DNS Project

**Course Title:**  *Internet Application*

**Name:** *Jinyi Guo (2011213244)*

*Yiwei Huang (2011213197)*

**Date:**  *June 5, 2014*

1. **Overview**

Two people form a group, basing on the programming experiences of labs and relevant knowledge of DNS protocols, to construct a DNS client-server system based on Linux command line which can complete Chinese character domain name resolution.

**2. Requirements Analysis**

**Development Environment**

1. C programming language and relevant function library.
2. Gcc compiler, gdb debugger and Wireshark capture packet analyzing tool.
3. Linux OS with its commands, nslookup and help.

**Functional requirements**

1. Achieve Chinese character domain name resolution, using “的” as separation symbol.
2. Support at least 4 top level domain including “中国”, “组织”, “商业”, “美国”, and 4 second level domains. One client and at least 4 DNS server (local DNS server, root server, TLD server 1, TLD server 2) should be realized.
3. Messages sent between each client or server should be in DNS message form, which should be able to be captured and identified correctly by Wireshark.
4. Following Resource Record types should be supported: A, NS, MX, CNAME. For NS and MX query, relative IP address should be carried in Additional Section.
5. Iteration method should be used for resolution structure.
6. UDP protocol should be used for communication at Transport Layer between client and local DNS server, while TCP protocol should be used between each DNS server.
7. Use files to maintain database records for DNS servers.
8. Support some simple fault disposes.

**3. Preliminary Design**

**Decomposition of functional modules**

Module 1: Database records retrieve and compare function. This part checks the respective database with the wanted DNS name and type. For root server, it only compares the top level domain with the database records. If the needed record exists, returns this record. Otherwise, indicates the record could not be found.

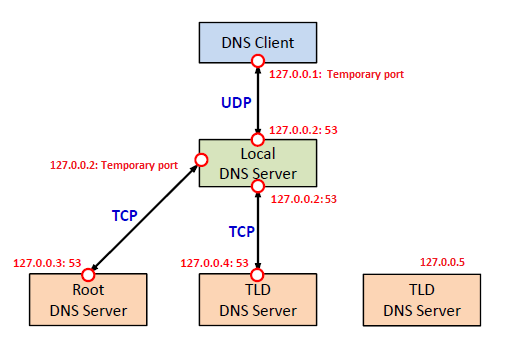
Module 2: Client function. Client port receives user command and sends query to local DNS server with wanted DNS name and type in DNS message form.

Module 3: Local DNS server function. Local server receives client’s message and check its local database. If no record matches the type and name, it sends the same query message to other servers. After asking each server in order where is the domain, it finally returns the result to the client. If local records have such one, then it returns a response message that gives particular answers.

Module 4: Root server function. This server receives the query form local server and sees if the top level domain of the requested name is included in the root database. If so, it returns (in DNS message form) the IP of the TLD server which corresponds to that top level domain. Otherwise it tells local server that no such domain in the network.

Module 5: TLD server function. TLD server also receives query from local server. After that it checks if the TLD database contains asked domain name and type. If such record exists, it returns a response message that gives particular answers. Otherwise it tells no record found, but still in DNS response format. There are two TLD servers but they only different in IP and database record.

**Relationship and interface between the modules**



Above is the communication diagram of module 2, 3, 4 and 5. Client, root server and two TLD servers only communicate with local server, while local server can interact with any other servers. Only one of the two TLD servers is used during one query event, but both TLD servers can communicate with local server. The behavior of server is the behavior of module because the action of server is implemented in a whole function. Besides, module 1 is a set of customized functions and can be called by all the other modules.

The interfaces between module 2, 3, 4 and 5 are the content in the buffers which are sent between each server as packets. The interface between module 1 and other modules is the requested name and type, or the target array record as the parameters.

**Overall flow chart**



**Design of data structures**

struct DNS\_HEADER

{

unsigned short id;

unsigned char rd :1;

unsigned char tc :1;

unsigned char aa :1;

unsigned char opcode :4;

unsigned char qr :1; //

unsigned char rcode :4;

unsigned char cd :1;

unsigned char ad :1;

unsigned char z :1;

unsigned char ra :1;

unsigned short q\_count;

unsigned short ans\_count;

unsigned short auth\_count;

unsigned short add\_count;

};

struct QUESTION

{

unsigned short qtype;

unsigned short qclass;

};

struct RES\_RECORD

{

unsigned short type;

unsigned short \_class;

unsigned int ttl;

unsigned short data\_len;

};

**4. Detailed Design**

**Design analysis of each module**

Module 1: This module consists of two major functions. For servers except root server, the first function reads in two parameters “name” and “type” which we are searching for. At first it changes the data type of “type” from int to char. After that it checks every record in respective database whether its type is the wanted type. If the type is the same, then the function checks the corresponding name. When both the type and name are identical to the value of the parameters, the function returns the matched record. Otherwise the function returns a NULL pointer. For root server, the first function has only one parameter that is the domain name. It just checks if the top level domain of the requested name exists in the root database. If so, it returns the corresponding IP. Otherwise it returns NULL pointer. The second function has two parameters. It split the record given as the first parameter into separate fields and stores them in a two-dimensional array as the second parameter. Then it returns 1 indicating process successful. If the record is void it returns 0 as failed.

Module 2: The function of client first read the user input of target domain name and type. Then it sets up and binds socket to connect with the local server. Client constructs DNS query message and put questions with headers into packet. After that it sends this message packet to the local server and waits for response. When the response packet is received by the client, it resolves the response message, gets the result from Answer (and Additional Section for MX or NS type) and prints them on the screen.

Module 3: The function of local server first sets up and binds socket with client, and waits for DNS query sent from client. Then it receives the query message and resolves it. After that local server extract the requested name and type in the query and check local database whether there is identical record using functions in module 1. If the record is found, local server creates DNS response message with the result in Answer and Additional Section. Otherwise it establishes TCP connection with root server and creates DNS query message with the same content as client query and sends the packet to root server. Then it receives the response from root server, closes TCP link and resolves the message, getting the IP of TLD server. After that it asks TLD server and receives response with the same process. During the whole process, if no other server knows the answer or the address of next level server, local server creates DNS response message indicating that no result found. If any of the other servers returns the result, local server will creates DNS response message with that result in Answer Section. If the type is “MX” or “NS”, it checks database again to find the IP of the mail or domain name server. If found the result will be sent in Additional Section of the response message. At last it sends the response DNS message to local server and waits for client’s new query again.

Module 4: Root server function initializes socket at first and listens for local server’s connection request. Then it establishes TCP connection with local server and waits for query message to come. After receiving the query packet, root server resolves the message and check root database using functions in module 1 if the top level domain of the requested name exists in the records. If so, it makes DNS response message with next TLD server’s IP address in Answer Section. Otherwise it makes no result DNS response message. Finally root server will send the response packet to local server, close the TCP connection and stand by for TCP connection again.

Module 5: Two TLD servers have similar functions. Both function initialize socket at first and listen for local server’s connection request. Then one of the two servers sets up TCP connection with local server and waits for query message to come. After one of them receives the query packet, that server resolves the message and checks its TLD database using functions in module 1 if there is identical type and name in the records. If so, it makes DNS response message with the result in Answer Section. And if the type is “MX” or “NS”, it checks database again to find the IP of the mail or domain name server. If found the result will be sent in Additional Section of the response message. When no record matches, it makes no answer found response message. Finally TLD server sends the response packet to local server, closes the TCP connection and waits for TCP connection again.

**Flow chart of each module**

Module 1:



Module 2:



Module 3:



Model 4:

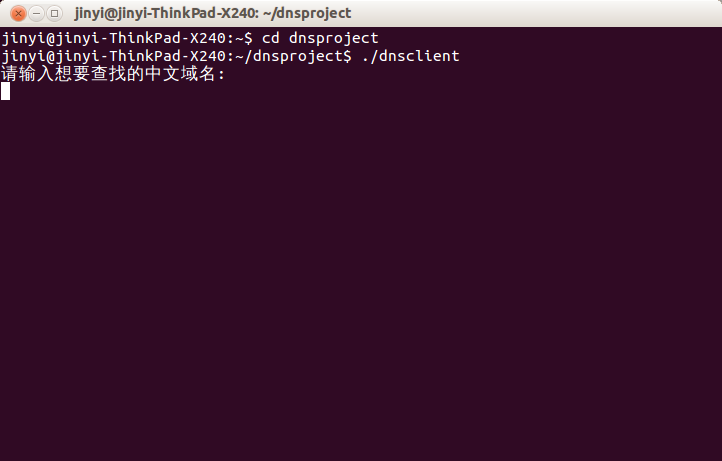
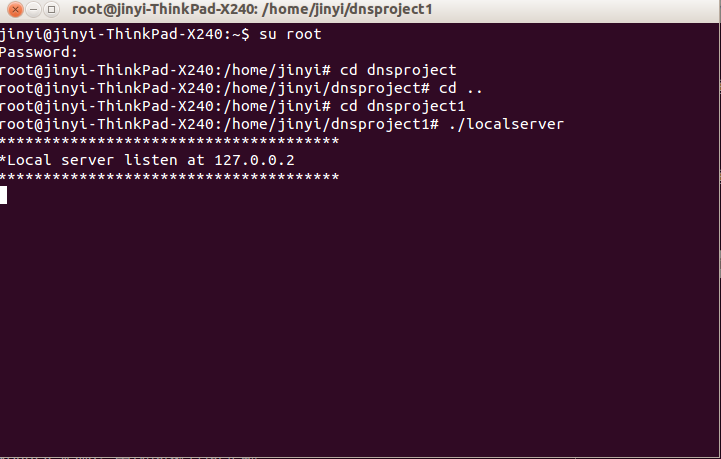


