Project report

Design and implementation of DNS Client and Server

Course Title: *Internet Application*

Name: *WenhaoZhang (2015213067)*

HeZhu (2015213059)

Date: 2018.06.20

CONTENT

1. Overview

1.1 The target of the project is about:

i. Deeply understand the knowledge of DNS(Domain Name System).
ii. Based on all the practices, design and complete a DNS client and server system which is based on Linux command line.
iii. Use the system to achieve Chinese domain name resolution.

1.2 The requirements of the project is about:

Major requirements:

i. Achieve Chinese domain name resolution. For example, 主页. 北邮. 教育. 中国(correspond to www.bupt.edu.cn in database)
ii. At least 4 top-level domains are supported. At least three levels domain names can be resolved.
iii. Supported resource record types: A, MX, CNAME.
iv. Supported resolution method: iteration resolution.
v. Cache file is available, and the trace record of query has to be printed out.
vi. The transport layer protocol between client and server is TCP.The transport layer protocol between server and server is UDP.
vii. The application layer protocol is DNS.

Optional requirements:

i. Support the resource record of PTR type.
ii. Support recursive resolution.
iii. Support several query questions in one DNS message.

2. Requirements Analysis

2.1Development Environment.

The environment of this coursework is based on Ubuntu operation system and using C language with gcc compiler and gdb debugger.

2.2Functional requirements in details

i. There are 3 levels of 6 DNS servers: The server which communicates directly with the client is the local server. The root DNS server is root server. The top level and second level DNS servers are nation server, education server, other server and government server.

ii. When query MX type of resource record, the additional section in DNS packets will carry a corresponding IP address.

iii. The local trace record will have the query path and responding time of the servers.

iv. All the DNS packets should be resolved normally and correctly by wireshark.

v. Use txt files as database to store all domain names and addresses.
An example of database record:
主页.北邮.教育.中国，86400，IN，A，192.168.1.25
北邮.教育.中国，86400，IN，MX，邮件服务器.北邮.教育.中国
邮件服务器.北邮.教育.中国，86400，IN ，A，192.168.1.37

vi. We choose one optional requirements to achieve. The system should support several query questions in one DNS message.

3. Preliminary Design

3.1 Decomposition of functional modules and relationship between modules

We design the DNS client in three main parts, including processing user's command, sending corresponding query and process the received response.

For the design of DNS server, it includes 3 main parts, including receiving the query from DNS client or other server, analyzing the query and check cache or do iteration and recursion query and receiving the response and analyzing it, refresh or add the result into the cache. Finally send the response back to the DNS client.

DNS Client:

The specific process is as follow:

1. Process the input of the user.

The program will first process use ' s input. By typing in the parameters of the domain name and the type and input whether use iteration or recursion to query the server.The client will extract the query type and the domain name. The user can query A, CNAME, and MX type recourse records to the DNS server. The user can also input two domain names and split them using a space to achieve multiple queries. After the user input all the domain name that he or she want to query, the client will

begin to construct the TCP packet according to the input parameters. If the user use -I instead of -R, the RD field in the Flag part will be 0 to indicate that the query will use iteration.

2. Construct the packet according to the user's input and send to the local server.

The transaction id will be a random number generate by the client to indicate the DNS query packet. According to the numbers of the user's wanted query domain name, the questions count will be added to indicate the number of the queries. And then follows the Answer RRs, Authority RRs and Additional RRs. All these three parts are 0, because that those parts will be filled by the DNS server if there are any RRs. That's the end of the header part.

Then according to the number of the user's input domain name, the client start to generate the Queries part. In this part, all the query will be separate with each other. Since the domain name is in Chinese, and the encoding of Chinese used in Ubuntu is UFT-8, so each Chinese character's length is equals to three chars. By splitting the domain by dot and then calculate the sub domain's length, the sub domain's length will be get which will be in the start of the domain to indicate the start and the length of the sub domain instead of the dot in the human reading language. After finish the name part of the single query, the query type will be added following to the name, and the query will follows with the class type.

After adding all the queries to the send buffer, the client will send the packet to the local DNS server by TCP protocol using port 53, which is the standard port of DNS.

3. Process the received data from the local server

After send out the query packet, the client start to listen to the connection and waiting for receiving the response of the server. After received the packet from the server, the client will first analyze the flag part, so that it can know whether there are answers or the input domain is correct or not. If the Rcode is 0x03 which means no domain, and if the Rcode is 0x00 which means correct and there are answers. If the authority bit is set to 1, that mean the answer is resolved by the authority server not from other server that have the corresponding cache.

Then is to check the answer number, authority and addition numbers. These three parameters indicate the how many records are there in the answer area. Then according to the answer, authority and answer number to extract the corresponding records.

The client will first skip the query part by get the length of the query section. If the name is a kind of pointer or mixed of pointer. The server will look for the domain name in the corresponding area of the received packet. If not, the client will first get the length of the sub-domain and then according the length of the sub-domain to separate each domain using dot, and if the domain's length is zero which means the end of the domain. Then the client will follow the protocol to get the following section value according the specific length. For A record, the IP address's length is 4, and the IP address is in numeric format, so there is no separate byte like the domain name section. For the CNAME records, the data part will be resolved with the same method that used to resolve the domain name, since the CNAME value is still a name. For the MX records, the data is the domain name of the mail exchange server and there are corresponding A records for the MX server, so after get an MX records, there always have one or several A records in the addition part.

DNS local server:

The DNS local server is the core of the whole DNS program, is will separate different queries to single one and request the query to the corresponding server.

The specific process is as follow.

1. Process the query from the DNS client

The client will first send the query to the local server, the local server will first get the parameter including RD bit and query number from the header of the DNS packet to make a further process. Then the server will according to the query number to extract the query records one by one and then according to whether the client want iterative or recursive query. The local will first get the name field in the query records. Each domain name will start with the length of it, also it is the method to separate each domain. The last domain is the TLD (top level domain) which will be used for recursive query. Because according to the requirement, for the recursive query, the local server will send the packet directly to the TLD server, so the top level domain of the query domain name is important.

2. Check whether there is a cache

After get each domain name, the local server will first check whether there is a cache at local server. By combine the query type (A/CNAME/MX) and the domain name, the local server will get the corresponding file and by reading the file, in the file, there are data and its corresponding TTL, if it is the records of MX server, there is addition preference in the cache file.

After getting the data from the cache, the local server will continue to check for the next query until there are no queries. If there are no cache for the specific query, then the local server will query to the corresponding server, if it's iterative query, the local server will query the root server for the TLD name server.

3. Query the upper level server.

After receiving an iterative query, the server will first send the query to the root server and the root server will resolve the top level domain of the queried domain name and then return the responsible next level name serve with and NS record in the answer section as well as a corresponding record that indicates the IP address of the name server. After the local server receiving the name server of the next domain, it will query the next level domain name server until there are answers or no such domain name returned by the SOA. The program will use iterative method to achieve that. After receiving the results, the local server will return the results to the client.

4. Storing the response and response to the client

After receiving the answer from the name server which is from the records in the SOA, the local server will automatically store the answer.

DNS root server:

The local server will send the query to the root server if and only if there is no cache in the local server.

1. Resolve the query form the local server:

The local server will send the packet to the root server and the local server will extract the TLD of the query domain, since the local server will first split the queries if there are multiple query in the received packet, there is only one query in the packet that send to the root server. The server will just get only one query in the query field, which will be more easier to handle and process.

2. Check the TLD in the local records and return the result:

The TLD information is stored in the local NS folder in the root server. After getting the TLD of the requested domain name, the root server will start check the TLD server that responsible for the TLD domain, if there are the record in the folder, the root server will read the file and also read the corresponding record in the folder to get the IP address of the TLD server. If there are no TLD record in the root server, the server will read the SOA file and put the SOA information into the authority field, and also the Rcode will be set to 0x03 to indicate "no such name". If there is

another TLD or 2nd level server in the record, the root server will return the domain name of the server in the authority filed.

DNS TLD server:

1. Resolve the query form the local server

This program is the same as the root server. It will not be discussed here.

2. Check the 2nd level domain in the local records and return the result to local server:

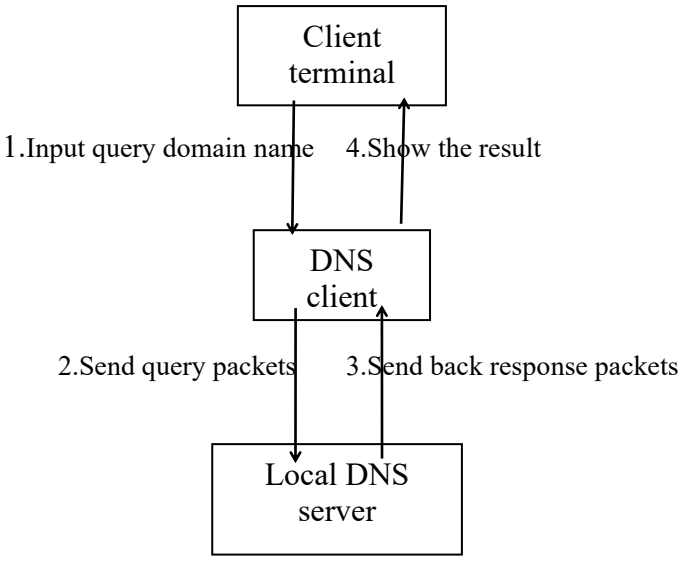
The program is almost the same as the root server, the only difference is that the TLD server will resolve the 2nd level domain instead of the TLD domain and the record in the NS file is not different from the TLD server.

DNS 2nd level server:

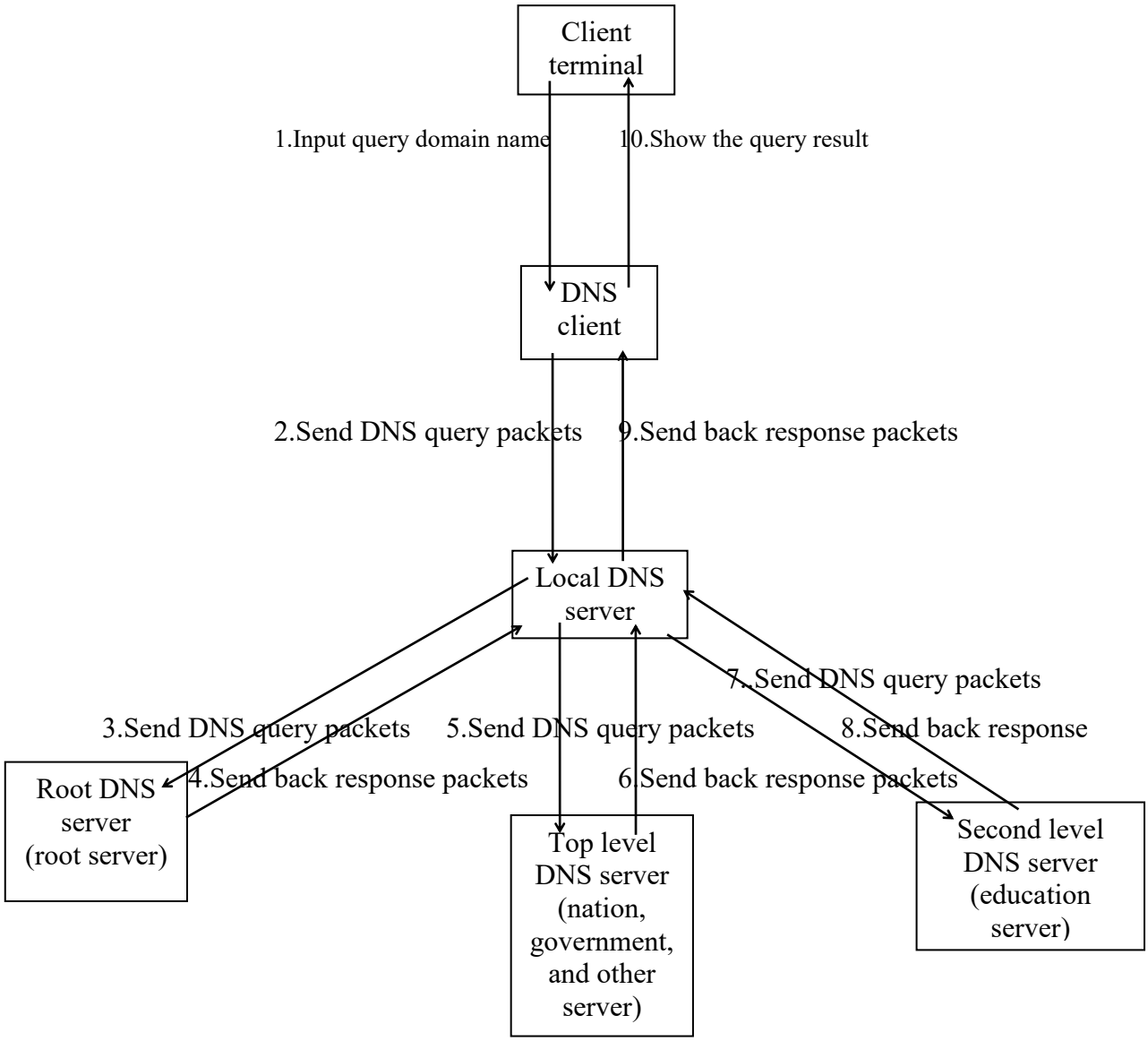
The 2nd level server's program is almost the same as the TLD server, if there are no lower level name serves, the 2nd level name server must have the record for the domain and without calculate the TTL, since the name server is the SOA of the domain name, the domain name and it's record is always correct and available.

3.2 Overall flow chart

When local DNS server received query packets, if it can find the query domain name in cache file, it will send back the response packets directly to the DNS client.



When local DNS server received query packets, if it can find the query domain name in cache file, it will proceed DNS iteration resolution.



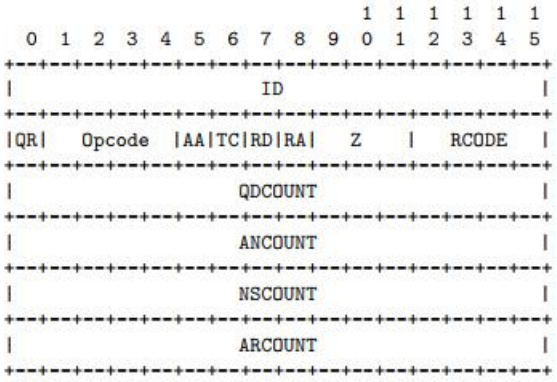
3.3 Design of data structures

Structure of a DNS packet:

```
+---------------------+
|        Header       |
+---------------------+
|       Question      |
+---------------------+
|        Answer       |
+---------------------+
|      Authority      |
+---------------------+
|      Additional     |
+---------------------+
```

The question section is the query of the client for the server. The answer section is answer of the query. The additional section is used when querying an MX type.

Structure of DNS packet header:



ID A 16 bit identifier assigned by the program that generates any kind of query. This identifier is copied the corresponding reply and can be used by the requester to match up replies to outstanding queries. You should select a new, random 16 bit number for each request.

QR A one bit field that specifies whether this message is a query (0), or a response (1). Obviously, you should use 0 for your requests, and expect to see a 1 in the response you receive.

OPCODE A four bit field that specifies kind of query in this message. You should use 0, representing a standard query.

AA Authoritative Answer - this bit is only meaningful in responses, and specifies that the responding name server is an authority for the domain name in question section. You should use this bit to report whether or not the response you receive is authoritative.

TC TrunCation - specifies that this message was truncated. For this project, you must exit and return an error if you receive a response that is truncated.

RD Recursion Desired - this bit directs the name server to pursue the query recursively. You should use 1, representing that you desire recursion.

RA Recursion Available - this be is set or cleared in a response, and denotes whether recursive query support is available in the name server. Recursive query support is optional. You must exit and return an error if you receive a response that indicates the server does not support recursion.

Z Reserved for future use. You must set this field to 0.

RCODE Response code - this 4 bit field is set as part of responses. The values have the following interpretation:

0 No error condition

1 Format error - The name server was unable to interpret the query.

2 Server failure - The name server was unable to process this query due to a problem with the name server.

3 Name Error - Meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist.

4 Not Implemented - The name server does not support the requested kind of query.

5 Refused - The name server refuses to perform the specified operation for policy reasons.

You should set this field to 0, and should assert an error if you receive a response indicating an error condition. You should treat 3 differently, as this represents the case where a requested name doesn't exist.

QDCOUNT an unsigned 16 bit integer specifying the number of entries in the question section. You should set this field to 1, indicating you have one question.

ANCOUNT an unsigned 16 bit integer specifying the number of resource records in the answer section. You should set this field to 0, indicating you are not providing any answers.

NSCOUNT an unsigned 16 bit integer specifying the number of name server resource records in the authority records section. You should set this field to 0, and should ignore any response entries in this section.

