**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** | 计算机学院 | **指导教师** | 蒋砚军 |
| **班 级** | **班内序号** | **学 号** | | **学生姓名** | **成绩** | |
| 2019211303 | 9 | 2019211205 | | 陈荣第 |  | |
|  |  |  | |  |  | |
|  |  |  | |  |  | |
| **课**  **程**  **设**  **计**  **内**  **容** | **基本内容：**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。若为不良网站则实现屏蔽功能；若为普通域名，则向用户返回其对应的IP地址；若未搜索到该域名，则向外部DNS服务器发出请求，并将结果返回会给客户端。  **设计方法：**本课程设计采用C++语言对DNS中继服务器进行开发；  **实验成果：**本次实验最终完成了具有不良网站拦截功能、服务器功能以及中继功能的DNS服务器系统；团队还对服务器功能进行了拓展，有针对性地设计了服务器的高速缓存（Cache）系统，提升了DNS服务器的效率。  **其它内容：**socket下UDP编程，设计哈希表以及使用。 | | | | | |
| **学生**  **课程设计**  **报告**  （附页） | 见附页设计报告以及程序源代码 | | | | | |
| **课**  **程**  **设**  **计**  **成**  **绩**  **评**  **定** | **评语**:  **成绩:**  指导教师签名：  年 月 日 | | | | | |

1. **实验内容**

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，实现下 列三种情况：

1. 检索结果为普通 IP 地址，则向客户返回这个地址(即 DNS 服务器功能)；
2. 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(即不良网站拦截功能)；
3. 表中未检到该域名，则向实际的本地 DNS 服务器发出查询， 并将结果返给客户端(即 DNS 中继功能)，同时程序可支持多客户端同时查询的情况(进行消息ID的转换)。

DNS（域名系统）简述及其具体工作过程：

域名系统（英文：Domain Name System，缩写：DNS）是互联 网的一项服务。它作为将域名和 IP 地址相互映射的一个分布式数 据库，能够使人更方便地访问互联网。DNS 在传输层主要使用 UDP 端口 53，其采用 Client-Server 模式，具有管理域名、域名-IP 地址转换等功能。当前，对于每一级域名长度的限制是 63 个字符， 域名总长度则不能超过 253 个字符。

1. **实验环境**

运行环境：Windows 10操作系统

编译：Visual Studio 2019 集成开发环境

语言：使用C语言进行开发

1. **实验目的**
2. 了解掌握DNS使用域名解析ip的具体过程。
3. 了解使用Socket Udp协议进行通信。
4. 使用合理的数据结构存储cache和存储本地文件信息
5. **整体分析及架构**
6. **正常DNS功能**

将计算机DNS服务器地址设置为本地回环地址127.0.0.1指向本机，当用户输入域名之后，DNS查询本地存储的域名-IP对照表以及告诉缓存cache，若将查询到的ip地址返回给用户，使客户端能够正常的访问此域名相应的网站。

1. **不良网站拦截功能**

当用户想要访问某个网站并输入其域名之后，DNS中继服务器解析其域名信息并得到对应的ip,如果此域名对应的ip地址全为0，即“0.0.0.0”，向客户端返回特殊的数据包，实现网站屏蔽功能。

1. **DNS中继功能**

当DNS中继服务器本地的域名-IP对照表和高速缓存cache中查询不到域名对应的ip地址，则向外部DNS服务器发出查询请求，等待外部相应包返回，再返回给客户端，同时将此域名-IP存入高速缓存cache中，以便之后访问相同网站一致。

1. **多客户端并发处理能力**

DNS服务器能够同时收到位于不同计算机上多个客户端的查询需求，即需要允许第一个查询商为得到答案之前能够启动并且处理另外一个客户端查询请求。因此我们需要使用DNS协议中ID字段的作用，为DNS服务器设计一个ID转换表，对不同进程或用户的请求进行区分，最后将查询结果能够返回到正确的客户端.。

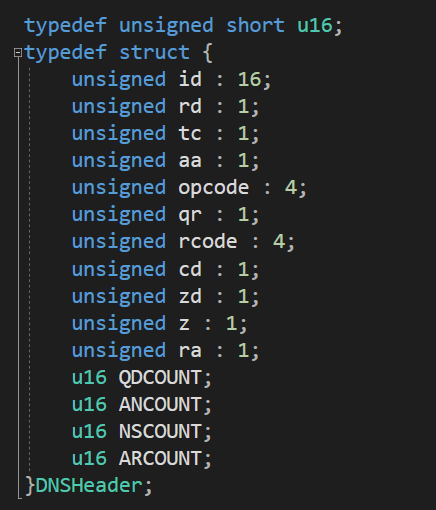
1. **超时处理**

由于我们的Socket通信选择的是·无连接的不可靠的udp协议，因此外部的DNS服务器很可能出现未收到请求包或者请求包丢失的情况而导致无法收到应答。由于发出查询请求到的收到相应报文之间存在一定的时间，我们可以选择一个合适的时间阈值来判断ID转换表中这个ID的请求是否超时，若超时我们删除掉ID转换表中的这一项，用新的用户请求将其覆盖，避免了DNS服务器资源被长期占用而造成内存资源浪费和卡顿。