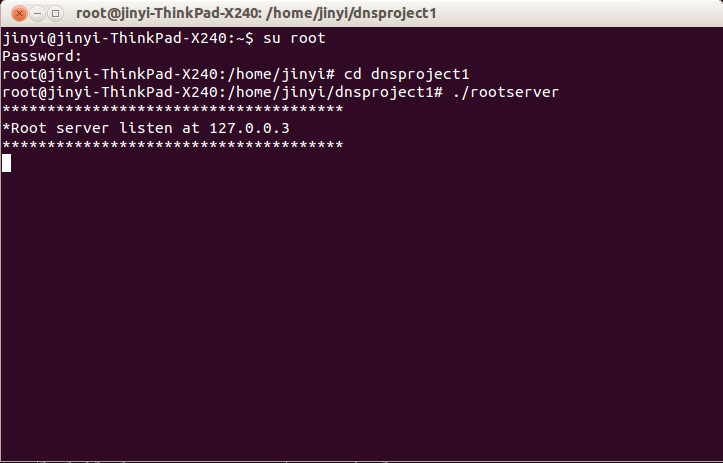
Module 5:

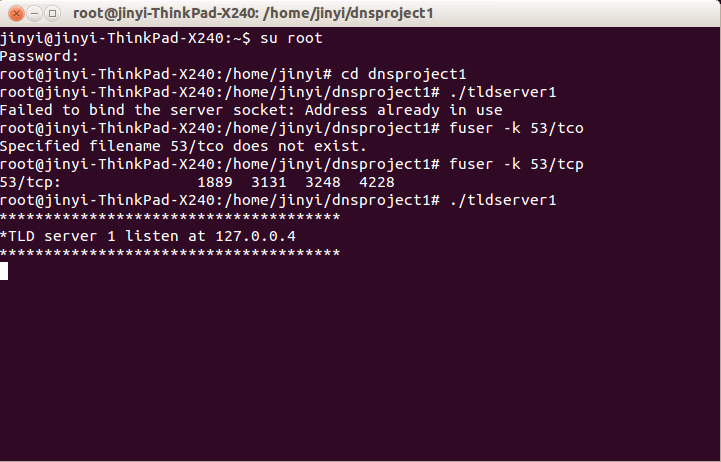


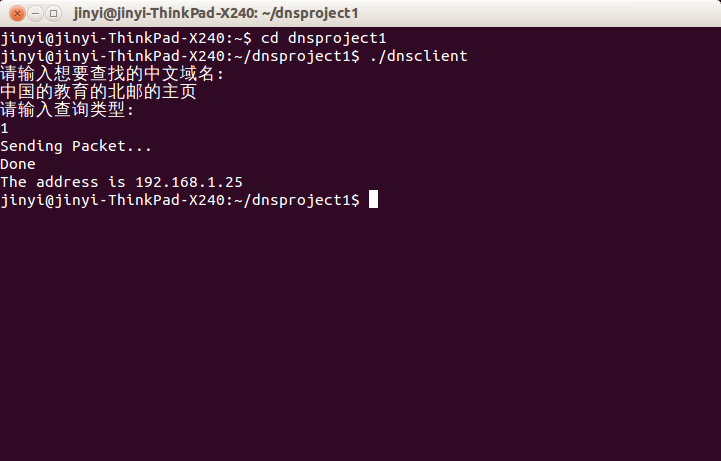
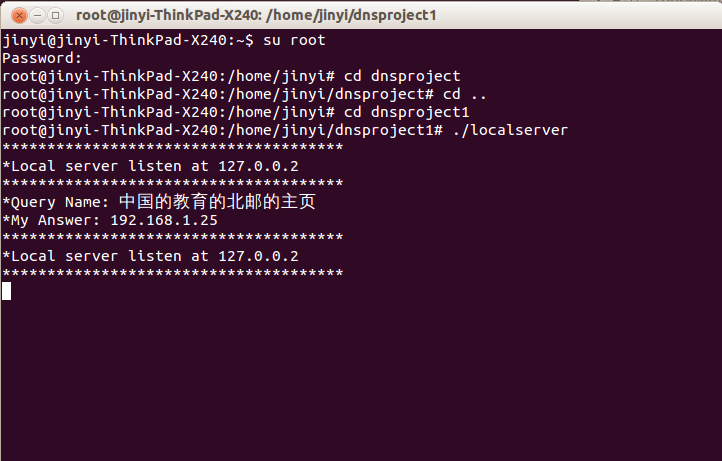
**5. Results**

The initial state of the client and servers:

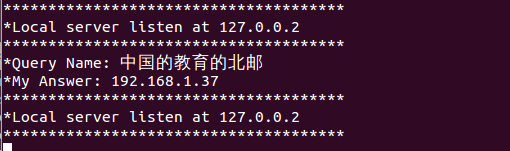
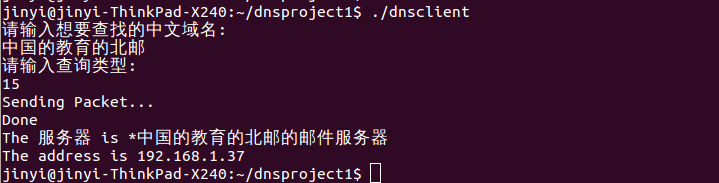




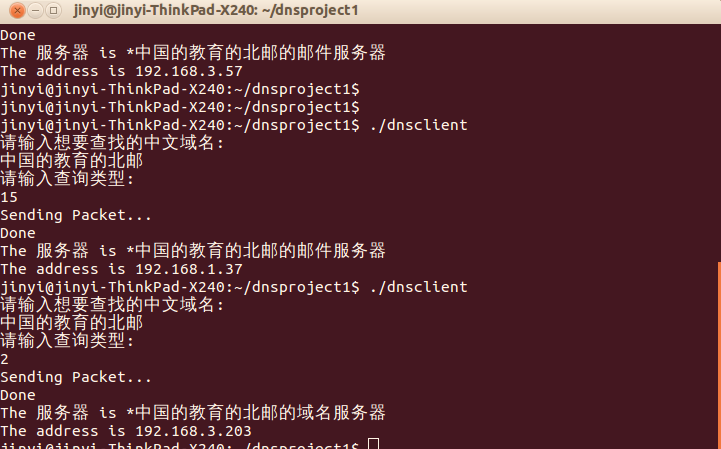
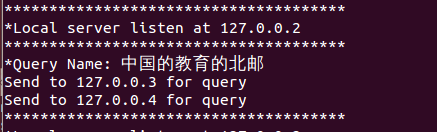


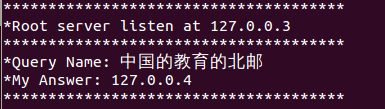
Client queries the domain with “A” type when local server has the same record:

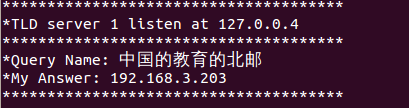
Client queries the domain with “MX” type when local server has the same record.



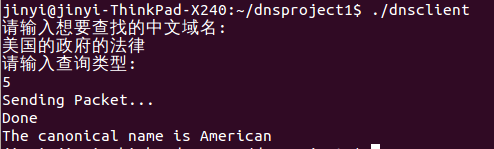
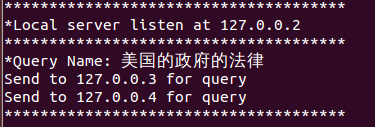
Client queries the domain with “NS” type which local server does not have:

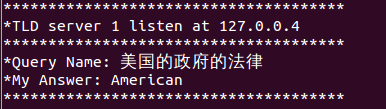






Client queries the domain with “CNAME” type which local server does not have:





**6. Summary and Conclusion**

**Job division:**

Jinyi Guo is responsible for the main structure of the whole system and all major functions of the whole task. He writes code for all the servers and clients to let them able to connect with each other by TCP and UDP. He also designs the data structure and makes the codes for sending, receiving and resolving DNS packets. And he helps writing detailed design part of the report and creates screenshots. Yiwei Huang takes charge of the various functions that supports the system to run correctly. For example, the function that reads file and compares requested search content with file records; the function that split a whole record into fields; the function that changes data type from int to char; and the function that sets up data structure. He also tests the system and helps debugging with Guo in making right packet form to for Wireshark to capture correctly and other problems. And he writes the major part of the report.