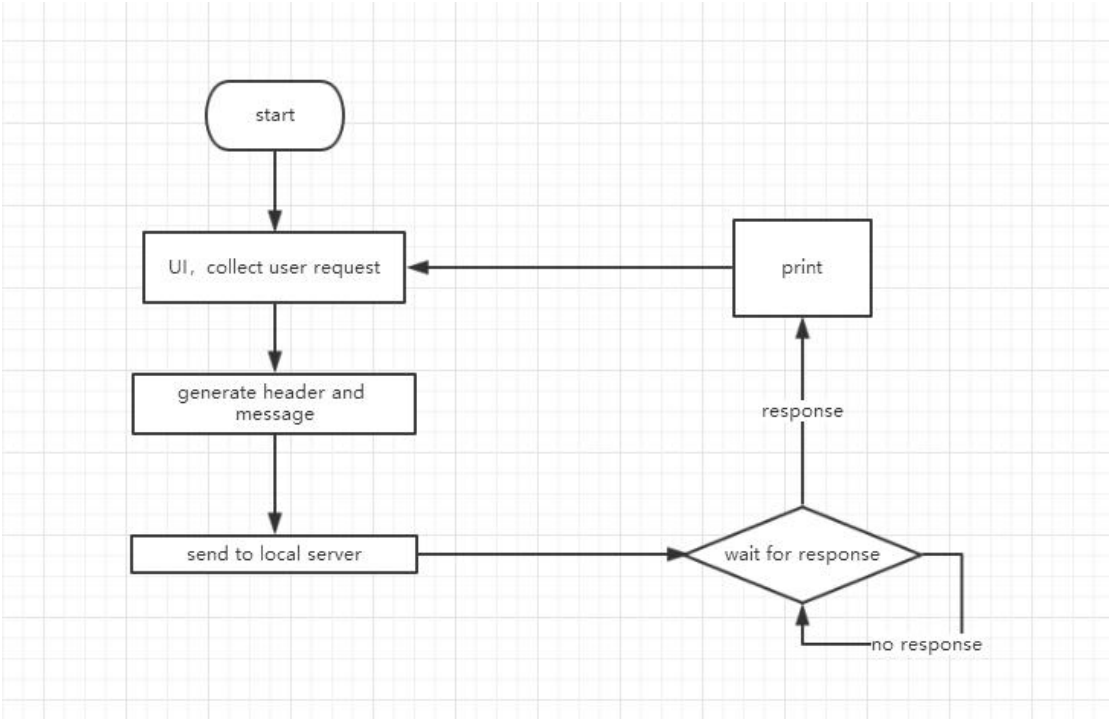
ARCOUNT an unsigned 16 bit integer specifying the number of resource records in the additional records section. You should set this field to 0, and should ignore any response entries in this section.

4. Detailed Design

4.1 Design analysis of each module

DNS Client:

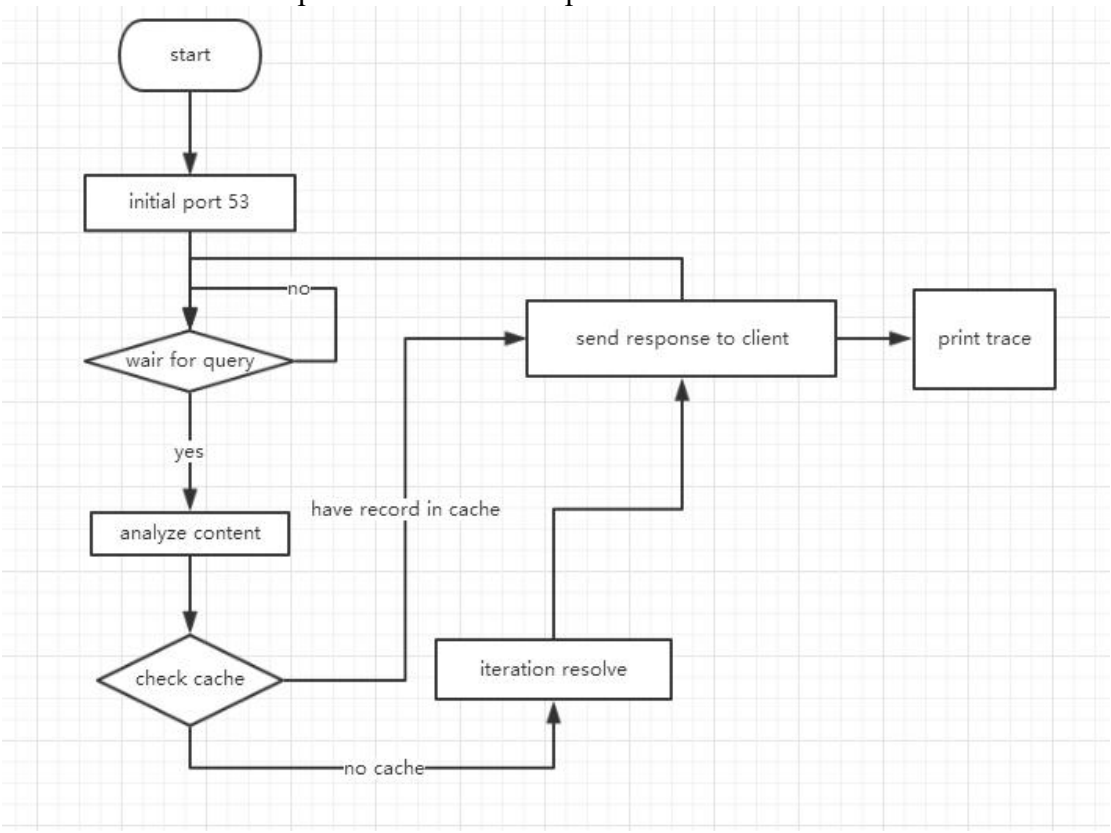
The Client should have these several functions: initial a socket, generate a dns request header, generate a dns request message, send the request to local server and receive the response, and print the query result as well as supply an user interface for user to operator.



DNS local server:

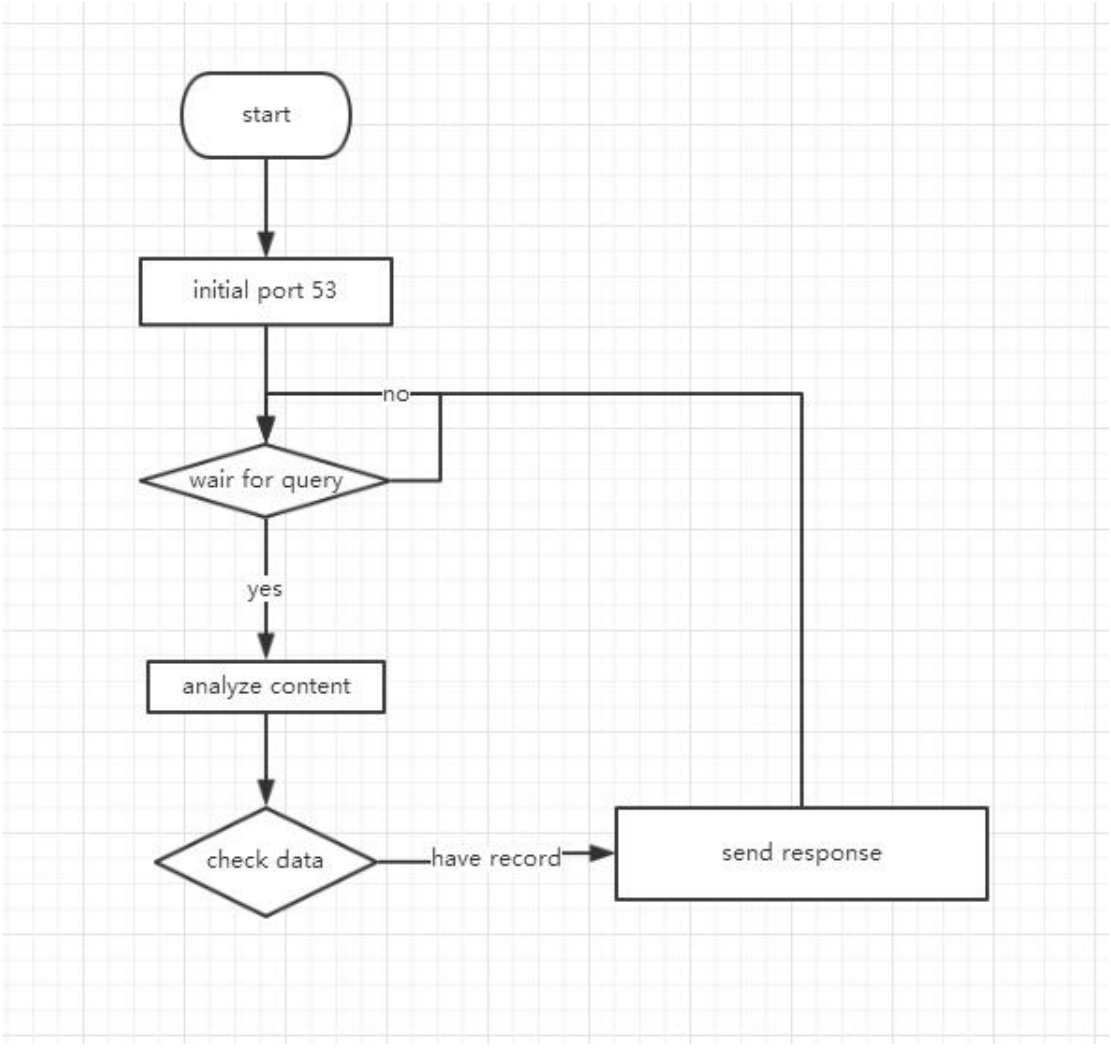
The Local server should have these several functions: initial the socket of 53

and another one, receive the query from client, cache and Iterative resolve and finally print the trace. To achieve the cache , the local server have 3 method to manage the cache file: check cache ,add cache and clear cache. To achieve receive, the local server should have a method to accept the dns query packet and analyze the content. The local server will first check whether the query is already in the cache. If not, server will Iterative resolve. To achieve this, the server have a method named iteration(), the method have the ability to generate a dns request header, generate a dns request message and send the packet to root dns server. After the local receive the response from root server, the local server have a method to resend to packet to the server in root response. After getting the final result, the local server will response the result to client and use printtrace method to print the trace.



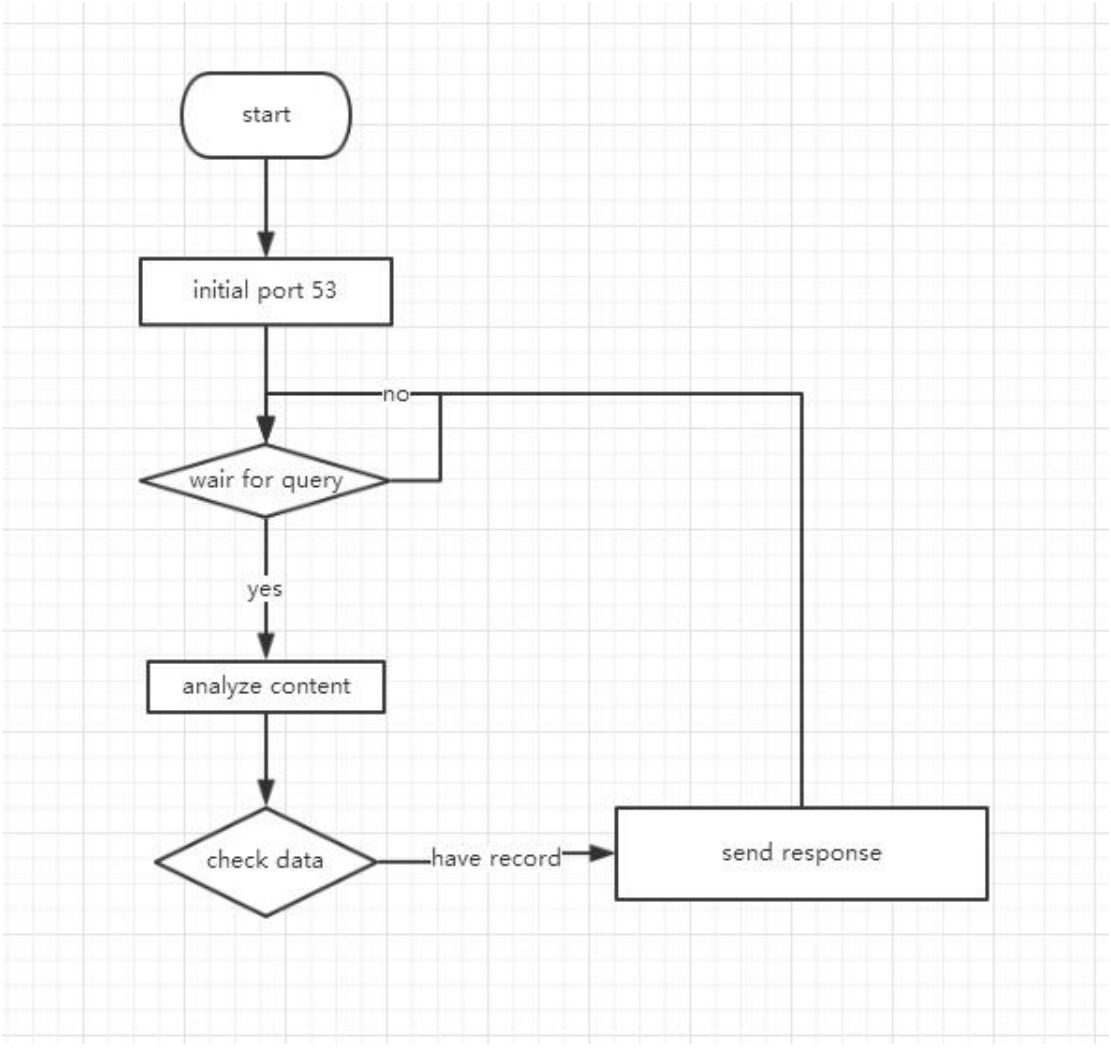
DNS root server:

The root server should have these several functions: initial the socket of 53, receive dns request, check the data in root , response the NS record. To achieve receive, the server have an method to read the content of the dns query packet. And use another method to check the data in file. Then, the server have a method to generate a query response.



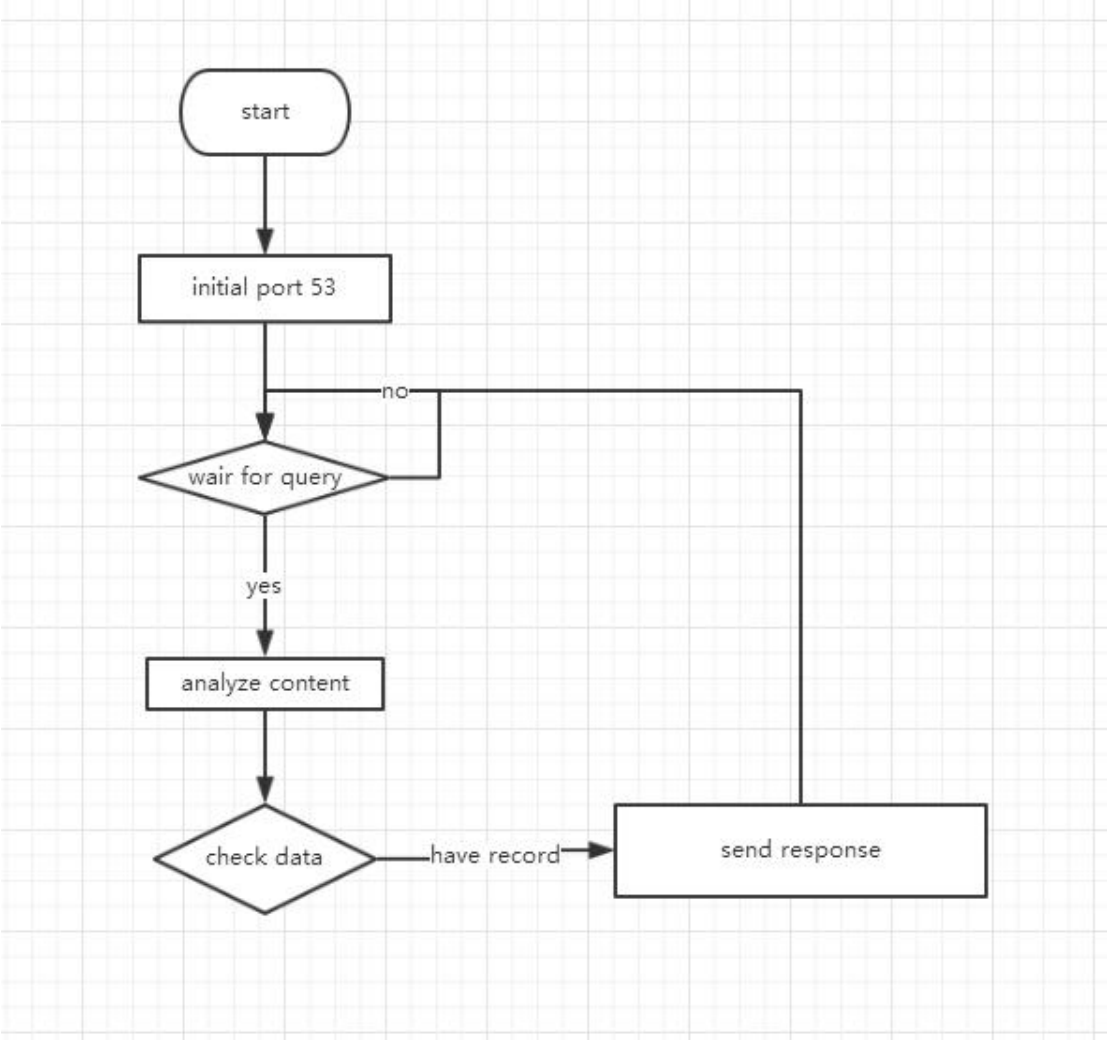
DNS TLD server:

The TLD server should have these several functions: initial the socket of 53, receive dns request, check the data in root , response the NS record. To achieve receive, the server have an method to read the content of the dns query packet. And use another method to check the data in file. Then, the server have a method to generate a query response.



DNS 2nd level server:

The 2nd level server should have these several functions: initial the socket of 53, receive dns request, check the data in root , response the NS record. To achieve receive, the server have an method to read the content of the dns query packet. And use another method to check the data in file. Then, the server have a method to generate a query response.