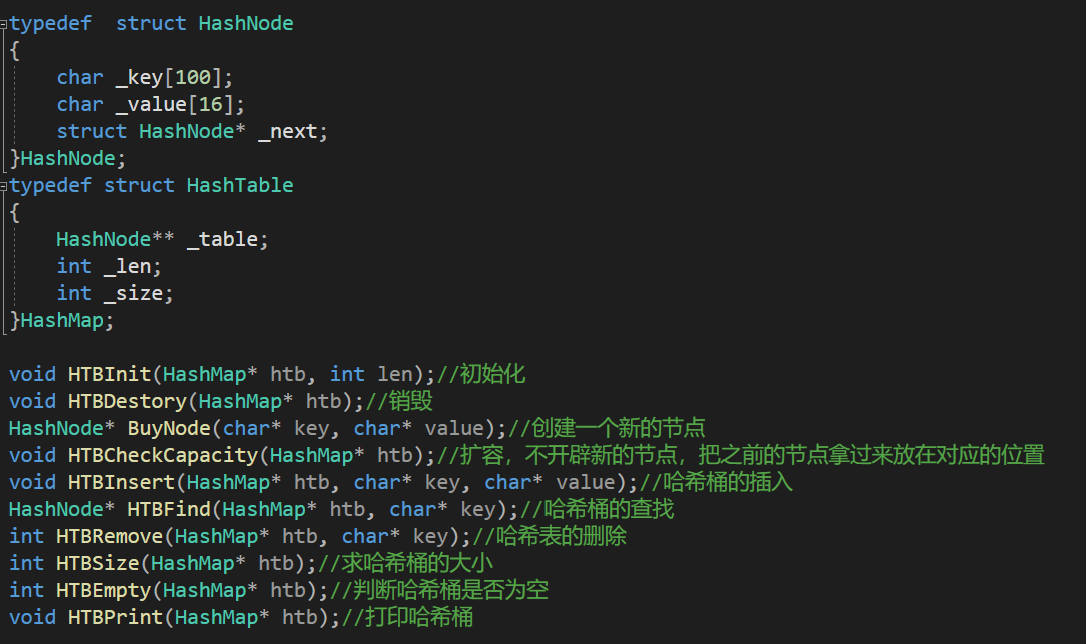
**五、软件设计**

1. **数据结构**

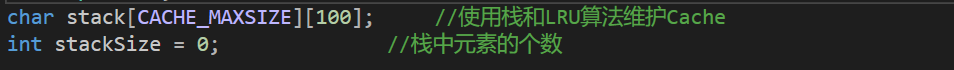
i. 使用指定bit的结构来保存DNS报文头



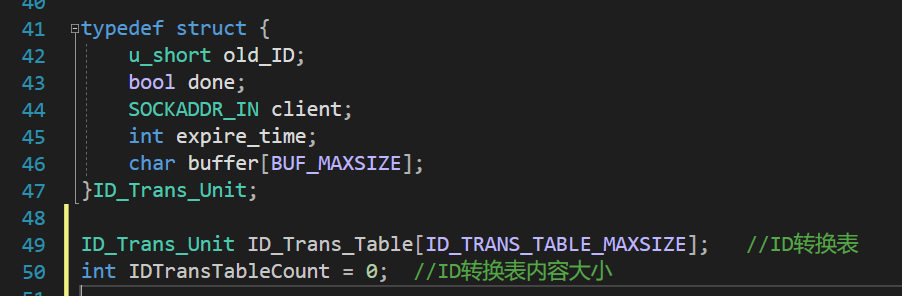
ii. 使用结构体实现哈希桶来存储本地文件。



iii 使用栈来保存cahce中存在的域名。（使用LRU算法来更新cache,当cache满之后将其中最不常用的域名用新域名覆盖）（用双向链表设计cache插入删除时间复杂度降低）



iv. ID转换表，负责DNS中继器与外部服务器之间报文的更新



1. **全局变量及定义**

#define BUF\_MAXSIZE 1024

#define DNS\_PORT 53

#define ID\_TRANS\_TABLE\_MAXSIZE 20

#define ID\_EXPIRE\_TIME 10

#define CACHE\_MAXSIZE 7

#define DNS\_HEAD\_SIZE 12

#define LOCAL\_FILE\_MAXSIZE 1000

int debug\_level = 1; // Debug level

char DNS\_Server\_IP[16] = "114.114.114.114"; // 服务器IP

WSADATA wsaData;

SOCKET local\_socket; 中继器Scoket

struct sockaddr\_in client, server;

int client\_size = sizeof(struct sockaddr\_in);

HashMap local; 存储本地域名-IP对照表和Cache的哈希表

1. **函数说明**
2. 打印标题，学号姓名.

extern void printListTitle(void);

1. 处理命令行输入

extern void Process\_Parameters(int argc,char\* argv[]);

1. 初始化ID转换表，将每一项赋值为初始值

extern void initIDTransTable(void);

1. 将本地文件中的域名-IP对照表插入哈希表当中

extern void readIPURLReflectTable(void);

1. 处理socket从外部服务器发来的相应报文

extern void Receive\_From\_Out(char\* rcvbuf, int length, DNSHeader\* header);

1. 处理socket从客户端发来的请求报文

extern void Receive\_From\_Local(char\* rcvbuf, int length);

1. 将报文中的域名转换为字母和.的形式，即3www5baidu3com ----🡪 www.baidu.com

extern void Convert\_to\_Url(char\* buf, char\* dest);

1. 打印收到的一个包的详细信息，调试时使用

extern void Output\_Packet(char\* buf, int length);

1. 将新的域名-IP更新到cache的同时加入哈希表当中

extern void Add\_Record\_to\_Cache(char\* url, char\* ip);

1. 使用LRU算法更新告诉缓存cache

extern void updated\_Cache(char\* url);

1. 为从客户端发来的在本地找不到ip的请求报文注册一个新的ID发到外部服务器

extern unsigned short Register\_New\_ID(unsigned short ID, SOCKADDR\_IN temp,char rcvbuf[]);