**Self-evaluation:**

Jinyi Guo: In the coursework I was mainly in charge of writing the mainframe of the program, including setting up communication between servers and making the eligible DNS packet in both TCP and UDP socket. During programming I deeply realized the importance of a well understanding of DNS packet structure and the use of pointers in C. The entire process of resolving DNS query is quite complicated and thus very likely to make mistakes. It often takes longer to debug than to write code. So I need to be more careful when coding so that better efficiency could be obtained. Plus patience is also very important when debugging since it could be extremely boring and frustrating. I think next time I can have a try to work out extra functions because we didn’t finish any of them in this project.

Yiwei Huang: I mainly acted as an assistant in the whole process. I made basic functions for my partner to let him focus on the main part of the system. I also helped him finding errors and provided suggestions to solve the problem. Although my tasks were not as difficult as my partner because he was responsible for all the complex codes implementing the major functions, I had to revise various parts of the system continuously as minor problems came out all the time. We faced many frustrating errors but we finally managed to solve them. This project greatly improved my ability to communicate with other people because I constantly talked my partner with how to modify my part and how to solve his problems. I also gained deep understanding of the principals of DNS by studying the structures and processes. And of course, my coding ability is improved as well. However, the system only realizes basic requirements. Extra functions are not implemented at all. And our system still has some minor problems that make it seems imperfect. These can be improved in the future.

**Appendix: Source Codes**

**dnsclient.c**

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#define T\_A 1 //Ipv4 address

#define T\_NS 2 //Nameserver

#define T\_CNAME 5 // canonical name

#define T\_SOA 6 /\* start of authority zone \*/

#define T\_PTR 12 /\* domain name pointer \*/

#define T\_MX 15 //Mail server

//DNS header structure

struct DNS\_HEADER

{

unsigned short id; // identification number

unsigned char rd :1; // recursion desired

unsigned char tc :1; // truncated message

unsigned char aa :1; // authoritive answer

unsigned char opcode :4; // purpose of message

unsigned char qr :1; // query/response flag

unsigned char rcode :4; // response code

unsigned char cd :1; // checking disabled

unsigned char ad :1; // authenticated data

unsigned char z :1; // its z! reserved

unsigned char ra :1; // recursion available

unsigned short q\_count; // number of question entries

unsigned short ans\_count; // number of answer entries

unsigned short auth\_count; // number of authority entries

unsigned short add\_count; // number of resource entries

};

//Constant sized fields of query structure

struct QUESTION

{

unsigned short qtype;

unsigned short qclass;

};

//Pointers to resource record contents

struct RES\_RECORD

{

unsigned short type;

unsigned short \_class;

unsigned int ttl;

unsigned short data\_len;

};

//Structure of a Query

typedef struct

{

unsigned char \*name;

struct QUESTION \*ques;

} QUERY;

void set\_header(struct DNS\_HEADER \*dns) {

dns->id = (unsigned short) htons(getpid());

dns->qr = 0; //This is a query

dns->opcode = 0; //This is a standard query

dns->aa = 0; //Not Authoritative

dns->tc = 0; //This message is not truncated

dns->rd = 1; //Recursion Desired

dns->ra = 0; //Recursion not available! hey we dont have it (lol)

dns->z = 0;

dns->ad = 0;

dns->cd = 0;

dns->rcode = 0;

dns->q\_count = htons(1); //we have only 1 question

dns->ans\_count = 0;

dns->auth\_count = 0;

dns->add\_count = 0;

}

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

int sock, sendSize, i;

struct sockaddr\_in servAddr;

struct sockaddr\_in fromAddr;

unsigned int fromSize;

int respStringLen, type, position = 0;

unsigned char buf[1024],\*reader;

char \*qname, host[100], \*rname, \*rdata, \*qname\_len, \*ip;

//struct RES\_RECORD answers[20],auth[20],addit[20]; //the replies from the DNS server

struct RES\_RECORD \*rrecord = NULL;

struct DNS\_HEADER \*dns = NULL;

struct QUESTION \*qinfo = NULL;

//initialize socket

if((sock = socket(PF\_INET, SOCK\_DGRAM, IPPROTO\_UDP)) < 0)

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

memset(&servAddr, 0, sizeof(servAddr));

servAddr.sin\_family = AF\_INET;

servAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.2");

servAddr.sin\_port = htons(53);

//Set the DNS structure to standard queries

dns = (struct DNS\_HEADER \*)&buf;

set\_header(dns);

printf("请输入想要查找的中文域名:\n");

scanf("%s", host);

position += sizeof(struct DNS\_HEADER);

qname\_len =(char\*)&buf[position];

//\*qname\_len = strlen(host);

sprintf(qname\_len, "%c", (int)strlen(host));

//printf("qname len %s\n",qname\_len);

position += 1;

qname =(char\*)&buf[position];

strcpy(qname, host);

printf("请输入查询类型: \n");

scanf("%d",&type);

position += (strlen((const char\*)qname) + 1);

qinfo = (struct QUESTION\*)&buf[position];

qinfo->qtype = htons(type); //type of the query , A , MX , CNAME , NS etc

qinfo->qclass = htons(1); //its internet

position += sizeof(struct QUESTION);

printf("Sending Packet...\n");

if((sendSize = sendto(sock,buf,position,0,(struct sockaddr\*)&servAddr,sizeof(servAddr)))<0)

{

perror("sendto failed");

}

fromSize = sizeof(fromAddr);

if((respStringLen = recvfrom(sock, buf, 1024, 0,

(struct sockaddr \*) &fromAddr, &fromSize)) < 0)

printf("recvfrom() failed.\n");

if(servAddr.sin\_addr.s\_addr != fromAddr.sin\_addr.s\_addr) {

printf("Error: received a packet from unknown source.\n");

exit(1);

}

printf("Done\n");

dns = (struct DNS\_HEADER\*) &buf;

if(ntohs(dns->add\_count)==0) {

position += sizeof(struct RES\_RECORD);

if(ntohs(qinfo->qtype)==1) {

struct in\_addr \*rdata = (struct in\_addr\*)&buf[position];

ip = inet\_ntoa(\*rdata);

if(dns->rcode!=3)

printf("The address is %s\n",ip);

else

printf("No record found!\n");

}

else if(ntohs(qinfo->qtype)==5) {//CNAME

position++;

rdata = (char \*)&buf[position];

printf("The canonical name is %s\n",rdata);

}

} else {//MX or NS

if(ntohs(qinfo->qtype)==15)

position += 2;

position += sizeof(struct RES\_RECORD);

rdata = (char \*)&buf[position];

printf("The 服务器 is %s\n",rdata);

position += (strlen(rdata)+1);

position += sizeof(struct RES\_RECORD);

struct in\_addr \*rdata = (struct in\_addr\*)&buf[position];

ip = inet\_ntoa(\*rdata);

printf("The address is %s\n",ip);

}

close(sock);

exit(0);

}

**localserver.c**

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <fcntl.h>

#include <string.h>

#define BUFFSIZE 1024

#define T\_A 1 //Ipv4 address

#define T\_NS 2 //Nameserver

#define T\_CNAME 5 // canonical name

#define T\_SOA 6 /\* start of authority zone \*/

#define T\_PTR 12 /\* domain name pointer \*/

#define T\_MX 15 //Mail server

//DNS header structure

struct DNS\_HEADER

{

unsigned short id; // identification number

unsigned char rd :1; // recursion desired

unsigned char tc :1; // truncated message

unsigned char aa :1; // authoritive answer

unsigned char opcode :4; // purpose of message

unsigned char qr :1; // query/response flag

unsigned char rcode :4; // response code

unsigned char cd :1; // checking disabled

unsigned char ad :1; // authenticated data

unsigned char z :1; // its z! reserved

unsigned char ra :1; // recursion available

unsigned short q\_count; // number of question entries

unsigned short ans\_count; // number of answer entries

unsigned short auth\_count; // number of authority entries

unsigned short add\_count; // number of resource entries

};

//Constant sized fields of query structure

struct QUESTION

{

unsigned short qtype;

unsigned short qclass;

};

//Pointers to resource record contents

struct RES\_RECORD

{

short type;

short \_class;

int ttl;

short data\_len;

};

//Structure of a Query

typedef struct

{

unsigned char \*name;

struct QUESTION \*ques;

} QUERY;

void Die(char \*mess) { perror(mess); exit(1); }

char\* convert\_type(int qtype) {

char \* q\_type = malloc(5);

switch(qtype) {

case 1: strcpy(q\_type, "A"); break;

case 2: strcpy(q\_type, "NS"); break;

case 5: strcpy(q\_type, "CNAME"); break;

case 6: strcpy(q\_type, "SOA"); break;

case 12: strcpy(q\_type, "PTR"); break;

case 15: strcpy(q\_type, "MX"); break;