5. Result

All the files of the project:

```
student@BUPTIA:~/DNSproject$ ls
cache.txt  client.out  dns.h        education.out  gov.c    gov.txt  local.out  nation.out  other.c    othersvr.c  other.txt  root.out
client.c   dns.c       education.c  education.txt  gov.out  local.c  nation.c   nation.txt  other.out  othersvr.h  root.c     root.txt
```

Open the local server: `student@BUPTIA:~/DNSproject$ sudo ./local.out`

Open the root server: `sudo ./root.out`

Open the nation server: `sudo ./nation.out`

Open the other server: `sudo ./other.out`

Open the government server: `sudo ./gov.out`

Open the education server: `sudo ./education.out`

Open the client:

```
student@BUPTIA:~/DNSproject$ sudo ./client.out
[sudo] password for student:
##############################
Please choose the query type
1.A      2.CNAME      3.MX
```

Then input 1 2 or 3 to choose the query type(choose type A):

```
##############################
Please choose the query type
1.A      2.CNAME      3.MX
1

Please choose the resolution type
1.iteration      2.recursion
1

Please enter the domain name,multiple names should be separated by space
```

The database of education:

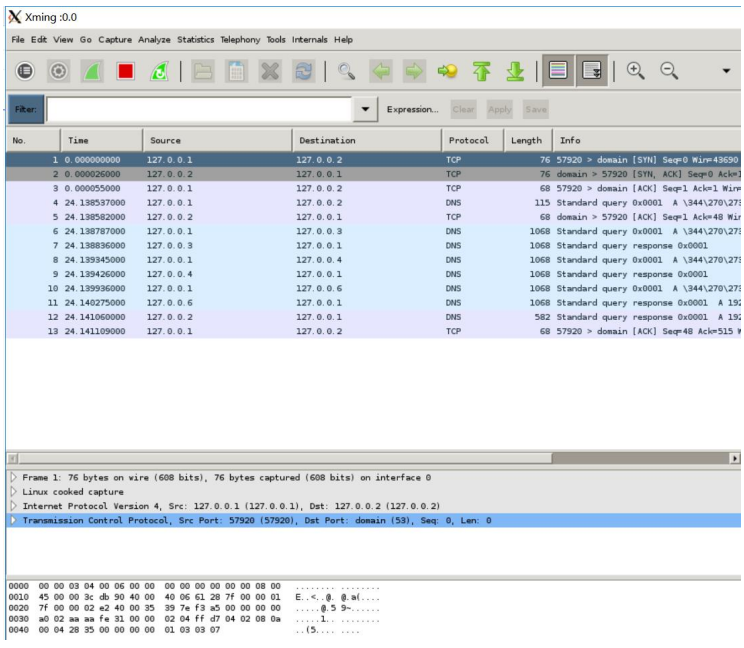
```
1 127.0.0.1:2000   2 127.0.0.1:2000   3 127.0.0.1:2000
Edu
www.bupt.edu.cn IN A 192.168.1.25
邮件服务器.北邮.教育.中国 IN MX 邮件.北邮.教育.中国
邮件.北邮.教育.中国 IN A 192.168.1.37
bbs.bupt.edu.cn IN CNAME www.bupt.edu.cn
主页.北邮.教育.中国 IN A 192.168.1.25
~
~
~
~
~
```

Query for 主页. 北邮. 教育. 中国

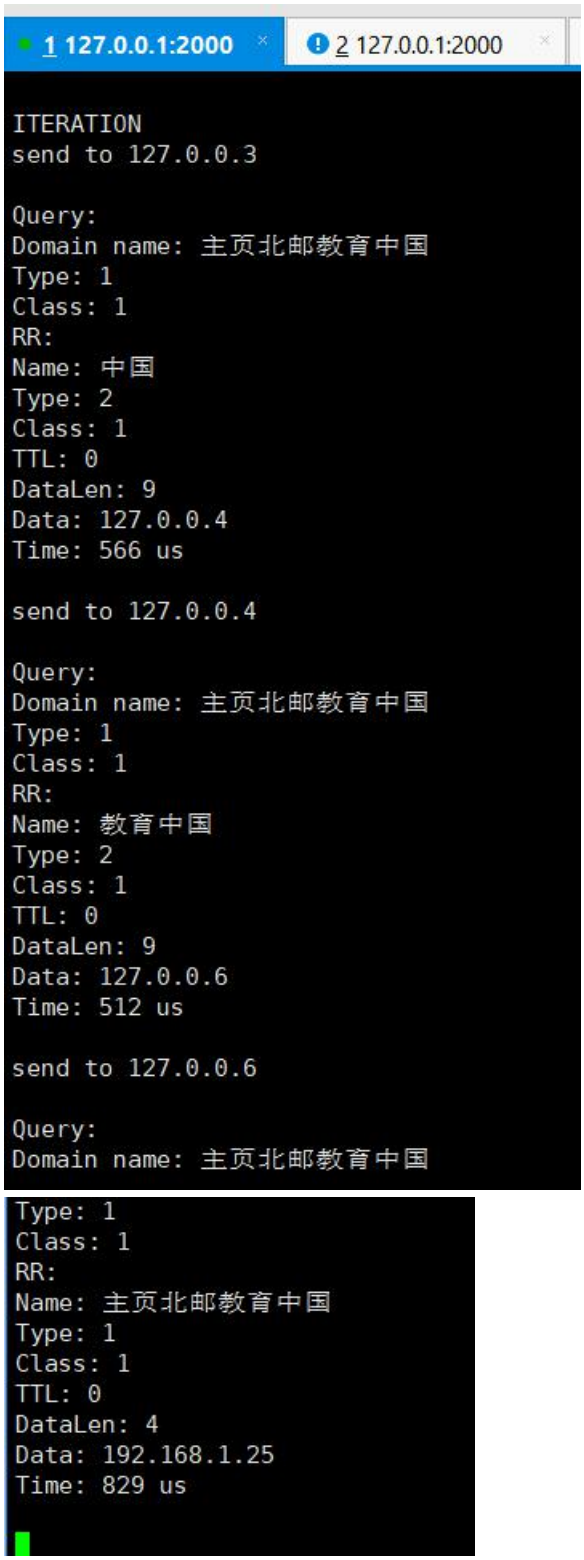
```
Please enter the domain name,multiple names should be separated by space
主页.北邮.教育.中国
Get IP address, the IP address is: 192.168.1.25
```

The answer is 192.168.1.25, it is the same as it is in the database.

Packets captured by wireshark:



The trace path result is showed on local server:



The result will also be stored in cache.txt

If query the same domain name again, the result will be sent back from the cache.

```
Please enter the domain name,multiple names should be separated by space
主页.北邮.教育.中国
Get IP address, the IP address is: 192.168.1.25
主页.北邮.教育.中国
Get IP address, the IP address is: 192.168.1.25
```

1 127.0.0.1:2000 | 2 127.0.0.1:2000

```
DataLen: 9
Data: 127.0.0.4
Time: 509 us

send to 127.0.0.4

Query:
Domain name: 主页北邮教育中国
Type: 1
Class: 1
RR:
Name: 教育中国
Type: 2
Class: 1
TTL: 0
DataLen: 9
Data: 127.0.0.6
Time: 701 us

send to 127.0.0.6

Query:
Domain name: 主页北邮教育中国
Type: 1
Class: 1
RR:
Name: 主页北邮教育中国
Type: 1
Class: 1
TTL: 0
DataLen: 4
Data: 192.168.1.25
Time: 412 us

get from cache
```

This time we choose CNAME as the query type and query for bbs.bupt.edu.cn:

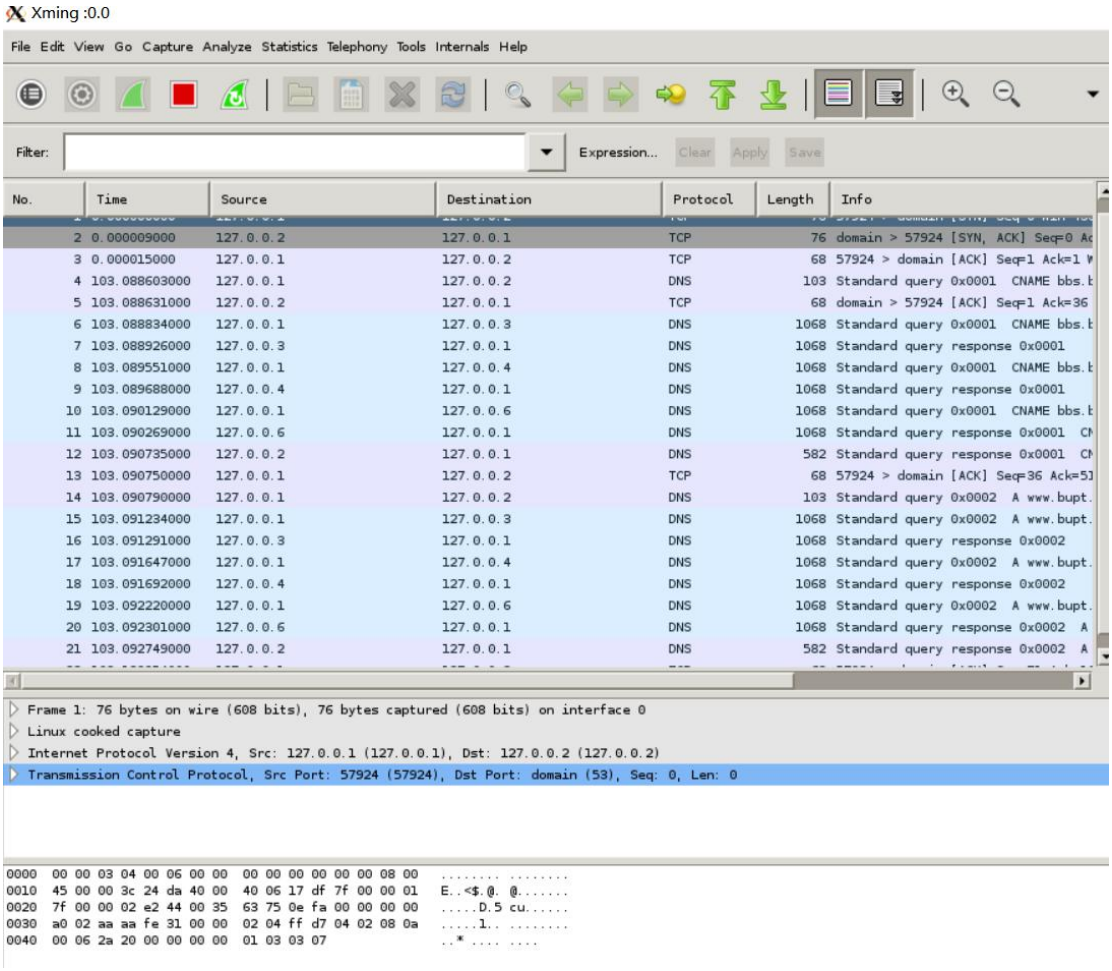
```
^Cstudent@BUPTIA:~/DNSproject$ sudo ./client.out
##############################
Please choose the query type
1.A      2.CNAME      3.MX
2

Please choose the resolution type
1.iteration      2.recursion
1

Please enter the domain name,multiple names should be separated by space
bbs.bupt.edu.cn
Get canonical name , the canonical name is: www.bupt.edu.cn
Get IP address, the IP address is: 192.168.1.25
```

The answer is 192.168.1.25, it is the same as it is in the database.

The packets captured by wireshark:



This time we choose MX as the query type and query for 邮件服务器.北邮.教育.中国

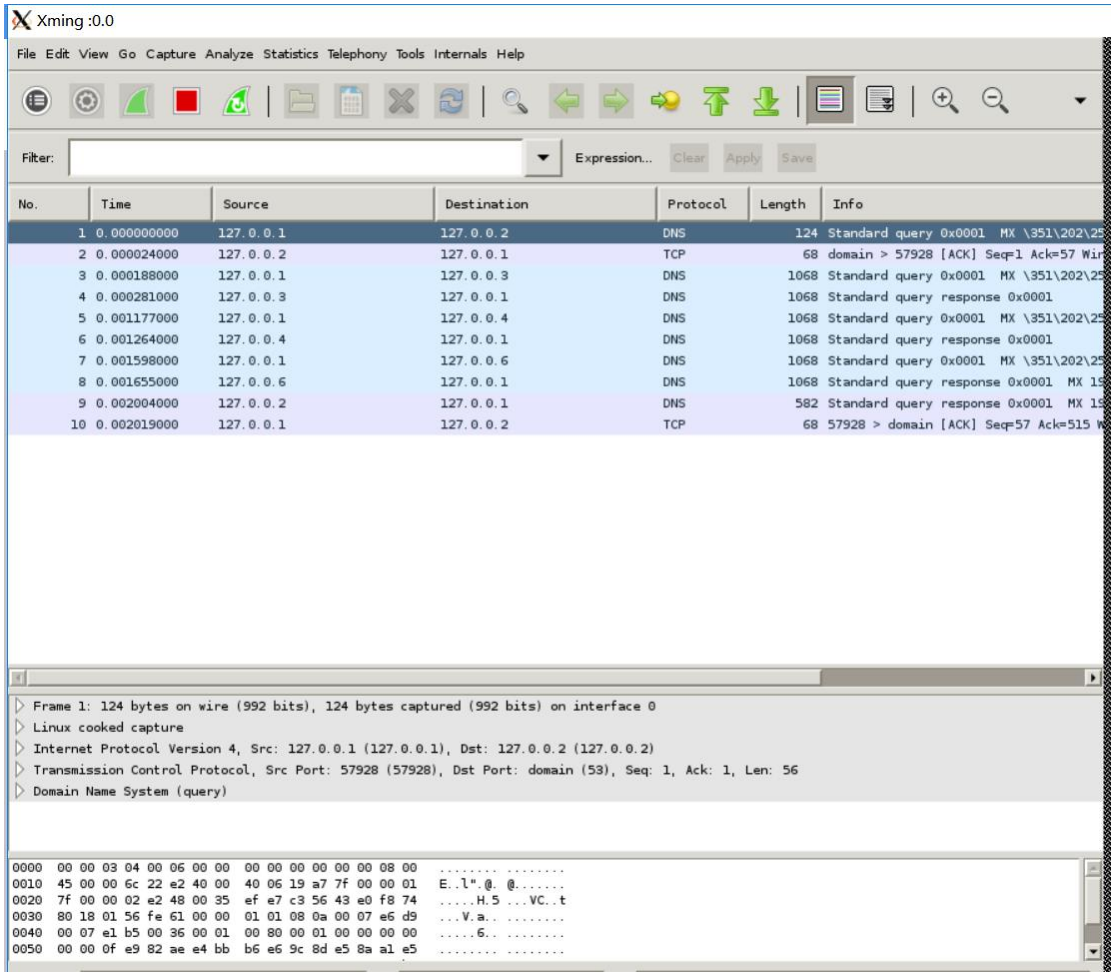
```
^Cstudent@BUPTIA:~/DNSproject$ sudo ./client.out
############################
Please choose the query type
1.A      2.CNAME      3.MX
3

Please choose the resolution type
1.iteration      2.recursion
1

Please enter the domain name,multiple names should be separated by space
邮件服务器.北邮.教育.中国
Get mail server, the mail server name is: 邮件.北邮.教育.中国. And it's IP address is 192.168.1.37
```

The answer is 192.168.1.25, it is the same as it is in the database.

The packets captured by wireshark:



Multiple queries and the results:

```
nails.bupt.edu.cn www.bupt.edu.cn
Get IP address, the IP address is: 192.168.1.37
Get IP address, the IP address is: 192.168.1.25
```

6. Summary and conclusion

WenhaoZhang(2015213067): Analysis of requirements, design the structure and modules of the project, write codes of root.c, nation.c, education.c, government.c other.c, othersvr.c, othersvr.h

HeZhu(2015213059): Write codes of client.c and local.c and make up the database.

We are using dns every time we surfing online. Every moment ,there may be a dns server process our query and give us response. However, we never care about it's operating principle. Though the theory course taught us the dns protocol, but the actually running program is still a mist. And this coursework helping us know more about it.After this coursework, we think that we've already got the basic ability to do network programming.