1. **DNS中继服务器流程**

设置新ID发到外部服务器查询

设置组装响应包

将响应包发回客户端

本地存在

客户端请求包

将响应包的ID变为原来的ID

外部服务器响应包

等待接收包

建立套接字信息，端口绑定客户端

初始化ID转换表、哈希表等等

开始

qr == 1响应包 qr == 0询问包

Y

N

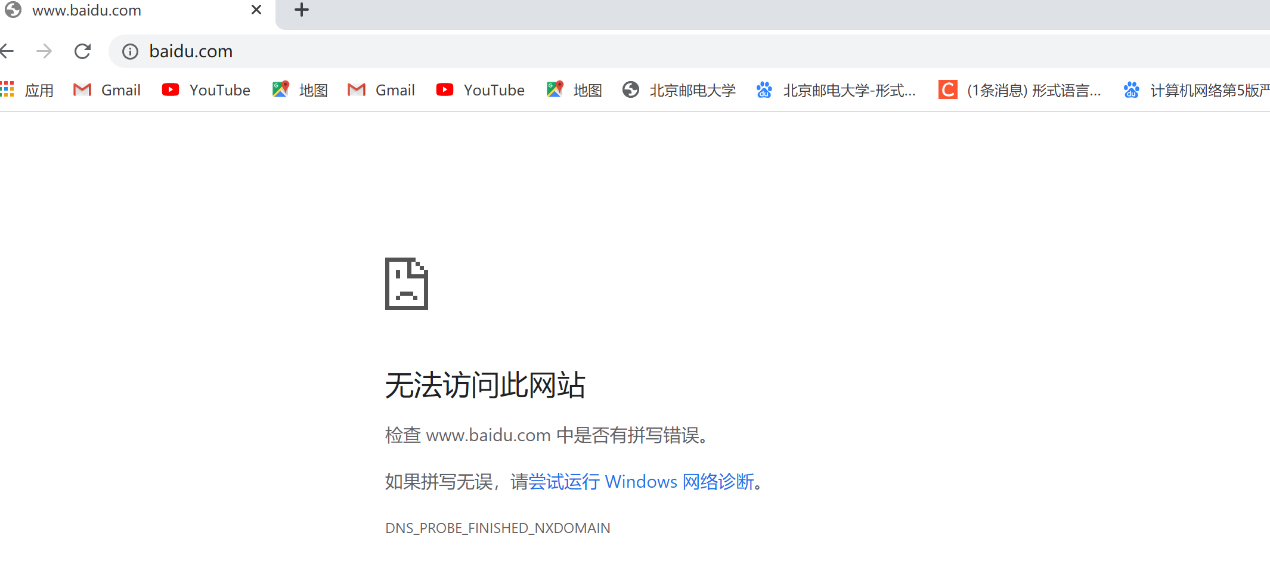
**六、 实验结果及其分析**

测试前先将计算机本地TCP/IPv4协议配置属性中的DNS服务器设置为本地回环地址”127.0.0.1”。

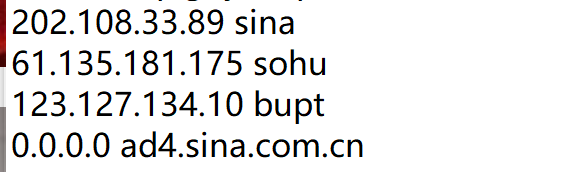
1. **屏蔽功能**



添加0.0.0.0 [www.baidu.com](http://www.baidu.com)之后



1. **服务器功能：本地cache或静态表以及保存对应域名ip**

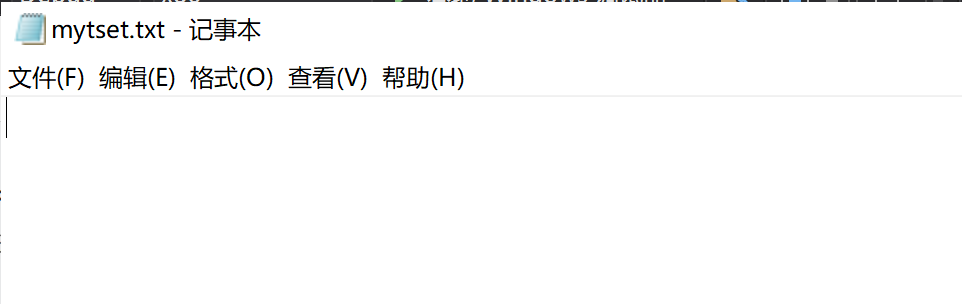
****

****

能够跳转到相应得网页。

1. **中继功能：本地cache未查到，向外部服务器转发**

读入一个空文件，当输入域名之后本地找不到，只能进行转发中继

****

****

读入空文件后仍能加载出网页，能够完成中继功能。

1. **单用户多请求处理**

****

能够同时打开多个快速网页

**七、实验总结及心得**

对于哈希表的设计，为了减少哈希冲突，在表项结构体中加入一个能够指向同类结构的指针变量，这样就变成了哈希桶。对于字符串函数的哈希函数设计，我借用了网上最流行的方式，通过一个种子和字符串自生的每一个值进行计算，其它哈希表操作在测试过程中都出现了一些小bug，几经测试之后能够完成基本要求。

对于Cache的设计，我将它的域名-IP对照与本地文件存于同一个哈希表。起初我是选择了栈使用LRU算法来更新栈中的最不常用元素，但这样数据结构插入删除时间复杂度较高，验收之后了解到使用双向链表在此更为合适。

实验起初，对Socket并不熟悉，通过在网上学习逐渐了解到UDP协议中函数的具体作用以及如何使用。

1. int sendto(int s, const void \*buf, int len, unsigned int flags, const struct sockaddr \*to, int tolen);