}

return q\_type;

}

char\* checklocalrecord(char\* name, int type) {

FILE \*fp;

int i, j, k, temp = 0;

char line[100], content[20][100], recordname[30];

char \*p, \*qtype;

qtype = convert\_type(type);

fp = fopen("local.db" , "r");

if(fp == NULL)

perror("Error opening file");

else

{

while(fgets(line, 200, fp)!= NULL)

{

strcpy(content[temp], line);

temp++;

}

fclose(fp);

}

for(i = 0; i < temp; i++){

if((p = strstr(content[i],qtype)))

{

for(j = 0; j<= strlen(content[i]); j++)

{

if(content[i][j] == ',')

{

for(k = 0; k < j; k++)

recordname[k] = content[i][k];

recordname[j]='\0';

if(strcmp(name, recordname) == 0)

{

p =strstr(content[i], name);

return p;

}

else

break;

}

}

}

}

p = NULL;

return p;

}

int split(char \*record,char field[5][100]) {

int i, j, k, mark = -1, type = 0;

if(record != NULL)

{

for(i = 0; i <= strlen(record); i++)

{

if(record[i] == ',' || record[i] == '\n')

{

k = 0;

for(j = mark + 1; j < i; j++)

{

field[type][k] = record[j];

k++;

}

field[type][k] = '\0';

mark = i;

type++;

}

}

return 1;

}

else

return 0;

}

short check\_name\_position(char\* buf, char\* name) {

short offset = 0;

char \*start = buf;

char \*end = buf + sizeof(struct DNS\_HEADER)+1;

if(strcmp(name,end)==0) {

offset = end - start-1;

}

end = buf + 1;

return offset;

}

void set\_record(struct RES\_RECORD \*rrecord, char\* rtype, char\* rclass, char\* rttl, short length) {

int i;

for(i=0;i<16;i++) {

if(strcmp(convert\_type(i),rtype)==0)

rrecord->type = htons(i);

}

rrecord->\_class = htons(1);

rrecord->ttl = htonl(86400);

rrecord->data\_len = htons(length);

}

char\* queryrootserver(char\* data, short length, char\* ansbuf) {

int sock, received = 0;

struct sockaddr\_in echoserver;

char buf[BUFFSIZE];

char \*ip=NULL, \*qname;

struct DNS\_HEADER \*dns = NULL;

//dns = (struct DNS\_HEADER \*)data;

/\* Create the TCP socket \*/

if ((sock = socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP)) < 0) {

Die("Failed to create socket");

}

/\* Construct the server sockaddr\_in structure \*/

memset(&echoserver, 0, sizeof(echoserver)); /\* Clear struct \*/

echoserver.sin\_family = AF\_INET; /\* Internet/IP \*/

echoserver.sin\_addr.s\_addr = inet\_addr("127.0.0.3"); /\* IP address \*/

echoserver.sin\_port = htons(53); /\* server port \*/

/\* Establish connection \*/

if (connect(sock,

(struct sockaddr \*) &echoserver,

sizeof(echoserver)) < 0) {

Die("Failed to connect with server");

}

/\* Send the word to the server \*/

if (send(sock, data, length, 0) != length) {

Die("Mismatch in number of sent bytes");

}

/\* Receive the next ip back from the server \*/

//while (received < BUFFSIZE) {

int bytes = 0;

if ((bytes = recv(sock, buf, BUFFSIZE, 0)) < 0) {

printf("receive error.\n");

}

received += bytes;

buf[bytes] = '\0';

dns = (struct DNS\_HEADER \*)&buf[2];

if(ntohs(dns->add\_count)>0) {

short position = 0;

position += (3+sizeof(struct DNS\_HEADER));

qname = (char \*)&buf[position];

position += (strlen(qname)+1);

position += (sizeof(struct QUESTION)+sizeof(struct RES\_RECORD));

struct in\_addr \*rdata = (struct in\_addr\*)&buf[position];

ip = inet\_ntoa(\*rdata);

printf("Send to %s for query\n", ip);

}else {

char \*tempbuf = (char \*)buf;

memcpy(ansbuf, tempbuf, received);

}

return ip;

}

short querytldserver(char\* next\_ip, char\* data, short length, char\* ansbuf) {

int sock;

struct sockaddr\_in echoserver;

char buf[BUFFSIZE];

//char \*qname, \*rname, \*rdata;

int received = 0;

//struct DNS\_HEADER \*dns = NULL;

/\* Create the TCP socket \*/

if ((sock = socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP)) < 0) {

Die("Failed to create socket");

}

/\* Construct the server sockaddr\_in structure \*/

memset(&echoserver, 0, sizeof(echoserver)); /\* Clear struct \*/

echoserver.sin\_family = AF\_INET; /\* Internet/IP \*/

echoserver.sin\_addr.s\_addr = inet\_addr(next\_ip); /\* IP address \*/

echoserver.sin\_port = htons(53); /\* server port \*/

/\* Establish connection \*/

if (connect(sock,

(struct sockaddr \*) &echoserver,

sizeof(echoserver)) < 0) {

Die("Failed to connect with server");

}

/\* Send the word to the server \*/

if (send(sock, data, length, 0) != length) {

Die("Mismatch in number of sent bytes");

}

int bytes = 0;

if ((bytes = recv(sock, buf, BUFFSIZE-1, 0)) < 0) {

printf("receive error.\n");

//Die("Failed to receive bytes from server");

}

received += bytes;

buf[bytes] = '\0'; /\* Assure null terminated string \*/

char \*tempbuf = (char \*)buf;

tempbuf += 2;

memcpy(ansbuf, tempbuf, received);

close(sock);

return received -2;

}

void set\_header(struct DNS\_HEADER \*dns, int anscount, int authcount, int addcount) {

dns->qr = 1; //This is a response

dns->opcode = 0; //This is a standard query

dns->aa = 0; //Not Authoritative

dns->tc = 0; //This message is not truncated

dns->rd = 1; //Recursion Desired

dns->ra = 0; //Recursion not available!

dns->z = 0;

dns->ad = 0;

dns->cd = 0;

dns->rcode = 0;

dns->q\_count = htons(1); //we have only 1 question

dns->ans\_count = htons(1);

dns->auth\_count = htons(authcount);

dns->add\_count = htons(addcount);

}

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

int sock;

struct sockaddr\_in servAddr;

struct sockaddr\_in clntAddr;

unsigned int cliAddrLen;

int recvMsgSize;

char buf[BUFFSIZE];

char \*p, \*qname, \*name, \*rname, \*add, recordfield[5][100], addrecord[5][100], \*r\_data, \*next\_ip;

struct DNS\_HEADER \*dns = NULL;

struct QUESTION \*qinfo = NULL;

struct RES\_RECORD \*r\_record;

if((sock = socket(PF\_INET, SOCK\_DGRAM, 0)) < 0)

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

memset(&servAddr, 0, sizeof(servAddr));

servAddr.sin\_family = AF\_INET;

servAddr.sin\_addr.s\_addr = inet\_addr("127.0.0.2");

servAddr.sin\_port = htons(53);

if((bind(sock, (struct sockaddr \*) &servAddr, sizeof(servAddr)))<0)

printf("bind() failed.\n");

for(;;) {

int anscount = 0, authcount = 0, addcount = 0;

short position = 0;

cliAddrLen = sizeof(clntAddr);

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

printf("\*Local server listen at 127.0.0.2\n");

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

if((recvMsgSize = recvfrom(sock, buf, BUFFSIZE,

0, (struct sockaddr \*) &clntAddr, &cliAddrLen))<0)

printf("recvfrom() failed.\n");

dns = (struct DNS\_HEADER\*)&buf;

position += sizeof(struct DNS\_HEADER);

position++;

qname =(char\*)&buf[position];

printf("\*Query Name: %s\n", qname);

position += (strlen((const char\*)qname) + 1);

qinfo = (struct QUESTION\*)&buf[position];

p = checklocalrecord(qname,ntohs(qinfo->qtype));

position += sizeof(struct QUESTION);

if(split(p,recordfield)!=0) {

anscount++;

name = recordfield[0];

short temp = check\_name\_position((char \*)buf, name);

if(temp != 0) {

unsigned short \*offset = (unsigned short \*)&buf[position];

\*offset = htons(temp+49152);

position += 2;

} else {

char \*rname\_len = (char \*)&buf[position];

sprintf(rname\_len, "%c", (int)strlen(name));

position++;

rname = (char \*)&buf[position];

strcpy(rname,name);

position += (strlen((const char\*)rname) + 1);