Appendix: Source codes

Client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <sys/time.h>
#include <memory.h>
#include <strings.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define DNS_TYPE_A                  1
#define DNS_TYPE_NS                 2
#define DNS_TYPE_CNAME                  5
#define DNS_TYPE_MX                 15

//Client

struct header_flags { //大小端字节序
        uint8_t rcode:4;
        uint8_t z:3;
        uint8_t ra:1;

        uint8_t rd:1;
        uint8_t tc:1;
        uint8_t aa:1;
        uint8_t opcode:4;
        uint8_t qr:1;
};

struct header {
        uint16_t id;
        struct header_flags *flags;
        uint16_t queries;
        uint16_t answers;
        uint16_t auth_rr;
        uint16_t add_rr;
};

struct query {
        unsigned char *name;
        uint16_t type;
```

```
        uint16_t class;
};

struct record {
        unsigned char *name;
        uint16_t type;
        uint16_t class;
        uint32_t ttl;
        uint16_t len;
        unsigned char *data;
};

//DNS Local Server 的 socket
int server_socket;

//Buffer
unsigned char buf[512];

//记录发送出请求和接收回应的时刻
struct timeval start, end;

//逆转换域名
unsigned char *get_data_name(unsigned char *data) {
        unsigned char *name = malloc(64);
        memcpy(name, data, strlen((const char *) data) + 1);

        //3www5baidu3com0 -> www.baidu.com
        int i = 0;
        for (; i < strlen((const char *) name); i++) {
                int num = name[i];
                int j;
                for (j = 0; j < num; j++) {
                        name[i] = name[i + 1];
                        i++;
                }
                name[i] = '.';
        }
        name[i - 1] = '\0';
        return name;
}

//读取域名 3www5baidu3com0
unsigned char *get_name(int *loc, unsigned char *reader) {
        unsigned char *name = malloc(64);
        int num = 0;

        *loc = 0;

        while (*reader != 0) {
                name[num++] = *reader;
```

```
            reader++;
        }
        name[num] = '\0';
        (*loc)++;

        return get_data_name(name);
}

char *ptr(char *ip) {
        char *suffix = "in-addr.arpa";

        char *result = malloc(64);
        bzero(result, 64);
        char temp[4][4] = {0};

        char string[64] = {0};
        memcpy(string, ip, strlen(ip));

        char *token = strtok(string, ".");
        int i;
        for (i = 0; i < 4; i++) {
            memcpy(temp[3 - i], token, strlen(token));
            strcat(temp[3 - i], ".");
            token = strtok(NULL, ".");
        }
        for (i = 0; i < 4; i++) {
            strcat(result, temp[i]);
        }
        strcat(result, suffix);

        return result;
}

//转换域名
void transform(unsigned char *query_name, unsigned char *hostname) {
        int loc = 0;
        char host[64] = {0};
        memcpy(host, hostname, strlen((const char *) hostname));
        strcat(host, ".");

        int i;
        for (i = 0; i < strlen(host); i++) {
            if (host[i] == '.') {
                *query_name++ = (unsigned char) (i - loc);
                for (; loc < i; loc++) {
                    *query_name++ = (unsigned char) host[loc];
                }
                loc++;
            }
        }
```

```
        *query_name = 0;
}

size_t make_rr(size_t loc, unsigned char *buf, struct record *pRecord) {
        unsigned char *name = pRecord->name;
        uint16_t type = pRecord->type;
        uint16_t class = pRecord->class;
        uint32_t ttl = pRecord->ttl;
        uint16_t len = pRecord->len;
        unsigned char *data = pRecord->data;

        //Query Name
        unsigned char *query_name = &buf[loc];
        transform(query_name, name);
        loc += strlen((const char *) query_name) + 1;

        //Type
        type = htons(type);
        memcpy(&buf[loc], &type, sizeof(type));
        loc += sizeof(type);

        //Class
        class = htons(class);
        memcpy(&buf[loc], &class, sizeof(class));
        loc += sizeof(class);

        //TTL
        ttl = htonl(ttl);
        memcpy(&buf[loc], &ttl, sizeof(ttl));
        loc += sizeof(ttl);

        //Length
        len = htons(len);
        memcpy(&buf[loc], &len, sizeof(len));
        loc += sizeof(len);

        //Data
        memcpy(&buf[loc], data, pRecord->len);
        loc += pRecord->len;

        return loc;
}

//Header
struct header *make_header(uint16_t id, uint8_t qr, uint8_t rd, uint8_t rcode, uint16_t
queries, uint16_t answers, uint16_t auth_rr, uint16_t add_rr) {
        struct header *header = malloc(sizeof(struct header));

        if (id > 0)
                header->id = id;
```

```
        else
                header->id = (uint16_t) clock(); //clock()的作用是获得时间，在这里
的作用只是用作随机数

        header->flags = malloc(sizeof(struct header_flags));
        header->flags->qr = qr;
        header->flags->opcode = 0;
        header->flags->aa = 0;
        header->flags->tc = 0;
        header->flags->rd = rd; //0 迭代 1 递归
        header->flags->ra = 1;
        header->flags->z = 0;
        header->flags->rcode = rcode;

        header->queries = queries;
        header->answers = answers;
        header->auth_rr = auth_rr;
        header->add_rr = add_rr;

        return header;
}

//DNS packet
size_t make_packet(size_t loc, unsigned char *buf, struct header *header, struct query
**queries, struct record **answers, struct record **auths, struct record **adds) {
        //Transaction ID
        uint16_t id = htons(header->id);
        memcpy(&buf[loc], &id, sizeof(id));
        loc += sizeof(id);

        //Flags
        uint16_t flags;
        memcpy(&flags, header->flags, sizeof(*(header->flags)));
        flags = htons(flags);
        memcpy(&buf[loc], &flags, sizeof(flags));
        loc += sizeof(flags);

        //Number of queries in packet
        uint16_t query_num = htons(header->queries);
        memcpy(&buf[loc], &query_num, sizeof(query_num));
        loc += sizeof(query_num);

        //Number of answers in packet
        uint16_t answer_num = htons(header->answers);
        memcpy(&buf[loc], &answer_num, sizeof(answer_num));
        loc += sizeof(answer_num);

        //Number of authoritative records in packet
        uint16_t auth_rr = htons(header->auth_rr);
        memcpy(&buf[loc], &auth_rr, sizeof(auth_rr));
```

```
	loc += sizeof(auth_rr);

	//Number of additional records in packet
	uint16_t add_rr = htons(header->add_rr);
	memcpy(&buf[loc], &add_rr, sizeof(add_rr));
	loc += sizeof(add_rr);

	int i;
	for (i = 0; i < header->queries; i++) {
		unsigned char *name = queries[i]->name;
		uint16_t type = queries[i]->type;
		uint16_t class = queries[i]->class;

		//Query Name
		unsigned char *query_name = &buf[loc];
		transform(query_name, name);
		loc += strlen((const char *) query_name) + 1;

		//Query Type
		type = htons(type);
		memcpy(&buf[loc], &type, sizeof(type));
		loc += sizeof(type);

		//Query Class
		class = htons(class);
		memcpy(&buf[loc], &class, sizeof(class));
		loc += sizeof(class);
	}

	for (i = 0; i < header->answers; i++)
		loc = make_rr(loc, buf, answers[i]);

	for (i = 0; i < header->auth_rr; i++)
		loc = make_rr(loc, buf, auths[i]);

	for (i = 0; i < header->add_rr; i++)
		loc = make_rr(loc, buf, adds[i]);

	return loc;
}

//得到类型名称
char *get_type_name(uint16_t type) {
	switch (type) {
		case DNS_TYPE_A:
			return "A";
		case DNS_TYPE_NS:
			return "NS";
		case DNS_TYPE_CNAME:
			return "CNAME";
```

```
                case DNS_TYPE_PTR:
                        return "PTR";
                case DNS_TYPE_MX:
                        return "MX";
                default:
                        return "Unknown";
        }
}

//Resource Record
struct record *read_rr(size_t *loc, unsigned char *reader) {
        *loc = 0;
        int temp_loc = 0;
        struct record *pRecord = malloc(sizeof(struct record));

        //Record Name
        pRecord->name = get_name(&temp_loc, reader);
        reader += temp_loc;
        *loc += temp_loc;
        printf("Name: <%s> ", pRecord->name);

        //Record Type
        memcpy(&pRecord->type, reader, sizeof(pRecord->type));
        pRecord->type = ntohs(pRecord->type);
        reader += sizeof(pRecord->type);
        *loc += sizeof(pRecord->type);
        printf("Type: <%s> ", get_type_name(pRecord->type));

        //Record Class
        memcpy(&pRecord->class, reader, sizeof(pRecord->class));
        pRecord->class = ntohs(pRecord->class);
        reader += sizeof(pRecord->class);
        *loc += sizeof(pRecord->class);

        //Record TTL
        memcpy(&pRecord->ttl, reader, sizeof(pRecord->ttl));
        pRecord->ttl = ntohl(pRecord->ttl);
        reader += sizeof(pRecord->ttl);
        *loc += sizeof(pRecord->ttl);
        printf("Time to live: <%u> ", pRecord->ttl);

        //Record Length
        memcpy(&pRecord->len, reader, sizeof(pRecord->len));
        pRecord->len = ntohs(pRecord->len);
        reader += sizeof(pRecord->len);
        *loc += sizeof(pRecord->len);

        //Record Data
        unsigned char *_data = malloc(pRecord->len);
        bzero(_data, pRecord->len);
```

```
		memcpy(_data, reader, pRecord->len);
		pRecord->data = _data;

		if (pRecord->type == DNS_TYPE_A) {
			struct sockaddr_in t;
			memcpy(&t.sin_addr, pRecord->data, sizeof(struct in_addr));
			printf("Address: <%s> ", inet_ntoa(t.sin_addr));
		} else if (pRecord->type == DNS_TYPE_NS) {
			printf("Name Server: <%s> ", get_name(&temp_loc, reader));
		} else if (pRecord->type == DNS_TYPE_CNAME) {
			printf("CNAME: <%s> ", get_name(&temp_loc, reader));
		} else if (pRecord->type == DNS_TYPE_PTR) {
			printf("Domain Name: <%s> ", get_name(&temp_loc, reader));
		} else if (pRecord->type == DNS_TYPE_MX) {
			uint16_t preference;
			memcpy(&preference, pRecord->data, sizeof(preference));
			reader += sizeof(preference);
			printf("Preference: <%hu> Mail Exchange: <%s> ", ntohs(preference),
get_name(&temp_loc, reader));
			reader -= sizeof(preference);
		}
		printf("\n");
		reader += pRecord->len;
		*loc += pRecord->len;

		return pRecord;
}

//Header
struct header *read_header(size_t *loc, unsigned char *reader) {
		*loc = 0;
		struct header *header = malloc(sizeof(struct header));

		//Transaction ID
		memcpy(&header->id, reader, sizeof(header->id));
		header->id = ntohs(header->id);
		reader += sizeof(header->id);
		*loc += sizeof(header->id);

		//Flags
		uint16_t flags;
		memcpy(&flags, reader, sizeof(flags));
		flags = ntohs(flags);
		header->flags = malloc(sizeof(*(header->flags)));
		memcpy(header->flags, &flags, sizeof(*(header->flags)));
		reader += sizeof(*(header->flags));
		*loc += sizeof(*(header->flags));
		printf("Flags: <0x%04x> ", flags);

		//Queries
```

```
memcpy(&header->queries, reader, sizeof(header->queries));
header->queries = ntohs(header->queries);
reader += sizeof(header->queries);
*loc += sizeof(header->queries);

//Answers
memcpy(&header->answers, reader, sizeof(header->answers));
header->answers = ntohs(header->answers);
reader += sizeof(header->answers);
*loc += sizeof(header->answers);

//Authoritative
memcpy(&header->auth_rr, reader, sizeof(header->auth_rr));
header->auth_rr = ntohs(header->auth_rr);
reader += sizeof(header->auth_rr);
*loc += sizeof(header->auth_rr);

//Additional
memcpy(&header->add_rr, reader, sizeof(header->add_rr));
header->add_rr = ntohs(header->add_rr);
reader += sizeof(header->add_rr);
*loc += sizeof(header->add_rr);

if (header->flags->qr == 1) {
        if (header->flags->aa)
                printf("Authoritative response. ");
        else
                printf("Non-authoritative response. ");
}

if (header->flags->rd == 1)
        printf("Recursive query. ");
else
        printf("Iterative query. ");

if (header->flags->qr == 1) {
        if (header->flags->ra)
                printf("Server can do recursive queries. ");
        else
                printf("Server cannot do recursive queries. ");

        if (header->flags->rcode == 0)
                printf("No error. ");
        else if (header->flags->rcode == 2)
                printf("Server failure. ");
        else if (header->flags->rcode == 3)
                printf("No such name. ");
}

printf("\n");
```

```
	return header;
}

//Response
void resolve_tcp_response_packet() {
	uint16_t length = 0;

	recv(server_socket, &length, sizeof(uint16_t), 0);
	length = ntohs(length);
	bzero(buf, length);
	recv(server_socket, buf, length, 0);

	gettimeofday(&end, NULL);
	printf("Local server response in %lf seconds.\n", end.tv_sec - start.tv_sec +
(end.tv_usec - start.tv_usec) / 1000000.0);

	size_t loc = 0;
	unsigned char *reader = buf;

	//Header
	struct header *header = read_header(&loc, reader);
	reader += loc;

	//Queries
			int i;
	for (i = 0; i < header->queries; i++) {
		read_rr(&loc, reader);
		reader += loc;
	}

	//Answers
	for (i = 0; i < header->answers; i++) {
		read_rr(&loc, reader);
		reader += loc;
	}

	//Authoritative
	for (i = 0; i < header->auth_rr; i++) {
		read_rr(&loc, reader);
		reader += loc;
	}

	//Additional
	for (i = 0; i < header->add_rr; i++) {
		read_rr(&loc, reader);
		reader += loc;
	}
}
```

```
int main(int argc, char **argv) {
	//创建 socket
	if ((server_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) <
0) {
		printf("socket() failed.\n");
		exit(1);
	}

	//地址
	struct sockaddr_in server_add;
	memset(&server_add, 0, sizeof(struct sockaddr_in));
	server_add.sin_family = AF_INET;
	server_add.sin_port = htons(53);
	server_add.sin_addr.s_addr = inet_addr("127.0.0.2");

	connect(server_socket, (struct sockaddr *) &server_add, sizeof(server_add));

	rd:
	//表明递归或迭代
	int rd;
	printf("1. Recursive or 0. Iterative ?\n");
	scanf("%d", &rd);

	if (rd != 0 && rd != 1) {
		printf("Wrong input!\n");
		goto rd;
	}

	int query_num;
	printf("Queries amount?\n");
	scanf("%d", &query_num);

	bzero(buf, 512);

	size_t loc = sizeof(uint16_t);

	struct header *header = make_header(0, 0, rd, 0, query_num, 0, 0, 0);

	struct query **queries = malloc(sizeof(struct query) * query_num);

	char type[64] = {0};
	char name[64] = {0};
	int i;
	for (i = 0; i < query_num; i++) {
		start:
		printf("Type: ");
		scanf("%s", type);
		printf("Domain name: ");
		scanf("%s", name);
		queries[i] = malloc(sizeof(struct query));
```

```
            if (!strcmp(type, "A")) {
                queries[i]->name = (unsigned char *) name;
                queries[i]->type = 1;
            } else if (!strcmp(type, "CNAME")) {
                queries[i]->name = (unsigned char *) name;
                queries[i]->type = 5;
            } else if (!strcmp(type, "PTR")) {
                queries[i]->name = (unsigned char *) ptr(name);
                queries[i]->type = 12;
            } else if (!strcmp(type, "MX")) {
                queries[i]->name = (unsigned char *) name;
                queries[i]->type = 15;
            } else {
                printf("Wrong type!\n");
                goto start;
            }
            queries[i]->class = 1;
        }

        struct record **answers = NULL, **auths = NULL, **adds = NULL;

        loc = make_packet(loc, buf, header, queries, answers, auths, adds);

        uint16_t dns_packet_size = htons((uint16_t) (loc - 2));

        //前两个字节表示 packet 的长度
        memcpy(buf, &dns_packet_size, sizeof(uint16_t));

        send(server_socket, buf, loc, 0);