发送成功则返回实际传送出去的字符数，失败返回-1，错误原 因会存于 error 中。

s：socket 描述符；

buf：UDP 数据报缓存区（包含待发送数据）；

len：UDP 数据报的长度；

flags：调用方式标志位（一般设置为 0）；

to：指向接收数据的主机地址信息的结构体（sockaddr\_in 需类 型转换）；

tolen：to 所指结构体的长度。

1. int recvfrom(int s, void \*buf, int len, unsigned int flags, struct sockaddr \*from, int \*fromlen);

接收成功则返回实际收到的字符数，失败返回-1，错误原因会 存于 error 中。

s：socket 描述符；

buf：UDP 数据报缓存区（包含所接收的数据）；

len：缓冲区长度。

flags：调用操作方式（一般设置为 0）。

from：指向发送数据的客户端地址信息的结构体（sockaddr\_in 需类型转换）；

fromlen：指针，指向 from 结构体长度值

通过这次课程设计的学习以及完成，让我对域名IP，DNS有了更为全面的认识。这次实验首次让我接触到了网络编程并且对其进行实践，之前都是一直调用C++的哈希函数，这次设计哈希桶也让我对哈希的设计有了更加深层次的认识，哈希冲突是什么，如何去解决哈希冲突。

**八、 源代码**

**1.哈希表声明hash\_table.h**

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

#include<assert.h>

#include<malloc.h>

#include<string.h>

typedef struct HashNode

{

char \_key[100];

char \_value[16];

struct HashNode\* \_next;

}HashNode;

typedef struct HashTable

{

HashNode\*\* \_table;

int \_len;

int \_size;

}HashMap;

void HTBInit(HashMap\* htb, int len);//初始化

void HTBDestory(HashMap\* htb);//销毁

HashNode\* BuyNode(char\* key, char\* value);//创建一个新的节点

void HTBCheckCapacity(HashMap\* htb);//扩容，不开辟新的节点，把之前的节点拿过来放在对应的位置

void HTBInsert(HashMap\* htb, char\* key, char\* value);//哈希桶的插入

HashNode\* HTBFind(HashMap\* htb, char\* key);//哈希桶的查找

int HTBRemove(HashMap\* htb, char\* key);//哈希表的删除

int HTBSize(HashMap\* htb);//求哈希桶的大小

int HTBEmpty(HashMap\* htb);//判断哈希桶是否为空

void HTBPrint(HashMap\* htb);//打印哈希桶

**2.哈希表函数实现hash\_table.cpp**

#include"hash\_table.h"

void HTBInit(HashMap\* htb, int len)//初始化

{

assert(htb);

htb->\_len = len;//最开始表的长度

htb->\_size = 0;

htb->\_table = (HashNode\*\*)malloc(sizeof(HashNode\*) \* htb->\_len);//开辟表长的空间

memset(htb->\_table, NULL, sizeof(HashNode\*) \* htb->\_len);//把表初始化为零

}

void HTBDestory(HashMap\* htb)//销毁

{

//1.assert

assert(htb);

int i = 0;

for (i = 0; i < htb->\_len; i++)

{

HashNode\* cur = htb->\_table[i];

//2.判断这个节点是否为空，不为空销毁节点

while (cur)

{

HashNode\* next = cur->\_next;

free(cur);

cur = next;

}

htb->\_table[i] = NULL;//防止野指针的出现

}

//3.销毁表

free(htb->\_table);

htb->\_table = NULL;

htb->\_size = htb->\_len = 0;

printf("链表销毁成功\n");

}

unsigned long int GetNextPrime(int value)

{

int i = 0;

static const unsigned long \_PrimeList[28] =

{

53ul, 97ul, 193ul, 389ul, 769ul,

1543ul, 3079ul, 6151ul, 12289ul, 24593ul,

49157ul, 98317ul, 196613ul, 393241ul, 786433ul,

1572869ul, 3145739ul, 6291469ul, 12582917ul, 25165843ul,

50331653ul, 100663319ul, 201326611ul, 402653189ul, 805306457ul,

1610612741ul, 3221225473ul, 4294967291ul

};

for (i = 0; i < 28; i++)

{

if (\_PrimeList[i] > value)

{

return \_PrimeList[i];

}

}

return \_PrimeList[27];

}

HashNode\* BuyNode(char\* key, char\* value)//创建一个新的节点

{

HashNode\* node = (HashNode\*)malloc(sizeof(HashNode));//开辟一个节点大小的空间

memcpy(node->\_key, key, sizeof(char) \* 100);

memcpy(node->\_value, value, sizeof(char) \* 16);

node->\_next = NULL;

return node;

}

int CHashFunc(char\* key, int len)//计算字符串的插入位置

{

int count = 0;

int i = 0;

int seed = 31;//31,131(种子)

while (\*(key + i) != '\0')

{

count += count \* seed + \*(key + i);

++i;

}

if (count < 0) count \*= -1;

return count % len;

}

void HTBCheckCapacity(HashMap\* htb)//扩容，不开辟新的节点，把之前的节点拿过来放在对应的位置

{

assert(htb);

if (htb->\_len == htb->\_size)

{

HashMap newhtb;

newhtb.\_len = GetNextPrime(htb->\_len);//新表的表长根据素数表确定

HTBInit(&newhtb, newhtb.\_len);

int i = 0;

for (i = 0; i < htb->\_len; i++)

{

HashNode\* cur = htb->\_table[i];

while (cur)

{

HTBInsert(&newhtb, cur->\_key, cur->\_value);

cur = cur->\_next;

}

}

HTBDestory(htb);//销毁旧表

htb->\_table = newhtb.\_table;//把新表的内容给旧表

htb->\_len = newhtb.\_len;

htb->\_size = newhtb.\_size;

}

}

void HTBInsert(HashMap\* htb, char\* key,char\* value)//哈希桶的插入

{

assert(htb);