}

//Answer section

if((strcmp(recordfield[3], "MX") == 0) || (strcmp(recordfield[3], "NS") == 0)) {

r\_record = (struct RES\_RECORD\*)&buf[position];

int r\_length = strlen(recordfield[4]) + 4;

if(strcmp(recordfield[3], "NS")==0) r\_length-=2;

set\_record(r\_record, recordfield[3],recordfield[2], recordfield[1], r\_length);

position += sizeof(struct RES\_RECORD);

if(strcmp(recordfield[3], "NS") == 0) {

position -= 2;

//NS has not preference

}

char \*r\_data\_len = (char \*)&buf[position];

sprintf(r\_data\_len, "%c", (int)strlen(recordfield[4]));

int offset2 = position;

position++;

r\_data = (char \*)&buf[position];

strcpy(r\_data,recordfield[4]);

r\_data[strlen(r\_data)] = '\0';

position += (strlen(r\_data)+1);

//additional section

add = checklocalrecord(recordfield[4], 1);

if(split(add,addrecord)!=0) {

/\*\*

short \*add\_len = (short \*)&buf[position];

int temp\_add\_len = position;

position += 2;

temp = check\_name\_position((char \*)buf, addrecord[0]);

if(temp != 0) {

\*/

unsigned short \*offset = (unsigned short \*)&buf[position];

\*offset = htons(offset2+49152);

position += 2;

r\_record = (struct RES\_RECORD\*)&buf[position];

int r\_length2 = strlen(addrecord[4]);

printf("\*My Answer: %s\n", addrecord[4]);

set\_record(r\_record, addrecord[3],addrecord[2], addrecord[1], r\_length2);

position += sizeof(struct RES\_RECORD);

position -= 2;

struct in\_addr \*rdata = NULL;

rdata = (struct in\_addr\*)&buf[position];

inet\_aton(addrecord[4],rdata);

position += sizeof(struct in\_addr);

addcount++;

}

}

else {

r\_record = (struct RES\_RECORD\*)&buf[position];

int r\_length = sizeof(struct in\_addr);

set\_record(r\_record, recordfield[3],recordfield[2], recordfield[1], r\_length);

position += sizeof(struct RES\_RECORD);

position -= 2;

struct in\_addr \*rdata = NULL;

rdata = (struct in\_addr\*)&buf[position];

inet\_aton(recordfield[4],rdata);

position += sizeof(struct in\_addr);

printf("\*My Answer: %s\n",recordfield[4]);

}

set\_header(dns, anscount, authcount, addcount);

if((sendto(sock, buf, position, 0,

(struct sockaddr \*) &clntAddr, sizeof(clntAddr))) < 0)

printf("sendto() error.\n");

}

else {

printf("Send to 127.0.0.3 for query\n");

//switch\_to\_tcp

char tcpbuf[BUFFSIZE];

short \*buf\_len = malloc(2);

\*buf\_len = htons(position);

memcpy(&tcpbuf[0], buf\_len, 2);

char \*tempbuf = (char \*)&tcpbuf[2];

memcpy(tempbuf, buf, position);

char ansbuf1[BUFFSIZE];

next\_ip = queryrootserver(tcpbuf, position+2, ansbuf1);

if(next\_ip!=NULL) {

char ansbuf2[BUFFSIZE];

short ans\_len = querytldserver(next\_ip,tcpbuf, position+2, ansbuf2);

if((sendto(sock, ansbuf2, ans\_len, 0,

(struct sockaddr \*) &clntAddr, sizeof(clntAddr))) < 0)

printf("sendto() error.\n");

} else {

short \*len = (short \*)&ansbuf1;

char finalbuf[BUFFSIZE];

char \*tempbuf = (char \*)ansbuf1;

tempbuf += 2;

memcpy(finalbuf, tempbuf, \*len-2);

if((sendto(sock, finalbuf, \*len-2, 0,

(struct sockaddr \*) &clntAddr, sizeof(clntAddr))) < 0)

printf("sendto() error.\n");

}

}

}

}

**rootserver.c**

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <netinet/in.h>

#define MAXPENDING 5 /\* Max connection requests \*/

#define BUFFSIZE 1024

#define T\_A 1 //Ipv4 address

#define T\_NS 2 //Nameserver

#define T\_CNAME 5 // canonical name

#define T\_SOA 6 /\* start of authority zone \*/

#define T\_PTR 12 /\* domain name pointer \*/

#define T\_MX 15 //Mail server

//DNS header structure

struct DNS\_HEADER

{

unsigned short id; // identification number

unsigned char rd :1; // recursion desired

unsigned char tc :1; // truncated message

unsigned char aa :1; // authoritive answer

unsigned char opcode :4; // purpose of message

unsigned char qr :1; // query/response flag

unsigned char rcode :4; // response code

unsigned char cd :1; // checking disabled

unsigned char ad :1; // authenticated data

unsigned char z :1; // its z! reserved

unsigned char ra :1; // recursion available

unsigned short q\_count; // number of question entries

unsigned short ans\_count; // number of answer entries

unsigned short auth\_count; // number of authority entries

unsigned short add\_count; // number of resource entries

};

//Constant sized fields of query structure

struct QUESTION

{

unsigned short qtype;

unsigned short qclass;

};

//Pointers to resource record contents

struct RES\_RECORD

{

unsigned short type;

unsigned short \_class;

unsigned int ttl;

unsigned short data\_len;

};

//Structure of a Query

typedef struct

{

unsigned char \*name;

struct QUESTION \*ques;

} QUERY;

void Die(char \*mess) { perror(mess); exit(1); }

char\* convert\_type(int qtype) {

char \* q\_type = malloc(5);

switch(qtype) {

case 1: strcpy(q\_type, "A"); break;

case 2: strcpy(q\_type, "NS"); break;

case 5: strcpy(q\_type, "CNAME"); break;

case 6: strcpy(q\_type, "SOA"); break;

case 12: strcpy(q\_type, "PTR"); break;

case 15: strcpy(q\_type, "MX"); break;

}

return q\_type;

}

void set\_record(struct RES\_RECORD \*rrecord, char\* rtype, char\* rclass, char\* rttl, short length) {

int i;

for(i=0;i<16;i++) {

if(strcmp(convert\_type(i),rtype)==0)

rrecord->type = htons(i);

}

rrecord->\_class = htons(1);

rrecord->ttl = htonl(86400);

rrecord->data\_len = htons(length);

}

short check\_name\_position(char\* buf, char\* name) {

short offset = 0;

char \*start = buf;

char \*end = buf + sizeof(struct DNS\_HEADER)+3;

if(strcmp(name,end)==0) {

offset = end - start-3;

//printf("offset: %d\n",offset);

/\*\*

char \*temp = malloc(strlen(hexoffset) + strlen("C0")+1);

strcpy(temp, "c0");

strcat(temp,hexoffset);\*/

}

end = buf + 1;

return offset;

}

char\* check\_root\_record(char\* name) {

FILE \*fp;

char \*qname=malloc(strlen(name)+1);

strcpy(qname, name);

int i, temp = 0;

char line[100], content[20][100], \*record, \*ip=NULL;

char \*head = qname;

char \*end = strstr(qname,"的");

if(end!=NULL) {

qname[end-head] = '\0';

fp = fopen("root.db" , "r");

if(fp == NULL)

perror("Error opening file");

else {

while(fgets(line, 200, fp)!= NULL)

{

strcpy(content[temp], line);

temp++;

}

fclose(fp);

for(i = 0; i < temp; i++) {

if((record = strstr(content[i], qname)))

break;

}

}

ip = strstr(record,"1");

ip[strlen(ip)-2] = '\0';

}

//printf("the ip is %s\n",ip);

return ip;

}

void set\_header(struct DNS\_HEADER \*dns, int anscount, int authcount, int addcount) {

dns->qr = 1; //This is a response

dns->opcode = 0; //This is a standard query

dns->aa = 0; //Not Authoritative

dns->tc = 0; //This message is not truncated

dns->rd = 1; //Recursion Desired

dns->ra = 0; //Recursion not available!

dns->z = 0;

dns->ad = 0;

dns->cd = 0;

dns->rcode = 0;

dns->q\_count = htons(1); //we have only 1 question

dns->ans\_count = htons(anscount);

dns->auth\_count = htons(authcount);

dns->add\_count = htons(addcount);

}

void set\_error\_header(struct DNS\_HEADER \*dns) {

dns->qr = 1; //This is a response

dns->opcode = 0; //This is a standard query

dns->aa = 0; //Not Authoritative

dns->tc = 0; //This message is not truncated

dns->rd = 1; //Recursion Desired

dns->ra = 0; //Recursion not available!

dns->z = 0;

dns->ad = 0;

dns->cd = 0;

dns->rcode = 3;

dns->q\_count = htons(1); //we have only 1 question

dns->ans\_count = htons(0);

dns->auth\_count = htons(0);

dns->add\_count = htons(0);