        //记录时间
        gettimeofday(&start, NULL);

        for (i = 0; i < query_num; i++)
            resolve_tcp_response_packet();

        close(server_socket);
        return 0;
}
```

Local.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
//数据结构
```

```
struct DNS_Head
{
   unsigned short id;
   unsigned short tag;
   unsigned short queryNum;
   unsigned short answerNum;
   unsigned short authorNum;
   unsigned short addNum;
};

struct DNS_Query
{
   unsigned char *qname;
   unsigned short qtype;
   unsigned short qclass;
};

struct DNS_RR
{
   unsigned char *rname;
   unsigned short rtype;
   unsigned short rclass;
   unsigned int ttl;
   unsigned short datalen;
   unsigned char *rdata;
};
//DNS 服务器 ip
unsigned char ip[64];
//包缓存
unsigned char tcpsendpacket[1024];
unsigned char tcprecvpacket[1024];
unsigned char udpsendpacket[1024];
unsigned char udprecvpacket[1024];
int tcpsendpos,tcprecvpos, udpsendpos, udprecvpos;
//rr 缓存
struct DNS_RR rrdb[4];
int rrnum;
unsigned char db[1024];
unsigned char* dbptr;
//函数表
void formdomain(unsigned char*);
void gethead(unsigned char*, int*, struct DNS_Head*);
void getquery(unsigned char*, int*, struct DNS_Query*);
void getrr(unsigned char*, int*, struct DNS_RR*);
void setstdhead(unsigned char*, int*);
void setreshead(unsigned char*, int*, int id);
void setaquery(unsigned char*, int*, unsigned char*);
void setrr(unsigned char*, int*, struct DNS_RR);

int main()
```

```
{
    //初始化 socket
    int tcpServerSocket, tcpClientSocket;
    struct sockaddr_in tcpServerAddr, tcpClientAddr;
    int tcpaddrlen;

    int udpSocket;
    struct sockaddr_in udpRemoteAddr;
    int udpaddrlen;

    tcpServerSocket = socket(AF_INET, SOCK_STREAM, 0);
    udpSocket = socket(AF_INET, SOCK_DGRAM, 0);

    tcpServerAddr.sin_family = AF_INET;
    tcpServerAddr.sin_port = htons(53);
    tcpServerAddr.sin_addr.s_addr = inet_addr("127.0.0.2");
    udpRemoteAddr.sin_family = AF_INET;
    udpRemoteAddr.sin_port = htons(53);

    if (bind(tcpServerSocket, (struct sockaddr*)&tcpServerAddr, sizeof(struct
sockaddr)) < 0)
    {
        printf("server binds failed\n");
        exit(0);
    }
    listen(tcpServerSocket, 4);
    tcpaddrlen = sizeof(struct sockaddr_in);
    udpaddrlen = sizeof(struct sockaddr_in);

    tcpClientSocket = accept(tcpServerSocket, (struct sockaddr*)&tcpClientAddr,
&tcpaddrlen);
    //开始计时
    clock();
    //初始化缓存
    rrnum = 0;
    dbptr = db;

    while (1)
    {
        //初始化 DNS 服务器 ip
        strcpy(ip, "127.0.0.3");
        udpRemoteAddr.sin_addr.s_addr = inet_addr(ip);
        //接受报文
        recv(tcpClientSocket, tcprecvpacket, sizeof(tcprecvpacket), 0);
        printf("server recv packet from client\n");

        struct DNS_Head head;
        struct DNS_Query query;
        struct DNS_RR rr;
```

```
        //解析报文
        tcprecvpos = 2;
        gethead(tcprecvpacket, &tcprecvpos, &head);
        getquery(tcprecvpacket, &tcprecvpos, &query);
        //查询缓存
        int i;
        for (i = 0; i < rrnum; ++i)
        {
            if (strcmp(rrdb[i].rname, query.qname) == 0 && rrdb[i].rtype == query.qtype)
            {
                printf("using cache\n");
                tcpsendpos = 2;
                setreshead(tcpsendpacket, &tcpsendpos, head.id);
                setrr(tcpsendpacket, &tcpsendpos, rrdb[i]);
                *(unsigned short*)tcpsendpacket = htons(tcpsendpos - 2);
                send(tcpClientSocket, tcpsendpacket, tcpsendpos, 0);
                printf("server send packet to client\n");
                i = rrnum + 1;
                break;
            }
        }
        if (i == rrnum + 1)
            continue;
        //生成查询报文
        for (i = 2; i < sizeof(tcprecvpacket); ++i)
        {
            udpsendpacket[i - 2] = tcprecvpacket[i];
        }
        udpsendpos = tcprecvpos - 2;

        while (1)
        {
            int flag = 0;
            //发送报文
            sendto(udpSocket, udpsendpacket, udpsendpos, 0, (struct
sockaddr*)&udpRemoteAddr, sizeof(struct sockaddr));
            printf("server send packet to %s\n", ip);
            //接受报文
            recvfrom(udpSocket, udprecvpacket, sizeof(udprecvpacket), 0, (struct
sockaddr*)&udpRemoteAddr, &udpaddrlen);
            printf("server recv packet from %s\n", ip);
            //解析报文
            udprecvpos = 0;
            gethead(udprecvpacket, &udprecvpos, &head);
            getrr(udprecvpacket, &udprecvpos, &rr);

            switch (head.tag)
            {
            case (unsigned short)0x8000://回答是结果
```

```
            for (i = 2; i < sizeof(udprecvpacket); ++i)
            {
               tcpsendpacket[i] = udprecvpacket[i - 2];
            }
            tcpsendpos = udprecvpos + 2;
            *(unsigned short*)tcpsendpacket = htons(tcpsendpos - 2);

            if (rr.rtype == 1 || rr.rtype == 2)//A 或者 NS 类型的查询
            {
               flag = 1;
               break;
            }
            else if (rr.rtype == 5)//CNAME 类型的查询
            {
               struct DNS_Head h1;
               struct DNS_Query q1;
               udpsendpos = 0;
               gethead(udpsendpacket, &udpsendpos, &h1);
               getquery(udpsendpacket, &udpsendpos, &q1);

               if (q1.qtype == 5)//发送的查询是 CNAME
               {
                  flag = 1;
                  break;
               }
               else if (q1.qtype == 1)//发送的查询是 A 进行第二次查询
               {
                  //生成查询报文
                  unsigned char domain[64];
                  strcpy(domain, rr.rdata);
                  formdomain(domain);

                  udpsendpos = 0;
                  setstdhead(udpsendpacket, &udpsendpos);
                  setaquery(udpsendpacket, &udpsendpos, domain);
                  //初始化 DNS 服务器地址
                  strcpy(ip, "127.0.0.3");
                  udpRemoteAddr.sin_addr.s_addr = inet_addr(ip);
                  //查询
                  while (1)
                  {
                     sendto(udpSocket, udpsendpacket, udpsendpos, 0, (struct
sockaddr*)&udpRemoteAddr, sizeof(struct sockaddr));
                     printf("server send packet to %s\n", ip);
                     recvfrom(udpSocket, udprecvpacket, sizeof(udprecvpacket), 0,
(struct sockaddr*)&udpRemoteAddr, &udpaddrlen);
                     printf("server recv packet from %s\n", ip);
```

```
                struct DNS_Head h2;
                struct DNS_RR r2;
                udprecvpos = 0;
                gethead(udprecvpacket, &udprecvpos, &h2);
                getrr(udprecvpacket, &udprecvpos, &r2);

                if (r2.rtype == 2)
                {
                    strcpy(ip, r2.rdata);
                    udpRemoteAddr.sin_addr.s_addr = inet_addr(ip);
                }
                else if (r2.rtype == 1)
                {
                    for (i = 2; i < sizeof(udprecvpacket); ++i)
                    {
                        tcpsendpacket[i] = udprecvpacket[i - 2];
                    }
                    tcpsendpos = udprecvpos + 2;
                    *(unsigned short*)tcpsendpacket = htons(tcpsendpos - 2);
                    flag = 1;
                    break;
                }
            }
            break;
        }
    }
    else if (rr.rtype == 15)//MX 类型的查询 进行第二次查询
    {
        //生成查询报文
        unsigned char domain[64];
        strcpy(domain, rr.rdata);
        formdomain(domain);

        udpsendpos = 0;
        setstdhead(udpsendpacket, &udpsendpos);
        setaquery(udpsendpacket, &udpsendpos, domain);
        //初始化 DNS 服务器 ip
        strcpy(ip, "127.0.0.3");
        udpRemoteAddr.sin_addr.s_addr = inet_addr(ip);
        //查询
        while (1)
        {
            sendto(udpSocket, udpsendpacket, udpsendpos, 0, (struct
sockaddr*)&udpRemoteAddr, sizeof(struct sockaddr));
            printf("server send packet to %s\n", ip);

            recvfrom(udpSocket, udprecvpacket, sizeof(udprecvpacket), 0, (struct
sockaddr*)&udpRemoteAddr, &udpaddrlen);
            printf("server recv packet from %s\n", ip);
```

```
                struct DNS_Head h2;
                struct DNS_RR r2;
                udprecvpos = 0;
                gethead(udprecvpacket, &udprecvpos, &h2);
                getrr(udprecvpacket, &udprecvpos, &r2);

                if (r2.rtype == 2)
                {
                    strcpy(ip, r2.rdata);
                    udpRemoteAddr.sin_addr.s_addr = inet_addr(ip);
                }
                else if (r2.rtype == 1)
                {
                    *(unsigned short*)(tcpsendpacket + 12) = htons(1);
                    setrr(tcpsendpacket, &tcpsendpos, r2);
                    *(unsigned short*)tcpsendpacket = htons(tcpsendpos - 2);
                    flag = 1;
                    break;
                }
            }
            break;
        }
        break;
    case (unsigned short)0x8400://回答是另一个 DNS 服务器
        if (rr.rtype == 2)//NS
        {
            //设置新的 ip 地址
            strcpy(ip, rr.rdata);
            udpRemoteAddr.sin_addr.s_addr = inet_addr(ip);
        }
        break;
    }
    if (flag == 1)
        break;
}

send(tcpClientSocket, tcpsendpacket, tcpsendpos, 0);
printf("server send packet to client\n");

rrnum = rrnum % 4;
tcpsendpos = 2;
gethead(tcpsendpacket, &tcpsendpos, &head);
getrr(tcpsendpacket, &tcpsendpos, &rrdb[rrnum]);
if (rrdb[rrnum].rtype != 15)
{
    strcpy(dbptr, rrdb[rrnum].rname);
    rrdb[rrnum].rname = dbptr;
    dbptr += strlen(dbptr) + 1;
    strcpy(dbptr, rrdb[rrnum].rdata);
    rrdb[rrnum].rdata = dbptr;
```

```
            dbptr += strlen(dbptr) + 1;
            rrnum++;
        }
        //打印 cache
        printf("cache:\n");
            for (i = 0; i < rrnum; ++i)
        {
            printf("%s %s\n", rrdb[i].rname, rrdb[i].rdata);
        }
    }
}

void formdomain(unsigned char* domain)
{
    unsigned char dotdomain[64];
    int i = 0;
    while (domain[i] != '\0')
    {
        dotdomain[i] = domain[i];
        ++i;
    }
    dotdomain[i] = '\0';

    int counter;
    unsigned char* flag;
    unsigned char* ptr = dotdomain;
    while (1)
    {
        counter = 0;
        flag = domain++;
        while (1)
        {
            if (*ptr == '.' || *ptr == '\0')
                break;
            *domain++ = *ptr++;
            counter++;
        }
        *flag = (unsigned char)counter;
        if (*ptr == '\0')
        {
            *domain = '\0';
            break;
        }
        ptr++;
    }
}

void gethead(unsigned char* packet, int* packetlen, struct DNS_Head* head)
{
    packet += *packetlen;
```

```
    *head = *(struct DNS_Head*)packet;

    head->id = ntohs(head->id);
    head->tag = ntohs(head->tag);
    head->queryNum = ntohs(head->queryNum);
    head->answerNum = ntohs(head->answerNum);
    head->authorNum = ntohs(head->authorNum);
    head->addNum = ntohs(head->addNum);

    *packetlen += 12;
}

void getquery(unsigned char* packet, int* packetlen, struct DNS_Query* query)
{
    packet += *packetlen;

    query->qname = packet;
    *packetlen += strlen(packet) + 1;
    packet += strlen(packet) + 1;

    query->qtype = ntohs(*(unsigned short*)packet);
    packet += 2;
    query->qclass = ntohs(*(unsigned short*)packet);

    *packetlen += 4;
}

void getrr(unsigned char* packet, int* packetlen, struct DNS_RR* rr)
{
    packet += *packetlen;

    rr->rname = packet;
    *packetlen += strlen(packet) + 1;
    packet += strlen(packet) + 1;

    rr->rtype = ntohs(*(unsigned short*)packet);
    packet += 2;
    rr->rclass = ntohs(*(unsigned short*)packet);
    packet += 2;
    rr->ttl += ntohl(*(unsigned int*)packet);
    packet += 4;
    rr->datalen = ntohs(*(unsigned short*)packet) - 1;
    packet += 2;
    packet++;
    rr->rdata = packet;

    *packetlen += rr->datalen + 10 + 1;
}
```

```
void setstdhead(unsigned char* packet, int* packetlen)
{
   packet += *packetlen;

   struct DNS_Head head;

   head.id = htons((unsigned short)clock());
   head.tag = htons((unsigned short)0x0000);
   head.queryNum = htons((unsigned short)1);
   head.answerNum = htons((unsigned short)0);
   head.authorNum = htons((unsigned short)0);
   head.addNum = htons((unsigned short)0);

   unsigned char* ptr = (unsigned char*)&head;
   int i;
   for (i = 0; i < 12; ++i)
   {
      *packet++ = *ptr++;
   }

   *packetlen += 12;
}

void setreshead(unsigned char* packet, int* packetlen, int id)
{
   packet += *packetlen;

   struct DNS_Head head;

   head.id = htons((unsigned short)id);
   head.tag = htons((unsigned short)0x8000);
   head.queryNum = htons((unsigned short)0);
   head.answerNum = htons((unsigned short)1);
   head.authorNum = htons((unsigned short)0);
   head.addNum = htons((unsigned short)0);

   unsigned char* ptr = (unsigned char*)&head;
   int i;
   for (i = 0; i < 12; ++i)
   {
      *packet++ = *ptr++;
   }

   *packetlen += 12;
}

void setaquery(unsigned char* packet, int* packetlen, unsigned char* domain)
{
   struct DNS_Query query;
```

```
    query.qname = domain;
    query.qtype = htons(1);
    query.qclass = htons(1);

    packet += *packetlen;

    int i;
    unsigned char* ptr = query.qname;
    for (i = 0; i < strlen((char*)query.qname) + 1; ++i)
    {
        *packet++ = *ptr++;
    }
    *packetlen += strlen((char*)query.qname) + 1;

    *(unsigned short*)packet = query.qtype;
    packet += 2;
    *(unsigned short*)packet = query.qclass;

    *packetlen += 4;
}

void setrr(unsigned char* packet, int* packetlen, struct DNS_RR rr)
{
    packet += *packetlen;

    int i;
    unsigned char* ptr = rr.rname;
    for (i = 0; i < strlen(rr.rname) + 1; ++i)
    {
        *packet++ = *ptr++;
    }
    *packetlen += strlen(rr.rname) + 1;

    *(unsigned short*)packet = htons(rr.rtype);
    packet += 2;
    *(unsigned short*)packet = htons(rr.rclass);
    packet += 2;
    *(unsigned int*)packet = htonl(rr.ttl);
    packet += 4;
    *(unsigned short*)packet = htons(rr.datalen + 1);
    packet += 2;
    *packet++ = 0x68;
    ptr = rr.rdata;
    for (i = 0; i < rr.datalen; ++i)
    {
        *packet++ = *ptr++;
    }
    *packetlen += rr.datalen + 10 + 1;
}
```