HTBCheckCapacity(htb);

int index = CHashFunc(key, htb->\_len);

HashNode\* cur = htb->\_table[index];

HashNode\* NewNode = NULL;

while (cur) //判断哈希桶中是否有key,如果存在则修改ip

{

if (strcmp(cur->\_key, key) == 0) {

memcpy(cur->\_value, value, sizeof(char) \* 16);

return ;

}

else {

cur = cur->\_next;

}

}

//头插法，插入节点

NewNode = BuyNode(key, value);

NewNode->\_next = htb->\_table[index];//NewNode->next保存htb->table[index]对应的链表，htb->table[index]为头结点

htb->\_table[index] = NewNode;//让NewNode做htb->\_table[index]的头结点

htb->\_size++;

}

HashNode\* HTBFind(HashMap\* htb, char\* key)//哈希表的查找

{

assert(htb);

int index = CHashFunc(key, htb->\_len);

HashNode\* cur = htb->\_table[index];

while (cur) {

if (strcmp(cur->\_key, key) == 0)

return cur; //找到了

else

cur = cur->\_next;

}

return NULL;//没找到

}

int HTBRemove(HashMap\* htb, char\* key)//哈希表的删除

{

assert(htb);

int index = CHashFunc(key, htb->\_len);

HashNode\* cur = htb->\_table[index];

HashNode\* prev = NULL;

//直接删除法

while (cur)

{

if (strcmp(cur->\_key, key) == 0)

{

//寻找key

if (prev == NULL)//key为第一个节点

htb->\_table[index] = cur->\_next;

else//不是第一个节点

prev->\_next = cur->\_next;

//删除Key

free(cur);

htb->\_size--;

return 0;

}

prev = cur;//让prev指向cur的前一个节点

cur = cur->\_next;

}

}

int HTBSize(HashMap\* htb)//求哈希桶的大小

{

assert(htb);

return htb->\_size;

}

int HTBEmpty(HashMap\* htb)//判断哈希桶是否为空

{

assert(htb);

return htb->\_size ? 0 : 1;//空为0，非空为1

}

void HTBPrint(HashMap\* htb)//打印哈希桶

{

assert(htb);

int i = 0;

for (i = 0; i < htb->\_len; i++)

{

HashNode\* cur = htb->\_table[i];//创建一个节点去遍历哈希桶

printf("Table");

while (cur)//cur不为空进循环打印

{

printf("-->%s:%s", cur->\_key, cur->\_value);

cur = cur->\_next;

}

printf("-->NULL\n");

}

printf("\n");

}

1. **结构体全局变量函数声明head.h**

#include<WinSock2.h>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<fstream>

#include<string.h>

#include "hash\_table.h"

#pragma comment(lib,"Ws2\_32.lib")

#define BUF\_MAXSIZE 1024

#define DNS\_PORT 53

#define ID\_TRANS\_TABLE\_MAXSIZE 20

#define ID\_EXPIRE\_TIME 10

#define CACHE\_MAXSIZE 7

#define DNS\_HEAD\_SIZE 12

#define LOCAL\_FILE\_MAXSIZE 1000

#define bool int

#define true 1

#define false 0

typedef unsigned short u16;

typedef struct {

unsigned id : 16;

unsigned rd : 1;

unsigned tc : 1;

unsigned aa : 1;

unsigned opcode : 4;

unsigned qr : 1;

unsigned rcode : 4;

unsigned cd : 1;

unsigned zd : 1;

unsigned z : 1;

unsigned ra : 1;

u16 QDCOUNT;

u16 ANCOUNT;

u16 NSCOUNT;

u16 ARCOUNT;

}DNSHeader;

typedef struct {

u\_short old\_ID;

bool done;

SOCKADDR\_IN client;

int expire\_time;

char buffer[BUF\_MAXSIZE];

}ID\_Trans\_Unit;

ID\_Trans\_Unit ID\_Trans\_Table[ID\_TRANS\_TABLE\_MAXSIZE]; //ID转换表

int IDTransTableCount = 0; //ID转换表内容大小

int debug\_level = 1; // Debug level

char DNS\_Server\_IP[16] = "114.114.114.114"; // 服务器IP

WSADATA wsaData;

SOCKET local\_socket;

struct sockaddr\_in client, server;

int client\_size = sizeof(struct sockaddr\_in);

HashMap local;

char stack[CACHE\_MAXSIZE][100]; //使用栈和LRU算法维护Cache

int stackSize = 0; //栈中元素的个数

extern void printListTitle(void);

extern void Process\_Parameters(int argc,char\* argv[]);

extern void initIDTransTable(void);

extern void readIPURLReflectTable(void);

extern void Receive\_From\_Out(char\* rcvbuf, int length, DNSHeader\* header);

extern void Receive\_From\_Local(char\* rcvbuf, int length);

extern void Convert\_to\_Url(char\* buf, char\* dest);

extern void Output\_Packet(char\* buf, int length);

extern void Add\_Record\_to\_Cache(char\* url, char\* ip);

extern void updated\_Cache(char\* url);

extern unsigned short Register\_New\_ID(unsigned short ID, SOCKADDR\_IN temp,char rcvbuf[]);

1. **主函数main.cpp**

#include "head.h"

#pragma warning(disable : 4996)

int main(int argc, char\* argv[]) {

printListTitle();

Process\_Parameters(argc, argv);

initIDTransTable();

int ret;

// 初始化WinSocket服务,初始化套接字,并判断是否成功

if ((ret = WSAStartup(MAKEWORD(2, 2), &wsaData)) != 0) {

printf("Error:WSAStartup failed with error %d\n", ret);

exit(1);