}

void HandleClient(int sock) {

char buf[BUFFSIZE];

int received = -1;

char \*qname, \*rtype, \*rclass, \*ttl;

short position = 2;

struct DNS\_HEADER \*dns = NULL;

struct RES\_RECORD \*r\_record;

int anscount = 0, authcount = 0, addcount = 0;

/\* Receive message \*/

if ((received = recv(sock, buf, BUFFSIZE, 0)) < 0) {

Die("Failed to receive initial bytes from client");

}

short \*buf\_len = malloc(2);

qname =(char\*)&buf[sizeof(struct DNS\_HEADER)+3];

char \*ip = check\_root\_record(qname);

printf("\*Query Name: %s\n",qname);

dns = (struct DNS\_HEADER\*)&buf[position];

position += sizeof(struct DNS\_HEADER);

position++;

position += (strlen(qname) + 1);

position += sizeof(struct QUESTION);

if(ip!=NULL) {

printf("\*My Answer: %s\n",check\_root\_record(qname));

short temp = check\_name\_position((char \*)buf, qname);

unsigned short \*offset = (unsigned short \*)&buf[position];

\*offset = htons(temp+49152);

position += 2;

r\_record = (struct RES\_RECORD\*)&buf[position];

int r\_length = sizeof(struct in\_addr);

rtype = "A";

rclass = "IN";

ttl = "86400";

set\_record(r\_record, rtype, rclass, ttl, r\_length);

position += sizeof(struct RES\_RECORD);

position -= 2;

struct in\_addr \*rdata = NULL;

rdata = (struct in\_addr\*)&buf[position];

inet\_aton(ip,rdata);

position += sizeof(struct in\_addr);

addcount++;

\*buf\_len = htons(position-2);

memcpy(&buf[0], buf\_len, 2);

set\_header(dns, anscount, authcount, addcount);

} else {

printf("Server not found!\n");

set\_error\_header(dns);

\*buf\_len = htons(position-2);

memcpy(&buf[0], buf\_len, 2);

}

if (send(sock, buf, position, 0) < position) {

Die("Failed to send bytes to client");

}

close(sock);

}

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

int serversock, clientsock;

struct sockaddr\_in echoserver, echoclient;

/\* Create the TCP socket \*/

if ((serversock = socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP)) < 0) {

Die("Failed to create socket");

}

/\* Construct the server sockaddr\_in structure \*/

memset(&echoserver, 0, sizeof(echoserver)); /\* Clear struct \*/

echoserver.sin\_family = AF\_INET; /\* Internet/IP \*/

echoserver.sin\_addr.s\_addr = inet\_addr("127.0.0.3"); /\* Incoming addr \*/

echoserver.sin\_port = htons(53); /\* server port \*/

/\* Bind the server socket \*/

if (bind(serversock, (struct sockaddr \*) &echoserver,

sizeof(echoserver)) < 0) {

Die("Failed to bind the server socket");

}

/\* Listen on the server socket \*/

if (listen(serversock, MAXPENDING) < 0) {

Die("Failed to listen on server socket");

}

/\* Run until cancelled \*/

while (1) {

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

printf("\*Root server listen at 127.0.0.3\n");

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

unsigned int clientlen = sizeof(echoclient);

/\* Wait for client connection \*/

if ((clientsock =

accept(serversock, (struct sockaddr \*) &echoclient,

&clientlen)) < 0) {

Die("Failed to accept client connection");

}

//fprintf(stdout, "Client connected: %s\n",

// inet\_ntoa(echoclient.sin\_addr));

HandleClient(clientsock);

}

}

**tldserver1.c**

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <netinet/in.h>

#define MAXPENDING 5 /\* Max connection requests \*/

#define BUFFSIZE 1024

#define T\_A 1 //Ipv4 address

#define T\_NS 2 //Nameserver

#define T\_CNAME 5 // canonical name

#define T\_SOA 6 /\* start of authority zone \*/

#define T\_PTR 12 /\* domain name pointer \*/

#define T\_MX 15 //Mail server

//DNS header structure

struct DNS\_HEADER

{

unsigned short id; // identification number

unsigned char rd :1; // recursion desired

unsigned char tc :1; // truncated message

unsigned char aa :1; // authoritive answer

unsigned char opcode :4; // purpose of message

unsigned char qr :1; // query/response flag

unsigned char rcode :4; // response code

unsigned char cd :1; // checking disabled

unsigned char ad :1; // authenticated data

unsigned char z :1; // its z! reserved

unsigned char ra :1; // recursion available

unsigned short q\_count; // number of question entries

unsigned short ans\_count; // number of answer entries

unsigned short auth\_count; // number of authority entries

unsigned short add\_count; // number of resource entries

};

//Constant sized fields of query structure

struct QUESTION

{

unsigned short qtype;

unsigned short qclass;

};

//Pointers to resource record contents

struct RES\_RECORD

{

short type;

short \_class;

int ttl;

short data\_len;

};

//Structure of a Query

typedef struct

{

unsigned char \*name;

struct QUESTION \*ques;

} QUERY;

void Die(char \*mess) { perror(mess); exit(1); }

char\* convert\_type(int qtype) {

char \* q\_type = malloc(5);

switch(qtype) {

case 1: strcpy(q\_type, "A"); break;

case 2: strcpy(q\_type, "NS"); break;

case 5: strcpy(q\_type, "CNAME"); break;

case 6: strcpy(q\_type, "SOA"); break;

case 12: strcpy(q\_type, "PTR"); break;

case 15: strcpy(q\_type, "MX"); break;

}

return q\_type;

}

char\* checktld1record(char\* name, int type) {

FILE \*fp;

int i, j, k, temp = 0;

char line[100], content[20][100], recordname[30];

char \*p, \*qtype;

qtype = convert\_type(type);

fp = fopen("tld1.db" , "r");

if(fp == NULL)

perror("Error opening file");

else

{

while(fgets(line, 200, fp)!= NULL)

{

strcpy(content[temp], line);

temp++;

}

fclose(fp);

}

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

{

if((p = strstr(content[i],qtype)))

{

for(j = 0; j<= strlen(content[i]); j++)

{

if(content[i][j] == ',')

{

for(k = 0; k < j; k++)

recordname[k] = content[i][k];

recordname[j]='\0';

if(strcmp(name, recordname) == 0)

{

p =strstr(content[i], name);

return p;

}

else

break;

}

}

}

}

p = NULL;

return p;

}

int split(char \*record,char field[5][100]) {

int i, j, k, mark = -1, type = 0;

if(record != NULL)

{

for(i = 0; i <= strlen(record); i++)

{

if(record[i] == ',' || record[i] == '\n')

{

k = 0;

for(j = mark + 1; j < i; j++)

{

field[type][k] = record[j];

k++;

}

field[type][k] = '\0';

mark = i;

type++;

}

}

return 1;

}

else

return 0;

}

short check\_name\_position(char\* buf, char\* name) {

short offset = 0;

char \*start = buf;

char \*end = buf + sizeof(struct DNS\_HEADER)+3;

if(strcmp(name,end)==0) {

offset = end - start-3;

}

end = buf + 1;

return offset;

}

void set\_record(struct RES\_RECORD \*rrecord, char\* rtype, char\* rclass, char\* rttl, short length) {

int i;

for(i=0;i<16;i++) {

if(strcmp(convert\_type(i),rtype)==0)

rrecord->type = htons(i);

}

rrecord->\_class = htons(1);

rrecord->ttl = htonl(86400);

rrecord->data\_len = htons(length);

}

void set\_header(struct DNS\_HEADER \*dns, int anscount, int authcount, int addcount) {

dns->qr = 1; //This is a response

dns->opcode = 0; //This is a standard query

dns->aa = 0; //Not Authoritative

dns->tc = 0; //This message is not truncated

dns->rd = 1; //Recursion Desired

dns->ra = 0; //Recursion not available!

dns->z = 0;

dns->ad = 0;

dns->cd = 0;

dns->rcode = 0;

dns->q\_count = htons(1); //we have only 1 question

dns->ans\_count = htons(anscount);

dns->auth\_count = htons(authcount);

dns->add\_count = htons(addcount);

}

void set\_error\_header(struct DNS\_HEADER \*dns) {

dns->qr = 1; //This is a response

dns->opcode = 0; //This is a standard query

dns->aa = 0; //Not Authoritative

dns->tc = 0; //This message is not truncated

dns->rd = 1; //Recursion Desired

dns->ra = 0; //Recursion not available!

dns->z = 0;

dns->ad = 0;

dns->cd = 0;

dns->rcode = 3;

dns->q\_count = htons(1); //we have only 1 question

dns->ans\_count = htons(0);

dns->auth\_count = htons(0);

dns->add\_count = htons(0);

}

void HandleClient(int sock) {

char buf[BUFFSIZE];

int received = -1, anscount = 0, authcount = 0, addcount = 0;

short position = 2;

char \*p, \*qname, \*name, \*rname, \*add, recordfield[5][100], addrecord[5][100], \*r\_data;

struct DNS\_HEADER \*dns = NULL;

struct QUESTION \*qinfo = NULL;

struct RES\_RECORD \*r\_record;