root.c

```
struct DNSHeader header;
struct DNSQuery query;
struct DNSRR rr[10];
struct sockaddr_in clientAddr;          //记录 UDP 传输中的客户端地址
unsigned char dnsmessage[1024];//报文
unsigned char* rr_ptr;                  //记录 rr 的位置
unsigned char* get_rr_ptr;              //用于 getRR 的指针
char* filename;                                 //文件名
int socketudp;                                  //套接字标识符
int err;                                        //记录返回值
int len_header_query = 0;       //记录报文中资源记录之前部分的长度

void initSocket(const char* svr, const char* _filename)
{
        filename = _filename;
   //初始化 UDP 套接字
   socketudp = socket(AF_INET , SOCK_DGRAM , IPPROTO_UDP);
   struct sockaddr_in addr;
   memset(&addr, 0, sizeof(addr));
   addr.sin_family = AF_INET;
   addr.sin_port = htons(PORT);
   addr.sin_addr.s_addr = inet_addr(svr);
   err = bind(socketudp, (struct sockaddr*)&addr, sizeof(struct sockaddr));
   if(err < 0)
   {
      printf("bind failed: %d\n",errno);
      exit(0);
   }
}

int containStr(const unsigned char* dname, const unsigned char* rname, const
unsigned char type)
{
   int len1 = strlen(dname);
   int len2 = strlen(rname);
   int i = len1 - 1, j = len2 - 1;
   if(type == 'N')
   {
      for(;; i--,j--) //自后向前遍历
      {
         if(j < 0)//rname 读完,表示每一位都匹配上
         {
            return 1;
         }
         if(dname[i] != rname[j])//某一位未匹配上
            return -1;
      }
```

```
    }
    else
    {
        if(strcmp(dname, rname) == 0)
        {
            return 1;
        }
        return -1;
    }
}

void setRR()
{
    unsigned char temp_rr[256];
    rr_ptr = getMessage(&header, &query, dnsmessage, &len_header_query);
    get_rr_ptr = rr_ptr;
    memset(rr_ptr, 0, sizeof(dnsmessage) - len_header_query);//清空报文中的 rr 部分
    unsigned char* ptr = dnsmessage;
    ptr += 6;
    *((unsigned short*)ptr) = 0;//报头的资源记录数置零
    ptr += 2;
    *((unsigned short*)ptr) = 0;
    FILE *fp;
    fp = fopen(filename, "r");
    if(fp == NULL)
    {
        printf("the file cannot be opened: %d\n", errno);
        exit(0);
    }
    unsigned char dname[128];
    memset(dname, 0, sizeof(dname));
    unsigned char* temp_ptr = query.name;
    int flag, i, num = 0;
    for(;;)//将 query.name 转换成标准的域名格式
    {
        flag = (int)temp_ptr[0];
        for(i = 0; i < flag; i++)
        {
            dname[i + num] = temp_ptr[i + 1];
        }
        temp_ptr += (flag + 1);
        if((int)temp_ptr[0] == 0)
            break;
        dname[flag + num] = '.';
        num += (flag + 1);
    }
    while(fgets(temp_rr, sizeof(temp_rr), fp) != NULL)//逐行查询
    {
        unsigned char rname[128];//记录一条资源记录中第一个空格前的部分
```

```
        unsigned char type;//记录第二个空格后的字符，也就是 RR 类型的首字母
        memset(rname, 0, sizeof(rname));
        int len = strlen(temp_rr);
        for(i = 0; i < len; i++)
        {
            if(temp_rr[i] == ' ')
                break;
        }
        memcpy(rname, temp_rr, i);
        int numofspace = 0;
        for(i = 0; i < len; i++)
        {
            if(temp_rr[i] == ' ')
                numofspace++;
            if(temp_rr[i] == ' ' && numofspace == 2)
                break;
        }
        type = temp_rr[i + 1];
        if(containStr(dname, rname, type) == 1)
        {
            addRR(temp_rr, rname);
        }
        memset(temp_rr, 0, sizeof(temp_rr));
    }
    err = fclose(fp);
    if(err == EOF)
    {
        printf("The file close failed: %d\n", errno);
        exit(0);
    }
}

void addRR(const unsigned char* str, const unsigned char* rname)
{
    unsigned char buf[128];
    unsigned char* ptr = dnsmessage;
    ptr += 6;
    *((unsigned short*)ptr) = htons(htons(*((unsigned short*)ptr)) + 1);//报头的资源
记录数加 1
    ptr = buf;
    char *pos;
    int n, len = 0;//len 记录域名的长度
    pos = (char*)rname;
    /*将域名存到 buf 中，buf 中存储每个域的长度和内容
    比如当前域是 edu.cn，存到 buf 中就变成了 3edu2cn0
    ,0 表示结尾*/
    for(;;)
    {
        n = strlen(pos) - (strstr(pos , ".") ? strlen(strstr(pos , ".")) : 0);
```

```
    *ptr ++ = (unsigned char)n;
    memcpy(ptr , pos , n);
    len += n + 1;
    ptr += n;
    if(!strstr(pos , "."))
    {
        *ptr = (unsigned char)0;
        ptr ++;
        len += 1;
        break;
    }
    pos += n + 1;
}
memcpy(rr_ptr, buf, len);
rr_ptr += len;
pos = (char*)str;
pos += (len + 2);
int flag = 0;
/*因为只考虑 A,NS,MX,CNAME 四种查询类型
，所以只做了匹配第一个字母的简单处理*/
switch(pos[0])
{
case'A':
{
    *((unsigned short*)rr_ptr) = htons(1);
    rr_ptr += 2;
    pos += 2;
    flag = 1;
    break;
}
case'N':
{
        unsigned char* _ptr = dnsmessage;
    _ptr += 6;
    *((unsigned short*)_ptr) = htons(htons(*((unsigned short*)_ptr)) - 1);
    _ptr += 2;
    *((unsigned short*)_ptr) = htons(htons(*((unsigned short*)_ptr)) + 1);
    *((unsigned short*)rr_ptr) = htons(2);
    rr_ptr += 2;
    pos += 3;
    break;
}
case'C':
{
    *((unsigned short*)rr_ptr) = htons(5);
    rr_ptr += 2;
    pos += 6;
    break;
}
case'M':
```

```
    {
        *((unsigned short*)rr_ptr) = htons(15);
        rr_ptr += 2;
        pos += 3;
        flag = 2;
        break;
    }
    }
    *((unsigned short*)rr_ptr) = htons(1);
    rr_ptr += 2;
    *((unsigned short*)rr_ptr) = htonl(0);
    rr_ptr += 4;
    len = strlen(pos);
    len = len - 2;//len - 2 是因为从文件中读取的字符串最后两位是回车加换行
    if (flag == 1)
    {
        *((unsigned short*)rr_ptr) = htons(4);
        rr_ptr += 2;
        struct in_addr addr;
        char ip[32];
        memset(ip, 0, sizeof(ip));
        memcpy(ip, pos, len);
        inet_aton(ip, &addr);
        *((unsigned long*)rr_ptr) = addr.s_addr;
        rr_ptr += 4;
    }
    else if(flag == 2)
    {
            *((unsigned short*)rr_ptr) = htons(len);
        rr_ptr += 2;
        memcpy(rr_ptr, pos - 3, 2);
        rr_ptr += 2;
        *rr_ptr = (unsigned char)len;
        rr_ptr += 1;
        memcpy(rr_ptr, pos, len);
        rr_ptr += len;
        memset(rr_ptr, 0, 1);
        rr_ptr++;
    }
    else
    {
        *((unsigned short*)rr_ptr) = htons(len);
        rr_ptr += 2;
        memcpy(rr_ptr, pos - 1, len + 1);
        rr_ptr += (len + 1);
    }
}

void setAddRR()
{
```

```
rr_ptr = getMessage(&header, &query, dnsmessage, &len_header_query);
rr_ptr = getRR(rr, &header, rr_ptr);
rr_ptr++;
int i, j;
for(j = 0; j < header.answerNum; j++)
{
   if(rr[i].type == 15)//找到 MX 对应 data 对应的 IP 地址
   {
      unsigned char temp_rr[256];
      unsigned char type;//记录第二个空格后的字符，也就是 RR 类型的首字母
      FILE *fp;
      fp = fopen(filename, "r");
      if(fp == NULL)
      {
         printf("the file cannot be opened: %d", errno);
         exit(0);
      }
      while(fgets(temp_rr, sizeof(temp_rr), fp) != NULL)//逐行查询
      {
         unsigned char rname[128];//记录一条资源记录中第一个空格前的部分
         memset(rname, 0, sizeof(rname));
         int len = strlen(temp_rr);
         for(i = 0; i < len; i++)
         {
            if(temp_rr[i] == ' ')
               break;
         }
         memcpy(rname, temp_rr, i);
         int numofspace = 0;
         for(i = 0; i < len; i++)
         {
            if(temp_rr[i] == ' ')
               numofspace++;
            if(temp_rr[i] == ' ' && numofspace == 2)
               break;
         }
         type = temp_rr[i + 1];
         if(containStr(rr[j].rdata, rname, type) == 1)
         {
            addRR(temp_rr, rname);
            unsigned char* ptr = dnsmessage;
            ptr += 6;
            /*因为添加 additional rr 也是用的添加 RR 的函数，所以
            需要报头的资源记录数减 1，然后附加资源记录数加 1*/
            *((unsigned short*)ptr) = htons(htons(*((unsigned short*)ptr)) - 1);
            ptr += 4;
            *((unsigned short*)ptr) = htons(htons(*((unsigned short*)ptr)) + 1);
         }
         memset(temp_rr, 0, sizeof(temp_rr));
```

```
        }
        err = fclose(fp);
        if(err == EOF)
        {
           printf("The file close failed: %d", errno);
           exit(0);
        }
        break;
     }
   }
}

void recvfromSvr(int flag)
{
        memset(dnsmessage, 0, 1024);
        switch(flag)
        {
                case 0:
                {
                        struct sockaddr_in addr;
                int len = sizeof(addr);
                err = recvfrom(socketudp, dnsmessage, sizeof(dnsmessage), 0, (struct
sockaddr*)&addr, &len);
                        break;
                }
                case 1:
                {
                        int len = sizeof(clientAddr);
                err = recvfrom(socketudp, dnsmessage, sizeof(dnsmessage), 0, (struct
sockaddr*)&clientAddr, &len);
                        break;
                }
        }
   if(err <= 0)//等于 0 时表示连接已终止
   {
      printf("UDP socket receive failed: %d\n", errno);
      exit(0);
   }
   int i;
}

void sendtoSvr(const unsigned char* svr, int flag)
{
        switch(flag)
        {
                case 0:
                {
        unsigned char* ptr = dnsmessage;
        ptr += 2;
        if (*((unsigned short*)ptr) == htons(0x8080))
```

```
        {
           *((unsigned short*)ptr) = htons(0x0080);
        }
        else if(*((unsigned short*)ptr) == htons(0x8180))
        {
           *((unsigned short*)ptr) = htons(0x0180);
        }
                        struct sockaddr_in destSvr;
                    memset(&destSvr, 0, sizeof(destSvr));
                    destSvr.sin_family = AF_INET;
                    destSvr.sin_port = htons(PORT);
                    destSvr.sin_addr.s_addr = inet_addr(svr);
                    int len = sizeof(dnsmessage);
                    err = sendto(socketudp, dnsmessage, len, 0, (struct
sockaddr*)&destSvr, sizeof(struct sockaddr));
                        break;
                }
                case 1:
                {
        unsigned char* ptr = dnsmessage;
        ptr += 2;
        if (*((unsigned short*)ptr) == htons(0x0080))
        {
           *((unsigned short*)ptr) = htons(0x8080);
        }
        else if(*((unsigned short*)ptr) == htons(0x0180))
        {
           *((unsigned short*)ptr) = htons(0x8180);
        }
                        err = sendto(socketudp, dnsmessage, sizeof(dnsmessage), 0,
(struct sockaddr*)&clientAddr, sizeof(struct sockaddr));
                }
        }
        if(err <= 0)
   {
      printf("send question to next dns failed: %d\n", errno);
      exit(0);
   }
}

void iterantion()
{
   printf("\nITERATION\n");
   sendtoSvr("", 1);
}

void recursion()
{
   printf("\nRECURSION\n");
   rr_ptr = getRR(rr, &header, get_rr_ptr);
```

```
    int i;
    for(i = 0; i < header.answerNum; i++)
    {
        if(rr[i].type == 2)
        {
            sendtoSvr(rr[i].rdata, 0);
            recvfromSvr(0);
            sendtoSvr("", 1);
        }
        else//如果查询类型不为 A 表示已经查到结果
        {
            sendtoSvr("", 1);
        }
    }
}

void process()
{
            while(1)
    {
        recvfromSvr(1);
        setRR();
        setAddRR();
        /*判断使用何种解析方式*/
        if(header.tag == 0x0080)
        {
            iterantion();
        }
        else if(header.tag == 0x0180)
        {
            recursion();
        }
    }
}
int main()
{
    initSocket(ROOT, "root.txt");
    process();
    return 0;
}
```

other.c

```
struct DNSHeader header;
struct DNSQuery query;
struct DNSRR rr[10];
struct sockaddr_in clientAddr;          //记录 UDP 传输中的客户端地址
unsigned char dnsmessage[1024];//报文
unsigned char* rr_ptr;                  //记录 rr 的位置
unsigned char* get_rr_ptr;              //用于 getRR 的指针
```

```
char* filename;                                    //文件名
int socketudp;                                     //套接字标识符
int err;                                           //记录返回值
int len_header_query = 0;      //记录报文中资源记录之前部分的长度

void initSocket(const char* svr, const char* _filename)
{
        filename = _filename;
   //初始化 UDP 套接字
   socketudp = socket(AF_INET , SOCK_DGRAM , IPPROTO_UDP);
   struct sockaddr_in addr;
   memset(&addr, 0, sizeof(addr));
   addr.sin_family = AF_INET;
   addr.sin_port = htons(PORT);
   addr.sin_addr.s_addr = inet_addr(svr);
   err = bind(socketudp, (struct sockaddr*)&addr, sizeof(struct sockaddr));
   if(err < 0)
   {
      printf("bind failed: %d\n",errno);
      exit(0);
   }
}

int containStr(const unsigned char* dname, const unsigned char* rname, const
unsigned char type)
{
   int len1 = strlen(dname);
   int len2 = strlen(rname);
   int i = len1 - 1, j = len2 - 1;
   if(type == 'N')
   {
      for(;; i--,j--) //自后向前遍历
      {
         if(j < 0)//rname 读完,表示每一位都匹配上
         {
            return 1;
         }
         if(dname[i] != rname[j])//某一位未匹配上
            return -1;
      }
   }
   else
   {
      if(strcmp(dname, rname) == 0)
      {
         return 1;
      }
      return -1;
   }
```

```
}

void setRR()
{
   unsigned char temp_rr[256];
   rr_ptr = getMessage(&header, &query, dnsmessage, &len_header_query);
   get_rr_ptr = rr_ptr;
   memset(rr_ptr, 0, sizeof(dnsmessage) - len_header_query);//清空报文中的 rr 部分
   unsigned char* ptr = dnsmessage;
   ptr += 6;
   *((unsigned short*)ptr) = 0;//报头的资源记录数置零
   ptr += 2;
   *((unsigned short*)ptr) = 0;
   FILE *fp;
   fp = fopen(filename, "r");
   if(fp == NULL)
   {
      printf("the file cannot be opened: %d\n", errno);
      exit(0);
   }
   unsigned char dname[128];
   memset(dname, 0, sizeof(dname));
   unsigned char* temp_ptr = query.name;
   int flag, i, num = 0;
   for(;;)//将 query.name 转换成标准的域名格式
   {
      flag = (int)temp_ptr[0];
      for(i = 0; i < flag; i++)
      {
         dname[i + num] = temp_ptr[i + 1];
      }
      temp_ptr += (flag + 1);
      if((int)temp_ptr[0] == 0)
         break;
      dname[flag + num] = '.';
      num += (flag + 1);
   }
   while(fgets(temp_rr, sizeof(temp_rr), fp) != NULL)//逐行查询
   {
      unsigned char rname[128];//记录一条资源记录中第一个空格前的部分
      unsigned char type;//记录第二个空格后的字符，也就是 RR 类型的首字母
      memset(rname, 0, sizeof(rname));
      int len = strlen(temp_rr);
      for(i = 0; i < len; i++)
      {
         if(temp_rr[i] == ' ')
            break;
      }
      memcpy(rname, temp_rr, i);
```

```
        int numofspace = 0;
        for(i = 0; i < len; i++)
        {
            if(temp_rr[i] == ' ')
                numofspace++;
            if(temp_rr[i] == ' ' && numofspace == 2)
                break;
        }
        type = temp_rr[i + 1];
        if(containStr(dname, rname, type) == 1)
        {
            addRR(temp_rr, rname);
        }
        memset(temp_rr, 0, sizeof(temp_rr));
    }
    err = fclose(fp);
    if(err == EOF)
    {
        printf("The file close failed: %d\n", errno);
        exit(0);
    }
}

void addRR(const unsigned char* str, const unsigned char* rname)
{
    unsigned char buf[128];
    unsigned char* ptr = dnsmessage;
    ptr += 6;
    *((unsigned short*)ptr) = htons(htons(*((unsigned short*)ptr)) + 1);//报头的资源
记录数加 1
    ptr = buf;
    char *pos;
    int n, len = 0;//len 记录域名的长度
    pos = (char*)rname;
    /*将域名存到 buf 中，buf 中存储每个域的长度和内容
    比如当前域是 edu.cn，存到 buf 中就变成了 3edu2cn0
    ,0 表示结尾*/
    for(;;)
    {
        n = strlen(pos) - (strstr(pos , ".") ? strlen(strstr(pos , ".")) : 0);
        *ptr ++ = (unsigned char)n;
        memcpy(ptr , pos , n);
        len += n + 1;
        ptr += n;
        if(!strstr(pos , "."))
        {
            *ptr = (unsigned char)0;
            ptr ++;
            len += 1;
```

```
        break;
    }
    pos += n + 1;
}
memcpy(rr_ptr, buf, len);
rr_ptr += len;
pos = (char*)str;
pos += (len + 2);
int flag = 0;
/*因为只考虑 A,NS,MX,CNAME 四种查询类型
，所以只做了匹配第一个字母的简单处理*/
switch(pos[0])
{
case'A':
{
    *((unsigned short*)rr_ptr) = htons(1);
    rr_ptr += 2;
    pos += 2;
    flag = 1;
    break;
}
case'N':
{
        unsigned char* _ptr = dnsmessage;
    _ptr += 6;
    *((unsigned short*)_ptr) = htons(htons(*((unsigned short*)_ptr)) - 1);
    _ptr += 2;
    *((unsigned short*)_ptr) = htons(htons(*((unsigned short*)_ptr)) + 1);
    *((unsigned short*)rr_ptr) = htons(2);
    rr_ptr += 2;
    pos += 3;
    break;
}
case'C':
{
    *((unsigned short*)rr_ptr) = htons(5);
    rr_ptr += 2;
    pos += 6;
    break;
}
case'M':
{
    *((unsigned short*)rr_ptr) = htons(15);
    rr_ptr += 2;
    pos += 3;
    flag = 2;
    break;
}
}
*((unsigned short*)rr_ptr) = htons(1);
```

```
    rr_ptr += 2;
    *((unsigned short*)rr_ptr) = htonl(0);
    rr_ptr += 4;
    len = strlen(pos);
    len = len - 2;//len - 2 是因为从文件中读取的字符串最后两位是回车加换行
    if (flag == 1)
    {
        *((unsigned short*)rr_ptr) = htons(4);
        rr_ptr += 2;
        struct in_addr addr;
        char ip[32];
        memset(ip, 0, sizeof(ip));
        memcpy(ip, pos, len);
        inet_aton(ip, &addr);
        *((unsigned long*)rr_ptr) = addr.s_addr;
        rr_ptr += 4;
    }
    else if(flag == 2)
    {
            *((unsigned short*)rr_ptr) = htons(len);
        rr_ptr += 2;
        memcpy(rr_ptr, pos - 3, 2);
        rr_ptr += 2;
        *rr_ptr = (unsigned char)len;
        rr_ptr += 1;
        memcpy(rr_ptr, pos, len);
        rr_ptr += len;
        memset(rr_ptr, 0, 1);
        rr_ptr++;
    }
    else
    {
        *((unsigned short*)rr_ptr) = htons(len);
        rr_ptr += 2;
        memcpy(rr_ptr, pos - 1, len + 1);
        rr_ptr += (len + 1);
    }
}

void setAddRR()
{
    rr_ptr = getMessage(&header, &query, dnsmessage, &len_header_query);
    rr_ptr = getRR(rr, &header, rr_ptr);
    rr_ptr++;
    int i, j;
    for(j = 0; j < header.answerNum; j++)
    {
        if(rr[i].type == 15)//找到 MX 对应 data 对应的 IP 地址
        {
            unsigned char temp_rr[256];
```

```
    unsigned char type;//记录第二个空格后的字符，也就是 RR 类型的首字母
    FILE *fp;
    fp = fopen(filename, "r");
    if(fp == NULL)
    {
       printf("the file cannot be opened: %d", errno);
       exit(0);
    }
    while(fgets(temp_rr, sizeof(temp_rr), fp) != NULL)//逐行查询
    {
       unsigned char rname[128];//记录一条资源记录中第一个空格前的部分
       memset(rname, 0, sizeof(rname));
       int len = strlen(temp_rr);
       for(i = 0; i < len; i++)
       {
          if(temp_rr[i] == ' ')
             break;
       }
       memcpy(rname, temp_rr, i);
       int numofspace = 0;
       for(i = 0; i < len; i++)
       {
          if(temp_rr[i] == ' ')
             numofspace++;
          if(temp_rr[i] == ' ' && numofspace == 2)
             break;
       }
       type = temp_rr[i + 1];
       if(containStr(rr[j].rdata, rname, type) == 1)
       {
          addRR(temp_rr, rname);
          unsigned char* ptr = dnsmessage;
          ptr += 6;
          /*因为添加 additional rr 也是用的添加 RR 的函数，所以
          需要报头的资源记录数减 1，然后附加资源记录数加 1*/
          *((unsigned short*)ptr) = htons(htons(*((unsigned short*)ptr)) - 1);
          ptr += 4;
          *((unsigned short*)ptr) = htons(htons(*((unsigned short*)ptr)) + 1);
       }
       memset(temp_rr, 0, sizeof(temp_rr));
    }
    err = fclose(fp);
    if(err == EOF)
    {
       printf("The file close failed: %d", errno);
       exit(0);
    }
    break;
}
```

```
    }
}

void recvfromSvr(int flag)
{
        memset(dnsmessage, 0, 1024);
        switch(flag)
        {
                case 0:
                {
                        struct sockaddr_in addr;
                int len = sizeof(addr);
                err = recvfrom(socketudp, dnsmessage, sizeof(dnsmessage), 0, (struct
sockaddr*)&addr, &len);
                        break;
                }
                case 1:
                {
                        int len = sizeof(clientAddr);
                err = recvfrom(socketudp, dnsmessage, sizeof(dnsmessage), 0, (struct
sockaddr*)&clientAddr, &len);
                        break;
                }
        }
   if(err <= 0)//等于 0 时表示连接已终止
   {
      printf("UDP socket receive failed: %d\n", errno);
      exit(0);
   }
   int i;
}

void sendtoSvr(const unsigned char* svr, int flag)
{
        switch(flag)
        {
                case 0:
                {
        unsigned char* ptr = dnsmessage;
        ptr += 2;
        if (*((unsigned short*)ptr) == htons(0x8080))
        {
           *((unsigned short*)ptr) = htons(0x0080);
        }
        else if(*((unsigned short*)ptr) == htons(0x8180))
        {
           *((unsigned short*)ptr) = htons(0x0180);
        }
                        struct sockaddr_in destSvr;
                    memset(&destSvr, 0, sizeof(destSvr));
```

```
                    destSvr.sin_family = AF_INET;
                    destSvr.sin_port = htons(PORT);
                    destSvr.sin_addr.s_addr = inet_addr(svr);
                    int len = sizeof(dnsmessage);
                    err = sendto(socketudp, dnsmessage, len, 0, (struct
sockaddr*)&destSvr, sizeof(struct sockaddr));
                        break;
                }
                case 1:
                {
          unsigned char* ptr = dnsmessage;
          ptr += 2;
          if (*((unsigned short*)ptr) == htons(0x0080))
          {
             *((unsigned short*)ptr) = htons(0x8080);
          }
          else if(*((unsigned short*)ptr) == htons(0x0180))
          {
             *((unsigned short*)ptr) = htons(0x8180);
          }
                        err = sendto(socketudp, dnsmessage, sizeof(dnsmessage), 0,
(struct sockaddr*)&clientAddr, sizeof(struct sockaddr));
                }
          }
          if(err <= 0)
    {
       printf("send question to next dns failed: %d\n", errno);
       exit(0);
    }
}

void iterantion()
{
   printf("\nITERATION\n");
   sendtoSvr("", 1);
}

void recursion()
{
   printf("\nRECURSION\n");
   rr_ptr = getRR(rr, &header, get_rr_ptr);
   int i;
   for(i = 0; i < header.answerNum; i++)
   {
      if(rr[i].type == 2)
      {
         sendtoSvr(rr[i].rdata, 0);
         recvfromSvr(0);
         sendtoSvr("", 1);
      }
```

```
        else//如果查询类型不为 A 表示已经查到结果
        {
            sendtoSvr("", 1);
        }
    }
}

void process()
{
        while(1)
    {
        recvfromSvr(1);
        setRR();
        setAddRR();
        /*判断使用何种解析方式*/
        if(header.tag == 0x0080)
        {
            iterantion();
        }
        else if(header.tag == 0x0180)
        {
            recursion();
        }
    }
}
int main()
{
    initSocket(OTHER, "other.txt");
    process();
    return 0;
}
```

