，我们没有采用图形化界面，而仅使用了控制台命令行的形式与管理员用户进行交互。整体而言，这样的界面对于软件的使用者来说体验并不友好。同时，在管理域名-IP 映射关系表时，管理员只能暂时关闭服务器，对文本文件进行增添、修改、删除等操作，而不能在程序内直接对该映射表的内容进行修改，给服务端的运营和管理带来一定不便。

2. 系统安全性有待提升

由于服务器的负载能力有限，当程序一次性收到过量 DNS 解析请求包时，可能会出现服务器卡顿的情况。服务器没有设计对于同一用户短时间内大量发出 DNS 请求做出限制，这一点可能会为遭遇网络黑客攻击埋下隐患。若有网络黑客在短时间内发送大量 DNS 请求数据包，DNS 服务器将很可能瘫痪。

3. 缓存资源没有记录 IPV6 地址

开这样的缓存是不完整，我们实现了 IPV4 的缓存，但可以通过单独建立 IPV6 缓存哈

希链表来解决 IPV6 资源缓存的问题。

5.3 实验心得

在计算机网络课程设计中，我们三个成员相互协作，共同完成了 DNS 中继服务器的设计与实现。通过完成本地域名解析、中继功能、不良网站拦截、超时处理、多并发等任务，实现了良好处理客户端域名解析请求的功能。

在任务前期，由于 DNS 并不是计网课程的核心重点，除了仅仅了解 DNS 报文解析过程这些浅显的知识以外，可以说其他的東西还很不熟悉，甚至连 DNS 报文也没有计网中其他的报文格式记忆准确，于是我们学习了很久 DNS 报文格式，尤其是后来资源记录部分的学习，还有认真学习了 A 记录 CNAME 记录等多挣资源记录的格式，还有报文压缩，这些都是实现实验的基础知识，也是最重要的知识。

在实验的过程中，我们对于网络通信函数有了更深刻的掌握，以前在实现多用户的时候经常用，但也不太清楚它的原理，这次算是对网络通信的函数有了更加深刻的理解。

这次的计网课程设计可以说是所有课程设计中最有成就感的，因为它真的可以在我的电脑实现一些我比较常用的功能，感觉非常开心。当然最值得和最重要的当然是在这次实践课中，我们确实学到了很多東西，比如报文的格式，比如 TTL 时间系统时钟，SOCKET 的真实含义，网络通信函数等等一大堆具有挑战性的东西，当时刚刚拿到题目的一筹莫展，到现在回头去看，确实收获颇丰。