/\* Receive message \*/

if ((received = recv(sock, buf, BUFFSIZE, 0)) < 0) {

Die("Failed to receive initial bytes from client");

}

dns = (struct DNS\_HEADER\*)&buf[position];

position += sizeof(struct DNS\_HEADER);

position++;

qname =(char\*)&buf[position];

printf("\*Query Name: %s\n", qname);

position += (strlen((const char\*)qname) + 1);

qinfo = (struct QUESTION\*)&buf[position];

p = checktld1record(qname,ntohs(qinfo->qtype));

position += sizeof(struct QUESTION);

if(split(p,recordfield)!=0) {

anscount++;

name = recordfield[0];

short temp = check\_name\_position((char \*)buf, name);

if(temp != 0) {

unsigned short \*offset = (unsigned short \*)&buf[position];

\*offset = htons(temp+49152);

position += 2;

} else {

char \*rname\_len = (char \*)&buf[position];

sprintf(rname\_len, "%c", (int)strlen(name));

position++;

rname = (char \*)&buf[position];

strcpy(rname,name);

position += (strlen((const char\*)rname) + 1);

}

if((strcmp(recordfield[3], "MX") == 0) || (strcmp(recordfield[3], "NS") == 0)) {

r\_record = (struct RES\_RECORD\*)&buf[position];

int r\_length = strlen(recordfield[4]) + 4;

if(strcmp(recordfield[3], "NS")==0) r\_length-=2;

set\_record(r\_record, recordfield[3],recordfield[2], recordfield[1], r\_length);

position += sizeof(struct RES\_RECORD);

if(strcmp(recordfield[3], "NS") == 0) {

position -= 2;

//NS has not preference

}

char \*r\_data\_len = (char \*)&buf[position];

sprintf(r\_data\_len, "%c", (int)strlen(recordfield[4]));

int offset2 = position;

position++;

r\_data = (char \*)&buf[position];

strcpy(r\_data,recordfield[4]);

r\_data[strlen(r\_data)] = '\0';

position += (strlen(r\_data)+1);

add = checktld1record(recordfield[4], 1);

if(split(add,addrecord)!=0) {

/\*\*

short \*add\_len = (short \*)&buf[position];

int temp\_add\_len = position;

position += 2;

temp = check\_name\_position((char \*)buf, addrecord[0]);

if(temp != 0) {

\*/

unsigned short \*offset = (unsigned short \*)&buf[position];

\*offset = htons(offset2+49150);

//if(strcmp(recordfield[3], "MX") == 0)

position += 2;//preference

r\_record = (struct RES\_RECORD\*)&buf[position];

int r\_length2 = strlen(addrecord[4]);

printf("\*My Answer: %s\n", addrecord[4]);

set\_record(r\_record, addrecord[3],addrecord[2], addrecord[1], r\_length2);

position += sizeof(struct RES\_RECORD);

position -= 2;

struct in\_addr \*rdata = NULL;

rdata = (struct in\_addr\*)&buf[position];

inet\_aton(addrecord[4],rdata);

position += sizeof(struct in\_addr);

addcount++;

}

}

else if(strcmp(recordfield[3], "CNAME") == 0){

r\_record = (struct RES\_RECORD\*)&buf[position];

int r\_length = strlen(recordfield[4])+2;

position -=2;

set\_record(r\_record, recordfield[3],recordfield[2], recordfield[1], r\_length);

position += sizeof(struct RES\_RECORD);

char \*r\_data\_len = (char \*)&buf[position];

sprintf(r\_data\_len, "%c", (int)strlen(recordfield[4]));

position++;

r\_data = (char \*)&buf[position];

strcpy(r\_data,recordfield[4]);

r\_data[strlen(r\_data)] = '\0';

position += (strlen(r\_data)+1);

printf("\*My Answer: %s\n",recordfield[4]);

}

else {

r\_record = (struct RES\_RECORD\*)&buf[position];

int r\_length = sizeof(struct in\_addr);

set\_record(r\_record, recordfield[3],recordfield[2], recordfield[1], r\_length);

position += sizeof(struct RES\_RECORD);

position -= 2;

struct in\_addr \*rdata = NULL;

rdata = (struct in\_addr\*)&buf[position];

inet\_aton(recordfield[4],rdata);

position += sizeof(struct in\_addr);

printf("\*My Answer: %s\n",recordfield[4]);

}

set\_header(dns, anscount, authcount, addcount);

}else {

//operation when no match found

printf("No record found!\n");

set\_error\_header(dns);

}

short \*buf\_len = malloc(2);

\*buf\_len = htons(position-2);

memcpy(&buf[0], buf\_len, 2);

/\* Send back received data \*/

if (send(sock, buf, position, 0) <0) {

Die("Failed to send bytes to client");

}

close(sock);

}

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

int serversock, clientsock;

struct sockaddr\_in echoserver, echoclient;

/\* Create the TCP socket \*/

if ((serversock = socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP)) < 0) {

Die("Failed to create socket");

}

/\* Construct the server sockaddr\_in structure \*/

memset(&echoserver, 0, sizeof(echoserver)); /\* Clear struct \*/

echoserver.sin\_family = AF\_INET; /\* Internet/IP \*/

echoserver.sin\_addr.s\_addr = inet\_addr("127.0.0.4"); /\* Incoming addr \*/

echoserver.sin\_port = htons(53); /\* server port \*/

/\* Bind the server socket \*/

if (bind(serversock, (struct sockaddr \*) &echoserver,

sizeof(echoserver)) < 0) {

Die("Failed to bind the server socket");

}

/\* Listen on the server socket \*/

if (listen(serversock, MAXPENDING) < 0) {

Die("Failed to listen on server socket");

}

/\* Run until cancelled \*/

while (1) {

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

printf("\*TLD server 1 listen at 127.0.0.4\n");

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

unsigned int clientlen = sizeof(echoclient);

/\* Wait for client connection \*/

if ((clientsock =

accept(serversock, (struct sockaddr \*) &echoclient,

&clientlen)) < 0) {

Die("Failed to accept client connection");

}

HandleClient(clientsock);

}

}

**tldserver2.c**

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <netinet/in.h>

#define MAXPENDING 5 /\* Max connection requests \*/

#define BUFFSIZE 1024

#define T\_A 1 //Ipv4 address

#define T\_NS 2 //Nameserver

#define T\_CNAME 5 // canonical name

#define T\_SOA 6 /\* start of authority zone \*/

#define T\_PTR 12 /\* domain name pointer \*/

#define T\_MX 15 //Mail server

//DNS header structure

struct DNS\_HEADER

{

unsigned short id; // identification number

unsigned char rd :1; // recursion desired

unsigned char tc :1; // truncated message

unsigned char aa :1; // authoritive answer

unsigned char opcode :4; // purpose of message

unsigned char qr :1; // query/response flag

unsigned char rcode :4; // response code

unsigned char cd :1; // checking disabled

unsigned char ad :1; // authenticated data

unsigned char z :1; // its z! reserved

unsigned char ra :1; // recursion available

unsigned short q\_count; // number of question entries

unsigned short ans\_count; // number of answer entries

unsigned short auth\_count; // number of authority entries

unsigned short add\_count; // number of resource entries

};

//Constant sized fields of query structure

struct QUESTION

{

unsigned short qtype;

unsigned short qclass;

};

//Pointers to resource record contents

struct RES\_RECORD

{

short type;

short \_class;

int ttl;

short data\_len;

};

//Structure of a Query

typedef struct

{

unsigned char \*name;

struct QUESTION \*ques;

} QUERY;

void Die(char \*mess) { perror(mess); exit(1); }

char\* convert\_type(int qtype) {

char \* q\_type = malloc(5);

switch(qtype) {

case 1: strcpy(q\_type, "A"); break;

case 2: strcpy(q\_type, "NS"); break;

case 5: strcpy(q\_type, "CNAME"); break;

case 6: strcpy(q\_type, "SOA"); break;

case 12: strcpy(q\_type, "PTR"); break;

case 15: strcpy(q\_type, "MX"); break;

}

return q\_type;

}

char\* checktld1record(char\* name, int type) {

FILE \*fp;

int i, j, k, temp = 0;

char line[100], content[20][100], recordname[30];

char \*p, \*qtype;

qtype = convert\_type(type);

fp = fopen("tld2.db" , "r");

if(fp == NULL)

perror("Error opening file");

else

{

while(fgets(line, 200, fp)!= NULL)

{

strcpy(content[temp], line);

temp++;

}

fclose(fp);

}

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

{

if((p = strstr(content[i],qtype)))

{

for(j = 0; j<= strlen(content[i]); j++)

{

if(content[i][j] == ',')

{

for(k = 0; k < j; k++)

recordname[k] = content[i][k];

recordname[j]='\0';

if(strcmp(name, recordname) == 0)