nation.c

```
struct DNSHeader header;
struct DNSQuery query;
struct DNSRR rr[10];
struct sockaddr_in clientAddr;          //记录 UDP 传输中的客户端地址
unsigned char dnsmessage[1024];//报文
unsigned char* rr_ptr;                  //记录 rr 的位置
unsigned char* get_rr_ptr;              //用于 getRR 的指针
char* filename;                                 //文件名
int socketudp;                                  //套接字标识符
int err;                                        //记录返回值
int len_header_query = 0;       //记录报文中资源记录之前部分的长度

void initSocket(const char* svr, const char* _filename)
{
        filename = _filename;
```

```
    //初始化 UDP 套接字
    socketudp = socket(AF_INET , SOCK_DGRAM , IPPROTO_UDP);
    struct sockaddr_in addr;
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(PORT);
    addr.sin_addr.s_addr = inet_addr(svr);
    err = bind(socketudp, (struct sockaddr*)&addr, sizeof(struct sockaddr));
    if(err < 0)
    {
        printf("bind failed: %d\n",errno);
        exit(0);
    }
}

int containStr(const unsigned char* dname, const unsigned char* rname, const
unsigned char type)
{
    int len1 = strlen(dname);
    int len2 = strlen(rname);
    int i = len1 - 1, j = len2 - 1;
    if(type == 'N')
    {
        for(;; i--,j--) //自后向前遍历
        {
            if(j < 0)//rname 读完,表示每一位都匹配上
            {
                return 1;
            }
            if(dname[i] != rname[j])//某一位未匹配上
                return -1;
        }
    }
    else
    {
        if(strcmp(dname, rname) == 0)
        {
            return 1;
        }
        return -1;
    }
}

void setRR()
{
    unsigned char temp_rr[256];
    rr_ptr = getMessage(&header, &query, dnsmessage, &len_header_query);
    get_rr_ptr = rr_ptr;
    memset(rr_ptr, 0, sizeof(dnsmessage) - len_header_query);//清空报文中的 rr 部分
```

```
unsigned char* ptr = dnsmessage;
ptr += 6;
*((unsigned short*)ptr) = 0;//报头的资源记录数置零
ptr += 2;
*((unsigned short*)ptr) = 0;
FILE *fp;
fp = fopen(filename, "r");
if(fp == NULL)
{
   printf("the file cannot be opened: %d\n", errno);
   exit(0);
}
unsigned char dname[128];
memset(dname, 0, sizeof(dname));
unsigned char* temp_ptr = query.name;
int flag, i, num = 0;
for(;;)//将 query.name 转换成标准的域名格式
{
   flag = (int)temp_ptr[0];
   for(i = 0; i < flag; i++)
   {
      dname[i + num] = temp_ptr[i + 1];
   }
   temp_ptr += (flag + 1);
   if((int)temp_ptr[0] == 0)
      break;
   dname[flag + num] = '.';
   num += (flag + 1);
}
while(fgets(temp_rr, sizeof(temp_rr), fp) != NULL)//逐行查询
{
   unsigned char rname[128];//记录一条资源记录中第一个空格前的部分
   unsigned char type;//记录第二个空格后的字符，也就是 RR 类型的首字母
   memset(rname, 0, sizeof(rname));
   int len = strlen(temp_rr);
   for(i = 0; i < len; i++)
   {
      if(temp_rr[i] == ' ')
         break;
   }
   memcpy(rname, temp_rr, i);
   int numofspace = 0;
   for(i = 0; i < len; i++)
   {
      if(temp_rr[i] == ' ')
         numofspace++;
      if(temp_rr[i] == ' ' && numofspace == 2)
         break;
   }
```

```
        type = temp_rr[i + 1];
        if(containStr(dname, rname, type) == 1)
        {
            addRR(temp_rr, rname);
        }
        memset(temp_rr, 0, sizeof(temp_rr));
    }
    err = fclose(fp);
    if(err == EOF)
    {
        printf("The file close failed: %d\n", errno);
        exit(0);
    }
}

void addRR(const unsigned char* str, const unsigned char* rname)
{
    unsigned char buf[128];
    unsigned char* ptr = dnsmessage;
    ptr += 6;
    *((unsigned short*)ptr) = htons(htons(*((unsigned short*)ptr)) + 1);//报头的资源
记录数加 1
    ptr = buf;
    char *pos;
    int n, len = 0;//len 记录域名的长度
    pos = (char*)rname;
    /*将域名存到 buf 中，buf 中存储每个域的长度和内容
    比如当前域是 edu.cn，存到 buf 中就变成了 3edu2cn0
    ,0 表示结尾*/
    for(;;)
    {
        n = strlen(pos) - (strstr(pos , ".") ? strlen(strstr(pos , ".")) : 0);
        *ptr ++ = (unsigned char)n;
        memcpy(ptr , pos , n);
        len += n + 1;
        ptr += n;
        if(!strstr(pos , "."))
        {
            *ptr = (unsigned char)0;
            ptr ++;
            len += 1;
            break;
        }
        pos += n + 1;
    }
    memcpy(rr_ptr, buf, len);
    rr_ptr += len;
    pos = (char*)str;
    pos += (len + 2);
```

```
int flag = 0;
/*因为只考虑 A,NS,MX,CNAME 四种查询类型
，所以只做了匹配第一个字母的简单处理*/
switch(pos[0])
{
case'A':
{
   *((unsigned short*)rr_ptr) = htons(1);
   rr_ptr += 2;
   pos += 2;
   flag = 1;
   break;
}
case'N':
{
      unsigned char* _ptr = dnsmessage;
   _ptr += 6;
   *((unsigned short*)_ptr) = htons(htons(*((unsigned short*)_ptr)) - 1);
   _ptr += 2;
   *((unsigned short*)_ptr) = htons(htons(*((unsigned short*)_ptr)) + 1);
   *((unsigned short*)rr_ptr) = htons(2);
   rr_ptr += 2;
   pos += 3;
   break;
}
case'C':
{
   *((unsigned short*)rr_ptr) = htons(5);
   rr_ptr += 2;
   pos += 6;
   break;
}
case'M':
{
   *((unsigned short*)rr_ptr) = htons(15);
   rr_ptr += 2;
   pos += 3;
   flag = 2;
   break;
}
}
*((unsigned short*)rr_ptr) = htons(1);
rr_ptr += 2;
*((unsigned short*)rr_ptr) = htonl(0);
rr_ptr += 4;
len = strlen(pos);
len = len - 2;//len - 2 是因为从文件中读取的字符串最后两位是回车加换行
if (flag == 1)
{
   *((unsigned short*)rr_ptr) = htons(4);
```

```
        rr_ptr += 2;
        struct in_addr addr;
        char ip[32];
        memset(ip, 0, sizeof(ip));
        memcpy(ip, pos, len);
        inet_aton(ip, &addr);
        *((unsigned long*)rr_ptr) = addr.s_addr;
        rr_ptr += 4;
    }
    else if(flag == 2)
    {
            *((unsigned short*)rr_ptr) = htons(len);
        rr_ptr += 2;
        memcpy(rr_ptr, pos - 3, 2);
        rr_ptr += 2;
        *rr_ptr = (unsigned char)len;
        rr_ptr += 1;
        memcpy(rr_ptr, pos, len);
        rr_ptr += len;
        memset(rr_ptr, 0, 1);
        rr_ptr++;
    }
    else
    {
        *((unsigned short*)rr_ptr) = htons(len);
        rr_ptr += 2;
        memcpy(rr_ptr, pos - 1, len + 1);
        rr_ptr += (len + 1);
    }
}

void setAddRR()
{
    rr_ptr = getMessage(&header, &query, dnsmessage, &len_header_query);
    rr_ptr = getRR(rr, &header, rr_ptr);
    rr_ptr++;
    int i, j;
    for(j = 0; j < header.answerNum; j++)
    {
        if(rr[i].type == 15)//找到 MX 对应 data 对应的 IP 地址
        {
            unsigned char temp_rr[256];
            unsigned char type;//记录第二个空格后的字符，也就是 RR 类型的首字母
            FILE *fp;
            fp = fopen(filename, "r");
            if(fp == NULL)
            {
                printf("the file cannot be opened: %d", errno);
                exit(0);
            }
```

```
        while(fgets(temp_rr, sizeof(temp_rr), fp) != NULL)//逐行查询
        {
            unsigned char rname[128];//记录一条资源记录中第一个空格前的部分
            memset(rname, 0, sizeof(rname));
            int len = strlen(temp_rr);
            for(i = 0; i < len; i++)
            {
                if(temp_rr[i] == ' ')
                    break;
            }
            memcpy(rname, temp_rr, i);
            int numofspace = 0;
            for(i = 0; i < len; i++)
            {
                if(temp_rr[i] == ' ')
                    numofspace++;
                if(temp_rr[i] == ' ' && numofspace == 2)
                    break;
            }
            type = temp_rr[i + 1];
            if(containStr(rr[j].rdata, rname, type) == 1)
            {
                addRR(temp_rr, rname);
                unsigned char* ptr = dnsmessage;
                ptr += 6;
                /*因为添加 additional rr 也是用的添加 RR 的函数，所以
                需要报头的资源记录数减 1，然后附加资源记录数加 1*/
                *((unsigned short*)ptr) = htons(htons(*((unsigned short*)ptr)) - 1);
                ptr += 4;
                *((unsigned short*)ptr) = htons(htons(*((unsigned short*)ptr)) + 1);
            }
            memset(temp_rr, 0, sizeof(temp_rr));
        }
        err = fclose(fp);
        if(err == EOF)
        {
            printf("The file close failed: %d", errno);
            exit(0);
        }
        break;
    }
  }
}

void recvfromSvr(int flag)
{
        memset(dnsmessage, 0, 1024);
        switch(flag)
        {
                case 0:
```

```
                {
                        struct sockaddr_in addr;
                int len = sizeof(addr);
                err = recvfrom(socketudp, dnsmessage, sizeof(dnsmessage), 0, (struct
sockaddr*)&addr, &len);
                        break;
                }
                case 1:
                {
                        int len = sizeof(clientAddr);
                err = recvfrom(socketudp, dnsmessage, sizeof(dnsmessage), 0, (struct
sockaddr*)&clientAddr, &len);
                        break;
                }
        }
   if(err <= 0)//等于 0 时表示连接已终止
   {
      printf("UDP socket receive failed: %d\n", errno);
      exit(0);
   }
   int i;
}

void sendtoSvr(const unsigned char* svr, int flag)
{
        switch(flag)
        {
                case 0:
                {
        unsigned char* ptr = dnsmessage;
        ptr += 2;
        if (*((unsigned short*)ptr) == htons(0x8080))
        {
           *((unsigned short*)ptr) = htons(0x0080);
        }
        else if(*((unsigned short*)ptr) == htons(0x8180))
        {
           *((unsigned short*)ptr) = htons(0x0180);
        }
                        struct sockaddr_in destSvr;
                   memset(&destSvr, 0, sizeof(destSvr));
                   destSvr.sin_family = AF_INET;
                   destSvr.sin_port = htons(PORT);
                   destSvr.sin_addr.s_addr = inet_addr(svr);
                   int len = sizeof(dnsmessage);
                   err = sendto(socketudp, dnsmessage, len, 0, (struct
sockaddr*)&destSvr, sizeof(struct sockaddr));
                        break;
                }
                case 1:
```

```
                {
            unsigned char* ptr = dnsmessage;
            ptr += 2;
            if (*((unsigned short*)ptr) == htons(0x0080))
            {
               *((unsigned short*)ptr) = htons(0x8080);
            }
            else if(*((unsigned short*)ptr) == htons(0x0180))
            {
               *((unsigned short*)ptr) = htons(0x8180);
            }
                                err = sendto(socketudp, dnsmessage, sizeof(dnsmessage), 0,
(struct sockaddr*)&clientAddr, sizeof(struct sockaddr));
                }
            }
            if(err <= 0)
   {
      printf("send question to next dns failed: %d\n", errno);
      exit(0);
   }
}

void iterantion()
{
   printf("\nITERATION\n");
   sendtoSvr("", 1);
}

void recursion()
{
   printf("\nRECURSION\n");
   rr_ptr = getRR(rr, &header, get_rr_ptr);
   int i;
   for(i = 0; i < header.answerNum; i++)
   {
      if(rr[i].type == 2)
      {
         sendtoSvr(rr[i].rdata, 0);
         recvfromSvr(0);
         sendtoSvr("", 1);
      }
      else//如果查询类型不为 A 表示已经查到结果
      {
         sendtoSvr("", 1);
      }
   }
}

void process()
{
```

```
        while(1)
    {
        recvfromSvr(1);
        setRR();
        setAddRR();
        /*判断使用何种解析方式*/
        if(header.tag == 0x0080)
        {
            iterantion();
        }
        else if(header.tag == 0x0180)
        {
            recursion();
        }
    }
}
int main()
{
    initSocket(NATION, "nation.txt");
    process();
    return 0;
}
```