}

else {

if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) {

printf("Error: not winsock 2.2\n");

WSACleanup(); //清空启动信息

exit(1);

}

}

// 创建本地的Socket

if ((local\_socket = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP)) < 0) {

printf("Error: socket failed with error %d\n", WSAGetLastError());

WSACleanup();

exit(1);

}

// 创建存文件的哈希表以及Cache

for (int i = 0; i < CACHE\_MAXSIZE; ++i) {

memset(stack[i], 0x00, sizeof(char) \* 100);

}

HTBInit(&local, LOCAL\_FILE\_MAXSIZE + CACHE\_MAXSIZE);

// 读取本地文件中的IP域名对照表存入HashMap

readIPURLReflectTable();

//HTBPrint(&local);

// 将Socket接口改为无阻塞模式

int non\_block = 1;

ioctlsocket(local\_socket, FIONBIO, (u\_long FAR\*) & non\_block);

// 检查本地socket是否创建成功

if (local\_socket < 0)

{

if (debug\_level >= 1)

printf("Create local socket failed.\n");

exit(1);

}

printf("Create local socket successfully.\n");

client.sin\_family = AF\_INET; // Address family AF\_INET代表TCP / IP协议族

client.sin\_addr.s\_addr = INADDR\_ANY; // 设置本地地址为任意IP地址,接收任意IP发来的数据

client.sin\_port = htons(DNS\_PORT); // 设置DNS接口为53

server.sin\_family = AF\_INET; // Address family AF\_INET代表TCP / IP协议族

server.sin\_addr.s\_addr = inet\_addr(DNS\_Server\_IP); // 设置外部DNS服务器IP地址

server.sin\_port = htons(DNS\_PORT); // 设置DNS接口为53

// 设置套接字的选项,避免出现本地端口被占用情况

int reuse = 1;

setsockopt(local\_socket, SOL\_SOCKET, SO\_REUSEADDR, (const char\*)&reuse, sizeof(reuse));

// 绑定该套接字到53端口

if (bind(local\_socket, (struct sockaddr\*)&client, sizeof(client)) == SOCKET\_ERROR)

{

if (debug\_level >= 1)

printf("Bind socket port failed %d.\n", WSAGetLastError());

closesocket(local\_socket);

WSACleanup();

exit(1);

}

printf("Bind socket port successfully.\n");

while (true) {

char rcvbuf[BUF\_MAXSIZE]; //每一级域名长度不能超过63个字符

memset(rcvbuf, 0x00, sizeof(char) \* BUF\_MAXSIZE);

int length = 0;

//struct sockaddr\_in address;

length = recvfrom(local\_socket, rcvbuf, sizeof(rcvbuf), 0, (struct sockaddr\*)&client, &client\_size);

if (length > 0) {

DNSHeader\* header = (DNSHeader\*)rcvbuf;

if (header->qr == 0) { //询问报

Receive\_From\_Local(rcvbuf, length);

}

else { //响应报

Receive\_From\_Out(rcvbuf, length, header);

}

}

}

closesocket(local\_socket);

WSACleanup();

HTBDestory(&local);

return 0;

}

void printListTitle(void)

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\* Course Name: Course Design of Computer Network \*\n");

printf("\* Name of Team members: Chen Rongdi \*\n");

printf("\* Class number: 2019211303 \*\n");

printf("\* ------------------------------------------------------- \*\n");

printf("\* DNS Relay Server - Ver 1.0 \*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Command syntax : dnsrelay [-d | -dd] [dns-server-IP-addr] \n");

}

void Process\_Parameters(int argc, char\* argv[])

{

bool user\_set\_dns\_flag = false;

if (argc > 1 && argv[1][0] == '-')

{

if (argv[1][1] == 'd') debug\_level++; /\* Debug level add to 1 \*/

if (argv[1][2] == 'd') debug\_level++; /\* Debug level add to 2 \*/

if (argc > 2)

{

user\_set\_dns\_flag = 1; /\* If user set the dns server ip address \*/

strcpy\_s(DNS\_Server\_IP, argv[2]);

}

}

if (user\_set\_dns\_flag) /\* If user set the dns server ip address \*/

printf("Set DNS server : %s\n", argv[2]);

else /\* If user do not set the dns server ip address, set it by default \*/

printf("Set DNS server : %s by default\n", DNS\_Server\_IP);

printf("Debug level : %d\n", debug\_level);

}

void initIDTransTable(void) {

/\* Initialize the ID transfer table \*/

for (int i = 0; i < ID\_TRANS\_TABLE\_MAXSIZE; i++)

{

ID\_Trans\_Table[i].old\_ID = 0;

ID\_Trans\_Table[i].done = TRUE;

ID\_Trans\_Table[i].expire\_time = 0;

memset(&(ID\_Trans\_Table[i].client), 0, sizeof(SOCKADDR\_IN));

}

}

void readIPURLReflectTable(void)

{

FILE\* file;

if ((file = fopen("mytset.txt", "r")) == NULL)

return;

char url[100] = {}, ip[16] = {};

while (fscanf(file, "%s %s", ip, url) > 0) {

if (debug\_level >= 1)

printf("Read from 'dnsrelay.txt' -> [Url : %s, IP : %s]\n", url, ip);

HTBInsert(&local, url, ip);

}

fclose(file);

}

void Convert\_to\_Url(char\* buf, char\* dest)

{

/\* 将形如6github3com转化为github.com \*/

int i = 0, k = 0;

int lableLength = buf[i];

i++;

while (buf[i] != '\0')

{

for (int j = 0; j < lableLength; j++)

{

dest[k] = buf[i];

i++;

k++;

}

if (buf[i] != '\0')

{

dest[k] = '.';

k++;

lableLength = buf[i];

i++;