{

p =strstr(content[i], name);

return p;

}

else

break;

}

}

}

}

p = NULL;

return p;

}

int split(char \*record,char field[5][100]) {

int i, j, k, mark = -1, type = 0;

if(record != NULL)

{

for(i = 0; i <= strlen(record); i++)

{

if(record[i] == ',' || record[i] == '\n')

{

k = 0;

for(j = mark + 1; j < i; j++)

{

field[type][k] = record[j];

k++;

}

field[type][k] = '\0';

mark = i;

type++;

}

}

return 1;

}

else

return 0;

}

short check\_name\_position(char\* buf, char\* name) {

short offset = 0;

char \*start = buf;

char \*end = buf + sizeof(struct DNS\_HEADER)+3;

if(strcmp(name,end)==0) {

offset = end - start-3;

}

end = buf + 1;

return offset;

}

void set\_record(struct RES\_RECORD \*rrecord, char\* rtype, char\* rclass, char\* rttl, short length) {

int i;

for(i=0;i<16;i++) {

if(strcmp(convert\_type(i),rtype)==0)

rrecord->type = htons(i);

}

rrecord->\_class = htons(1);

rrecord->ttl = htonl(86400);

rrecord->data\_len = htons(length);

}

void set\_header(struct DNS\_HEADER \*dns, int anscount, int authcount, int addcount) {

dns->qr = 1; //This is a response

dns->opcode = 0; //This is a standard query

dns->aa = 0; //Not Authoritative

dns->tc = 0; //This message is not truncated

dns->rd = 1; //Recursion Desired

dns->ra = 0; //Recursion not available!

dns->z = 0;

dns->ad = 0;

dns->cd = 0;

dns->rcode = 0;

dns->q\_count = htons(1); //we have only 1 question

dns->ans\_count = htons(anscount);

dns->auth\_count = htons(authcount);

dns->add\_count = htons(addcount);

}

void set\_error\_header(struct DNS\_HEADER \*dns) {

dns->qr = 1; //This is a response

dns->opcode = 0; //This is a standard query

dns->aa = 0; //Not Authoritative

dns->tc = 0; //This message is not truncated

dns->rd = 1; //Recursion Desired

dns->ra = 0; //Recursion not available!

dns->z = 0;

dns->ad = 0;

dns->cd = 0;

dns->rcode = 3;

dns->q\_count = htons(1); //we have only 1 question

dns->ans\_count = htons(0);

dns->auth\_count = htons(0);

dns->add\_count = htons(0);

}

void HandleClient(int sock) {

char buf[BUFFSIZE];

int received = -1, anscount = 0, authcount = 0, addcount = 0;

short position = 2;

char \*p, \*qname, \*name, \*rname, \*add, recordfield[5][100], addrecord[5][100], \*r\_data;

struct DNS\_HEADER \*dns = NULL;

struct QUESTION \*qinfo = NULL;

struct RES\_RECORD \*r\_record;

/\* Receive message \*/

if ((received = recv(sock, buf, BUFFSIZE, 0)) < 0) {

Die("Failed to receive initial bytes from client");

}

dns = (struct DNS\_HEADER\*)&buf[position];

position += sizeof(struct DNS\_HEADER);

position++;

qname =(char\*)&buf[position];

printf("\*Query Name: %s\n", qname);

position += (strlen((const char\*)qname) + 1);

qinfo = (struct QUESTION\*)&buf[position];

p = checktld1record(qname,ntohs(qinfo->qtype));

position += sizeof(struct QUESTION);

if(split(p,recordfield)!=0) {

anscount++;

name = recordfield[0];

short temp = check\_name\_position((char \*)buf, name);

if(temp != 0) {

unsigned short \*offset = (unsigned short \*)&buf[position];

\*offset = htons(temp+49152);

position += 2;

} else {

char \*rname\_len = (char \*)&buf[position];

sprintf(rname\_len, "%c", (int)strlen(name));

position++;

rname = (char \*)&buf[position];

strcpy(rname,name);

position += (strlen((const char\*)rname) + 1);

}

if((strcmp(recordfield[3], "MX") == 0) || (strcmp(recordfield[3], "NS") == 0)) {

r\_record = (struct RES\_RECORD\*)&buf[position];

int r\_length = strlen(recordfield[4]) + 4;

if(strcmp(recordfield[3], "NS")==0) r\_length-=2;

set\_record(r\_record, recordfield[3],recordfield[2], recordfield[1], r\_length);

position += sizeof(struct RES\_RECORD);

if(strcmp(recordfield[3], "NS") == 0) {

position -= 2;

//NS has not preference

}

char \*r\_data\_len = (char \*)&buf[position];

sprintf(r\_data\_len, "%c", (int)strlen(recordfield[4]));

int offset2 = position;

position++;

r\_data = (char \*)&buf[position];

strcpy(r\_data,recordfield[4]);

r\_data[strlen(r\_data)] = '\0';

position += (strlen(r\_data)+1);

add = checktld1record(recordfield[4], 1);

if(split(add,addrecord)!=0) {

/\*\*

short \*add\_len = (short \*)&buf[position];

int temp\_add\_len = position;

position += 2;

temp = check\_name\_position((char \*)buf, addrecord[0]);

if(temp != 0) {

\*/

unsigned short \*offset = (unsigned short \*)&buf[position];

\*offset = htons(offset2+49150);

position += 2;

r\_record = (struct RES\_RECORD\*)&buf[position];

int r\_length2 = strlen(addrecord[4]);

printf("\*My Answer: %s\n", addrecord[4]);

set\_record(r\_record, addrecord[3],addrecord[2], addrecord[1], r\_length2);

position += sizeof(struct RES\_RECORD);

position -= 2;

struct in\_addr \*rdata = NULL;

rdata = (struct in\_addr\*)&buf[position];

inet\_aton(addrecord[4],rdata);

position += sizeof(struct in\_addr);

addcount++;

}

}

else if(strcmp(recordfield[3], "CNAME") == 0){

r\_record = (struct RES\_RECORD\*)&buf[position];

int r\_length = strlen(recordfield[4])+2;

position -=2;//cuowei

set\_record(r\_record, recordfield[3],recordfield[2], recordfield[1], r\_length);

position += sizeof(struct RES\_RECORD);

char \*r\_data\_len = (char \*)&buf[position];

sprintf(r\_data\_len, "%c", (int)strlen(recordfield[4]));

position++;

r\_data = (char \*)&buf[position];

strcpy(r\_data,recordfield[4]);

r\_data[strlen(r\_data)] = '\0';

position += (strlen(r\_data)+1);

printf("\*My Answer: %s\n",recordfield[4]);

}

else {

r\_record = (struct RES\_RECORD\*)&buf[position];

int r\_length = sizeof(struct in\_addr);

set\_record(r\_record, recordfield[3],recordfield[2], recordfield[1], r\_length);

position += sizeof(struct RES\_RECORD);

position -= 2;

struct in\_addr \*rdata = NULL;

rdata = (struct in\_addr\*)&buf[position];

inet\_aton(recordfield[4],rdata);

position += sizeof(struct in\_addr);

printf("\*My Answer: %s\n",recordfield[4]);

}

set\_header(dns, anscount, authcount, addcount);

}else {

//operation when no match found

printf("No record found!\n");

set\_error\_header(dns);

}

short \*buf\_len = malloc(2);

\*buf\_len = htons(position-2);

memcpy(&buf[0], buf\_len, 2);

/\* Send back received data \*/

if (send(sock, buf, position, 0) <0) {

Die("Failed to send bytes to client");

}

close(sock);

}

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

int serversock, clientsock;

struct sockaddr\_in echoserver, echoclient;

/\* Create the TCP socket \*/

if ((serversock = socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP)) < 0) {

Die("Failed to create socket");

}

/\* Construct the server sockaddr\_in structure \*/

memset(&echoserver, 0, sizeof(echoserver)); /\* Clear struct \*/

echoserver.sin\_family = AF\_INET; /\* Internet/IP \*/

echoserver.sin\_addr.s\_addr = inet\_addr("127.0.0.5"); /\* Incoming addr \*/

echoserver.sin\_port = htons(53); /\* server port \*/

/\* Bind the server socket \*/

if (bind(serversock, (struct sockaddr \*) &echoserver,

sizeof(echoserver)) < 0) {

Die("Failed to bind the server socket");

}

/\* Listen on the server socket \*/

if (listen(serversock, MAXPENDING) < 0) {

Die("Failed to listen on server socket");

}

/\* Run until cancelled \*/

while (1) {

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

printf("\*TLD server 2 listen at 127.0.0.5\n");

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

unsigned int clientlen = sizeof(echoclient);

/\* Wait for client connection \*/

if ((clientsock =

accept(serversock, (struct sockaddr \*) &echoclient,

&clientlen)) < 0) {

Die("Failed to accept client connection");

}

HandleClient(clientsock);

}

}