}

}

dest[k] = '\0';

}

void updated\_Cache(char\* url) {

for (int i = 0; i < stackSize; ++i) {

if (strcmp(stack[i], url) == 0) {

for (int j = i; j < stackSize - 1; ++j) {

memcpy(stack[j], stack[j + 1], sizeof(char) \* 100);

}

memcpy(stack[stackSize - 1], url, sizeof(char) \* 100);

}

}

}

void Add\_Record\_to\_Cache(char\* url, char\* ip) {

HashNode\* it = HTBFind(&local, url);

if (it != NULL) {

memcpy(it->\_value, ip, sizeof(char) \* 16);

}

else {

if (stackSize == CACHE\_MAXSIZE) {

// 删除其中使用最不频繁的一个域名

char\* lru\_url = stack[0];

HTBRemove(&local, lru\_url);

--stackSize;

for (int i = 0; i < stackSize; ++i) {

memcpy(stack[i], stack[i + 1], sizeof(char) \* 100);

}

}

HTBInsert(&local, url, ip);

memcpy(stack[stackSize++], url, sizeof(char) \* 100);

}

}

void Output\_Packet(char\* buf, int length)

{

unsigned char unit;

printf("Packet length = %d\n", length);

printf("Details of the package:\n");

int x = 0;

for (int i = 0; i < length; i++)

{

unit = (unsigned char)buf[i];

printf("%02x ", unit); //十六进制输出，域宽为2,右对齐

if (x == 16) {

printf("\n");

x = 0;

}

else {

++x;

}

}

printf("\n");

}

unsigned short Register\_New\_ID(unsigned short ID, SOCKADDR\_IN temp,char rcvbuf[]) {

for (int i = 0; i < ID\_TRANS\_TABLE\_MAXSIZE; ++i) {

if ((ID\_Trans\_Table[i].expire\_time > 0 && time(NULL) > ID\_Trans\_Table[i].expire\_time) || ID\_Trans\_Table[i].done) {

ID\_Trans\_Table[i].old\_ID = ID;

ID\_Trans\_Table[i].client = temp;

ID\_Trans\_Table[i].done = false;

ID\_Trans\_Table[i].expire\_time = time(NULL) + ID\_EXPIRE\_TIME;

//memcpy(ID\_Trans\_Table[i].buffer, rcvbuf, sizeof(char) \* BUF\_MAXSIZE);

++IDTransTableCount;

if (debug\_level)

{

printf("New ID : %d registered successfully\n", i + 1);

printf("#ID Count : %d\n", IDTransTableCount);

}

return (unsigned int)i + 1;

}

}

return 0;

}

void Receive\_From\_Out(char rcvbuf[], int length, DNSHeader\* header) {

if (debug\_level) {

printf("\n\n---- Recv : server [IP:%s]----\n", inet\_ntoa(client.sin\_addr));

/\* Output time now \*/

time\_t t = time(NULL);

char temp[64];

strftime(temp, sizeof(temp), "%Y/%m/%d %X %A", localtime(&t));

printf("%s\n", temp);

if (debug\_level == 2)

Output\_Packet(rcvbuf, length);

}

/\* 获取ID \*/

//int id\_index = header->id - 1;

unsigned short\* serverID = (unsigned short\*)malloc(sizeof(unsigned short));

memcpy(serverID, rcvbuf, sizeof(unsigned short));

int id\_index = ntohs((\*serverID)) - 1;

free(serverID);

/\* 将DNS回复包的ID修改成原来请求包的ID \*/

memcpy(rcvbuf, &ID\_Trans\_Table[id\_index].old\_ID, sizeof(unsigned short));

--IDTransTableCount;

ID\_Trans\_Table[id\_index].done = true;

if (debug\_level >= 1)

printf("#ID Count : %d\n", IDTransTableCount);

/\* 获取客户端信息 \*/

int nquery = ntohs(header->QDCOUNT), nresponse = ntohs(header->ANCOUNT);

client = ID\_Trans\_Table[id\_index].client;

char\* searchPtr = rcvbuf + 12; //p指针指向查询问题区域

char ip[16];

int ip1, ip2, ip3, ip4;

char url[100];

memset(url, 0x00, sizeof(char) \* 100);

for (int i = 0; i < nquery; ++i) {

Convert\_to\_Url(searchPtr, url);

while (\*searchPtr > 0)

searchPtr += (\*searchPtr) + 1;

searchPtr += 5;

}

if (nresponse > 0 && debug\_level >= 1)

printf("Receive from extern [Url : %s]\n", url);

/\* 资源记录部分 \*/

for (int i = 0; i < nresponse; ++i) {

if ((unsigned char)\*searchPtr == 0xc0) /\* The name field is pointer \*/

searchPtr += 2;

else /\* The name field is Url \*/

{

while (\*searchPtr > 0)

searchPtr += (\*searchPtr) + 1;

++searchPtr;

}

unsigned short resp\_type = ntohs(\*(unsigned short\*)searchPtr); /\* 资源记录的类型 \*/

searchPtr += 2;

unsigned short resp\_class = ntohs(\*(unsigned short\*)searchPtr); /\* 地址类型 \*/

searchPtr += 2;

unsigned int ttl = ntohs(\*(unsigned int\*)searchPtr); /\* 生存时间，以秒为单位 \*/

searchPtr += 4;

unsigned short dataLen = ntohs(\*(unsigned short\*)searchPtr); /\* 资源数据长度 \*/

searchPtr += 2;

if (debug\_level >= 2)

printf("Type -> %d, Class -> %d, TTL -> %d\n", resp\_type, resp\_class, ttl);

if (resp\_type == 1) {

ip1 = (int)(unsigned char)\*searchPtr++;

ip2 = (int)(unsigned char)\*searchPtr++;

ip3 = (int)(unsigned char)\*searchPtr++;

ip4 = (int)(unsigned char)\*searchPtr++;

sprintf(ip, "%d.%d.%d.%d", ip1, ip2, ip3, ip4);

if (debug\_level)

printf("IP address : %d.%d.%d.%d\n", ip1, ip2, ip3, ip4);

/\* 加入Cache \*/

Add\_Record\_to\_Cache(url, ip);

break;

}

else {

searchPtr += dataLen;

}

}

length = sendto(local\_socket, rcvbuf, length, 0, (SOCKADDR\*)&client, sizeof(client));

}

void Receive\_From\_Local(char rcvbuf[], int length) {

char url[100];

char origin\_url[100];

memcpy(origin\_url, &(rcvbuf[DNS\_HEAD\_SIZE]), length);

Convert\_to\_Url(origin\_url, url);

if (debug\_level) {

printf("\n\n---- Recv : Client [IP:%s]----\n", inet\_ntoa(client.sin\_addr));

/\* Output time now \*/

time\_t t = time(NULL);

char temp[64];

strftime(temp, sizeof(temp), "%Y/%m/%d %X %A", localtime(&t));

printf("%s\n", temp);

printf("Receive from client [Query : %s]\n", url);

}

char\* ip;

HashNode\* It = HTBFind(&local, url);

if (It == NULL) {

printf("[Url : %s] not in local data and cache\n", url);

unsigned short\* pID = (unsigned short\*)malloc(sizeof(unsigned short));

memcpy(pID, rcvbuf, sizeof(unsigned short)); /\* Record ID \*/

unsigned short nID = Register\_New\_ID(\*pID, client, rcvbuf); /\* Register in the ID transfer table \*/

nID = htons(nID);

if (nID == 0) {

if (debug\_level >= 1)

printf("Register failed, the ID transfer table is full.\n");

}

else {

memcpy(rcvbuf, &nID, sizeof(unsigned short));

int len = sendto(local\_socket, rcvbuf, length, 0, (struct sockaddr\*)&server, sizeof(server));/\* Send the request to external DNS server \*/

if (debug\_level >= 1) {

printf("Send to server DNS server [Url : %s]\n", url);

if (len > 0) {

printf("Send Success to server");

}

else {

printf("Send fail to server");

}

}

}

free(pID);

pID = NULL;

}

else {

/\* 在本地(文件或者Cache)中找到 \*/

ip = It->\_value;

updated\_Cache(url);

if (debug\_level) {

printf("In file and cache find result：[url: %s, ip: %s]\n", url, ip);

}

/\* 我们已经能够在本地找到IP，开始自己准备一个响应包发送 \*/

char sendbuf[BUF\_MAXSIZE];

memcpy(sendbuf, rcvbuf, length);

/\* 设置标志位 \*/

unsigned short flags = htons(0x8180);

memcpy(&(sendbuf[2]), &flags, sizeof(unsigned short));

/\* 设置回答资源记录数 \*/

unsigned short cnt;

if (strcmp(ip, "0.0.0.0") == 0) { //需要屏蔽

unsigned short flags = htons(0x8183);

memcpy(&(sendbuf[2]), &flags, sizeof(unsigned short));

cnt = htons(0x0000);

}

else {

cnt = htons(0x0001);

}

memcpy(&(sendbuf[6]), &cnt, sizeof(unsigned short));

/\* 设置资源记录部分 \*/

int offset = 0; //偏移量

char answer[16];

// 1 - 域名

unsigned short name = htons(0xc00c);

memcpy(answer + offset, &name, sizeof(unsigned short));

offset += sizeof(unsigned short);

// 2 - 资源记录的类型

unsigned short typeA = htons(0x0001);

memcpy(answer + offset, &typeA, sizeof(unsigned short));

offset += sizeof(unsigned short);

// 3 - 地址类型

unsigned short classA = htons(0x0001);

memcpy(answer + offset, &classA, sizeof(unsigned short));

offset += sizeof(unsigned short);

unsigned long timeLive = htonl(0x7b);

memcpy(answer + offset, &timeLive, sizeof(unsigned long));

offset += sizeof(unsigned long);

/\*

// 4 - 生存时间

UINT32 timeLive = htonl(0x7b);

memcpy(answer + offset, &timeLive, sizeof(UINT32));

offset += sizeof(UINT32);

\*/

// 5 - 资源数据长度

unsigned short IPLen = htons(0x0004);

memcpy(answer + offset, &IPLen, sizeof(unsigned short));

offset += sizeof(unsigned short);

// 6 - 资源数据

unsigned long IP = (unsigned long)inet\_addr(ip);

memcpy(answer + offset, &IP, sizeof(unsigned long));

offset += sizeof(unsigned long);

/\*

// 6 - 资源数据

UINT32 IP = (UINT32)inet\_addr(ip);

memcpy(answer + offset, &IP, sizeof(UINT32));

offset += sizeof(UINT32);\*/

//最终将资源记录部分复制进待发送缓冲区

offset += length;

memcpy(sendbuf + length, answer, sizeof(answer));

length = sendto(local\_socket, sendbuf, offset, 0, (struct sockaddr\*)&client, sizeof(client));

if (length < 0) {

printf("发送到Client失败\n");

}

else {

printf("发送到Client成功\n");

}

char\* p;

p = sendbuf + length - 4;

if (debug\_level)

printf("发送包信息 [查询名称:%s -> 查询结果:%u.%u.%u.%u]\n", url, (unsigned char)\*p, (unsigned char)\*(p + 1), (unsigned char)\*(p + 2), (unsigned char)\*(p + 3));

}

}