gov.c

```
struct DNSHeader header;
struct DNSQuery query;
struct DNSRR rr[10];
struct sockaddr_in clientAddr;          //记录 UDP 传输中的客户端地址
unsigned char dnsmessage[1024];//报文
unsigned char* rr_ptr;                  //记录 rr 的位置
unsigned char* get_rr_ptr;              //用于 getRR 的指针
char* filename;                                 //文件名
int socketudp;                                  //套接字标识符
int err;                                        //记录返回值
int len_header_query = 0;       //记录报文中资源记录之前部分的长度

void initSocket(const char* svr, const char* _filename)
{
        filename = _filename;
    //初始化 UDP 套接字
    socketudp = socket(AF_INET , SOCK_DGRAM , IPPROTO_UDP);
    struct sockaddr_in addr;
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(PORT);
    addr.sin_addr.s_addr = inet_addr(svr);
    err = bind(socketudp, (struct sockaddr*)&addr, sizeof(struct sockaddr));
    if(err < 0)
```

```
    {
        printf("bind failed: %d\n",errno);
        exit(0);
    }
}

int containStr(const unsigned char* dname, const unsigned char* rname, const
unsigned char type)
{
    int len1 = strlen(dname);
    int len2 = strlen(rname);
    int i = len1 - 1, j = len2 - 1;
    if(type == 'N')
    {
        for(;; i--,j--) //自后向前遍历
        {
            if(j < 0)//rname 读完,表示每一位都匹配上
            {
                return 1;
            }
            if(dname[i] != rname[j])//某一位未匹配上
                return -1;
        }
    }
    else
    {
        if(strcmp(dname, rname) == 0)
        {
            return 1;
        }
        return -1;
    }
}

void setRR()
{
    unsigned char temp_rr[256];
    rr_ptr = getMessage(&header, &query, dnsmessage, &len_header_query);
    get_rr_ptr = rr_ptr;
    memset(rr_ptr, 0, sizeof(dnsmessage) - len_header_query);//清空报文中的 rr 部分
    unsigned char* ptr = dnsmessage;
    ptr += 6;
    *((unsigned short*)ptr) = 0;//报头的资源记录数置零
    ptr += 2;
    *((unsigned short*)ptr) = 0;
    FILE *fp;
    fp = fopen(filename, "r");
    if(fp == NULL)
    {
```

```
    printf("the file cannot be opened: %d\n", errno);
    exit(0);
}
unsigned char dname[128];
memset(dname, 0, sizeof(dname));
unsigned char* temp_ptr = query.name;
int flag, i, num = 0;
for(;;)//将 query.name 转换成标准的域名格式
{
    flag = (int)temp_ptr[0];
    for(i = 0; i < flag; i++)
    {
        dname[i + num] = temp_ptr[i + 1];
    }
    temp_ptr += (flag + 1);
    if((int)temp_ptr[0] == 0)
        break;
    dname[flag + num] = '.';
    num += (flag + 1);
}
while(fgets(temp_rr, sizeof(temp_rr), fp) != NULL)//逐行查询
{
    unsigned char rname[128];//记录一条资源记录中第一个空格前的部分
    unsigned char type;//记录第二个空格后的字符，也就是 RR 类型的首字母
    memset(rname, 0, sizeof(rname));
    int len = strlen(temp_rr);
    for(i = 0; i < len; i++)
    {
        if(temp_rr[i] == ' ')
            break;
    }
    memcpy(rname, temp_rr, i);
    int numofspace = 0;
    for(i = 0; i < len; i++)
    {
        if(temp_rr[i] == ' ')
            numofspace++;
        if(temp_rr[i] == ' ' && numofspace == 2)
            break;
    }
    type = temp_rr[i + 1];
    if(containStr(dname, rname, type) == 1)
    {
        addRR(temp_rr, rname);
    }
    memset(temp_rr, 0, sizeof(temp_rr));
}
err = fclose(fp);
if(err == EOF)
{
```

```
        printf("The file close failed: %d\n", errno);
        exit(0);
    }
}

void addRR(const unsigned char* str, const unsigned char* rname)
{
    unsigned char buf[128];
    unsigned char* ptr = dnsmessage;
    ptr += 6;
    *((unsigned short*)ptr) = htons(htons(*((unsigned short*)ptr)) + 1);//报头的资源
记录数加 1
    ptr = buf;
    char *pos;
    int n, len = 0;//len 记录域名的长度
    pos = (char*)rname;
    /*将域名存到 buf 中，buf 中存储每个域的长度和内容
    比如当前域是 edu.cn，存到 buf 中就变成了 3edu2cn0
    ,0 表示结尾*/
    for(;;)
    {
        n = strlen(pos) - (strstr(pos , ".") ? strlen(strstr(pos , ".")) : 0);
        *ptr ++ = (unsigned char)n;
        memcpy(ptr , pos , n);
        len += n + 1;
        ptr += n;
        if(!strstr(pos , "."))
        {
            *ptr = (unsigned char)0;
            ptr ++;
            len += 1;
            break;
        }
        pos += n + 1;
    }
    memcpy(rr_ptr, buf, len);
    rr_ptr += len;
    pos = (char*)str;
    pos += (len + 2);
    int flag = 0;
    /*因为只考虑 A,NS,MX,CNAME 四种查询类型
    ，所以只做了匹配第一个字母的简单处理*/
    switch(pos[0])
    {
    case'A':
    {
        *((unsigned short*)rr_ptr) = htons(1);
        rr_ptr += 2;
        pos += 2;
```

```
    flag = 1;
    break;
}
case'N':
{
        unsigned char* _ptr = dnsmessage;
    _ptr += 6;
    *((unsigned short*)_ptr) = htons(htons(*((unsigned short*)_ptr)) - 1);
    _ptr += 2;
    *((unsigned short*)_ptr) = htons(htons(*((unsigned short*)_ptr)) + 1);
    *((unsigned short*)rr_ptr) = htons(2);
    rr_ptr += 2;
    pos += 3;
    break;
}
case'C':
{
    *((unsigned short*)rr_ptr) = htons(5);
    rr_ptr += 2;
    pos += 6;
    break;
}
case'M':
{
    *((unsigned short*)rr_ptr) = htons(15);
    rr_ptr += 2;
    pos += 3;
    flag = 2;
    break;
}
}
*((unsigned short*)rr_ptr) = htons(1);
rr_ptr += 2;
*((unsigned short*)rr_ptr) = htonl(0);
rr_ptr += 4;
len = strlen(pos);
len = len - 2;//len - 2 是因为从文件中读取的字符串最后两位是回车加换行
if (flag == 1)
{
    *((unsigned short*)rr_ptr) = htons(4);
    rr_ptr += 2;
    struct in_addr addr;
    char ip[32];
    memset(ip, 0, sizeof(ip));
    memcpy(ip, pos, len);
    inet_aton(ip, &addr);
    *((unsigned long*)rr_ptr) = addr.s_addr;
    rr_ptr += 4;
}
else if(flag == 2)
```

```
    {
        *((unsigned short*)rr_ptr) = htons(len);
      rr_ptr += 2;
      memcpy(rr_ptr, pos - 3, 2);
      rr_ptr += 2;
      *rr_ptr = (unsigned char)len;
      rr_ptr += 1;
      memcpy(rr_ptr, pos, len);
      rr_ptr += len;
      memset(rr_ptr, 0, 1);
      rr_ptr++;
    }
    else
    {
      *((unsigned short*)rr_ptr) = htons(len);
      rr_ptr += 2;
      memcpy(rr_ptr, pos - 1, len + 1);
      rr_ptr += (len + 1);
    }
}

void setAddRR()
{
    rr_ptr = getMessage(&header, &query, dnsmessage, &len_header_query);
    rr_ptr = getRR(rr, &header, rr_ptr);
    rr_ptr++;
    int i, j;
    for(j = 0; j < header.answerNum; j++)
    {
      if(rr[i].type == 15)//找到 MX 对应 data 对应的 IP 地址
      {
        unsigned char temp_rr[256];
        unsigned char type;//记录第二个空格后的字符，也就是 RR 类型的首字母
        FILE *fp;
        fp = fopen(filename, "r");
        if(fp == NULL)
        {
          printf("the file cannot be opened: %d", errno);
          exit(0);
        }
        while(fgets(temp_rr, sizeof(temp_rr), fp) != NULL)//逐行查询
        {
          unsigned char rname[128];//记录一条资源记录中第一个空格前的部分
          memset(rname, 0, sizeof(rname));
          int len = strlen(temp_rr);
          for(i = 0; i < len; i++)
          {
            if(temp_rr[i] == ' ')
              break;
          }
```

```
                memcpy(rname, temp_rr, i);
                int numofspace = 0;
                for(i = 0; i < len; i++)
                {
                    if(temp_rr[i] == ' ')
                        numofspace++;
                    if(temp_rr[i] == ' ' && numofspace == 2)
                        break;
                }
                type = temp_rr[i + 1];
                if(containStr(rr[j].rdata, rname, type) == 1)
                {
                    addRR(temp_rr, rname);
                    unsigned char* ptr = dnsmessage;
                    ptr += 6;
                    /*因为添加 additional rr 也是用的添加 RR 的函数，所以
                    需要报头的资源记录数减 1，然后附加资源记录数加 1*/
                    *((unsigned short*)ptr) = htons(htons(*((unsigned short*)ptr)) - 1);
                    ptr += 4;
                    *((unsigned short*)ptr) = htons(htons(*((unsigned short*)ptr)) + 1);
                }
                memset(temp_rr, 0, sizeof(temp_rr));
            }
            err = fclose(fp);
            if(err == EOF)
            {
                printf("The file close failed: %d", errno);
                exit(0);
            }
            break;
        }
    }
}

void recvfromSvr(int flag)
{
        memset(dnsmessage, 0, 1024);
        switch(flag)
        {
                case 0:
                {
                        struct sockaddr_in addr;
                int len = sizeof(addr);
                err = recvfrom(socketudp, dnsmessage, sizeof(dnsmessage), 0, (struct
sockaddr*)&addr, &len);
                        break;
                }
                case 1:
                {
                        int len = sizeof(clientAddr);
```

```
                err = recvfrom(socketudp, dnsmessage, sizeof(dnsmessage), 0, (struct
sockaddr*)&clientAddr, &len);
                        break;
                }
        }
   if(err <= 0)//等于 0 时表示连接已终止
   {
      printf("UDP socket receive failed: %d\n", errno);
      exit(0);
   }
   int i;
}

void sendtoSvr(const unsigned char* svr, int flag)
{
        switch(flag)
        {
                case 0:
                {
        unsigned char* ptr = dnsmessage;
        ptr += 2;
        if (*((unsigned short*)ptr) == htons(0x8080))
        {
           *((unsigned short*)ptr) = htons(0x0080);
        }
        else if(*((unsigned short*)ptr) == htons(0x8180))
        {
           *((unsigned short*)ptr) = htons(0x0180);
        }
                        struct sockaddr_in destSvr;
                memset(&destSvr, 0, sizeof(destSvr));
                destSvr.sin_family = AF_INET;
                destSvr.sin_port = htons(PORT);
                destSvr.sin_addr.s_addr = inet_addr(svr);
                int len = sizeof(dnsmessage);
                err = sendto(socketudp, dnsmessage, len, 0, (struct
sockaddr*)&destSvr, sizeof(struct sockaddr));
                        break;
                }
                case 1:
                {
        unsigned char* ptr = dnsmessage;
        ptr += 2;
        if (*((unsigned short*)ptr) == htons(0x0080))
        {
           *((unsigned short*)ptr) = htons(0x8080);
        }
        else if(*((unsigned short*)ptr) == htons(0x0180))
        {
           *((unsigned short*)ptr) = htons(0x8180);
```

```
        }
                        err = sendto(socketudp, dnsmessage, sizeof(dnsmessage), 0,
(struct sockaddr*)&clientAddr, sizeof(struct sockaddr));
                }
        }
        if(err <= 0)
    {
        printf("send question to next dns failed: %d\n", errno);
        exit(0);
    }
}

void iterantion()
{
    printf("\nITERATION\n");
    sendtoSvr("", 1);
}

void recursion()
{
    printf("\nRECURSION\n");
    rr_ptr = getRR(rr, &header, get_rr_ptr);
    int i;
    for(i = 0; i < header.answerNum; i++)
    {
        if(rr[i].type == 2)
        {
            sendtoSvr(rr[i].rdata, 0);
            recvfromSvr(0);
            sendtoSvr("", 1);
        }
        else//如果查询类型不为 A 表示已经查到结果
        {
            sendtoSvr("", 1);
        }
    }
}

void process()
{
        while(1)
    {
        recvfromSvr(1);
        setRR();
        setAddRR();
        /*判断使用何种解析方式*/
        if(header.tag == 0x0080)
        {
            iterantion();
        }
```

```
        else if(header.tag == 0x0180)
        {
            recursion();
        }
    }
}
int main()
{
    initSocket(GOV, "gov.txt");
    process();
    return 0;
}
```

education.c

```
struct DNSHeader header;
struct DNSQuery query;
struct DNSRR rr[10];
struct sockaddr_in clientAddr;          //记录 UDP 传输中的客户端地址
unsigned char dnsmessage[1024];//报文
unsigned char* rr_ptr;                  //记录 rr 的位置
unsigned char* get_rr_ptr;              //用于 getRR 的指针
char* filename;                                 //文件名
int socketudp;                                  //套接字标识符
int err;                                        //记录返回值
int len_header_query = 0;       //记录报文中资源记录之前部分的长度

void initSocket(const char* svr, const char* _filename)
{
        filename = _filename;
    //初始化 UDP 套接字
    socketudp = socket(AF_INET , SOCK_DGRAM , IPPROTO_UDP);
    struct sockaddr_in addr;
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(PORT);
    addr.sin_addr.s_addr = inet_addr(svr);
    err = bind(socketudp, (struct sockaddr*)&addr, sizeof(struct sockaddr));
    if(err < 0)
    {
        printf("bind failed: %d\n",errno);
        exit(0);
    }
}

int containStr(const unsigned char* dname, const unsigned char* rname, const
unsigned char type)
{
    int len1 = strlen(dname);
    int len2 = strlen(rname);
```

```
    int i = len1 - 1, j = len2 - 1;
    if(type == 'N')
    {
        for(;; i--,j--) //自后向前遍历
        {
            if(j < 0)//rname 读完,表示每一位都匹配上
            {
                return 1;
            }
            if(dname[i] != rname[j])//某一位未匹配上
                return -1;
        }
    }
    else
    {
        if(strcmp(dname, rname) == 0)
        {
            return 1;
        }
        return -1;
    }
}

void setRR()
{
    unsigned char temp_rr[256];
    rr_ptr = getMessage(&header, &query, dnsmessage, &len_header_query);
    get_rr_ptr = rr_ptr;
    memset(rr_ptr, 0, sizeof(dnsmessage) - len_header_query);//清空报文中的 rr 部分
    unsigned char* ptr = dnsmessage;
    ptr += 6;
    *((unsigned short*)ptr) = 0;//报头的资源记录数置零
    ptr += 2;
    *((unsigned short*)ptr) = 0;
    FILE *fp;
    fp = fopen(filename, "r");
    if(fp == NULL)
    {
        printf("the file cannot be opened: %d\n", errno);
        exit(0);
    }
    unsigned char dname[128];
    memset(dname, 0, sizeof(dname));
    unsigned char* temp_ptr = query.name;
    int flag, i, num = 0;
    for(;;)//将 query.name 转换成标准的域名格式
    {
        flag = (int)temp_ptr[0];
        for(i = 0; i < flag; i++)
```

```
        {
            dname[i + num] = temp_ptr[i + 1];
        }
        temp_ptr += (flag + 1);
        if((int)temp_ptr[0] == 0)
            break;
        dname[flag + num] = '.';
        num += (flag + 1);
    }
    while(fgets(temp_rr, sizeof(temp_rr), fp) != NULL)//逐行查询
    {
        unsigned char rname[128];//记录一条资源记录中第一个空格前的部分
        unsigned char type;//记录第二个空格后的字符，也就是 RR 类型的首字母
        memset(rname, 0, sizeof(rname));
        int len = strlen(temp_rr);
        for(i = 0; i < len; i++)
        {
            if(temp_rr[i] == ' ')
                break;
        }
        memcpy(rname, temp_rr, i);
        int numofspace = 0;
        for(i = 0; i < len; i++)
        {
            if(temp_rr[i] == ' ')
                numofspace++;
            if(temp_rr[i] == ' ' && numofspace == 2)
                break;
        }
        type = temp_rr[i + 1];
        if(containStr(dname, rname, type) == 1)
        {
            addRR(temp_rr, rname);
        }
        memset(temp_rr, 0, sizeof(temp_rr));
    }
    err = fclose(fp);
    if(err == EOF)
    {
        printf("The file close failed: %d\n", errno);
        exit(0);
    }
}

void addRR(const unsigned char* str, const unsigned char* rname)
{
    unsigned char buf[128];
    unsigned char* ptr = dnsmessage;
    ptr += 6;
```

```
   *((unsigned short*)ptr) = htons(htons(*((unsigned short*)ptr)) + 1);//报头的资源
记录数加 1
   ptr = buf;
   char *pos;
   int n, len = 0;//len 记录域名的长度
   pos = (char*)rname;
   /*将域名存到 buf 中，buf 中存储每个域的长度和内容
   比如当前域是 edu.cn，存到 buf 中就变成了 3edu2cn0
   ,0 表示结尾*/
   for(;;)
   {
      n = strlen(pos) - (strstr(pos , ".") ? strlen(strstr(pos , ".")) : 0);
      *ptr ++ = (unsigned char)n;
      memcpy(ptr , pos , n);
      len += n + 1;
      ptr += n;
      if(!strstr(pos , "."))
      {
         *ptr = (unsigned char)0;
         ptr ++;
         len += 1;
         break;
      }
      pos += n + 1;
   }
   memcpy(rr_ptr, buf, len);
   rr_ptr += len;
   pos = (char*)str;
   pos += (len + 2);
   int flag = 0;
   /*因为只考虑 A,NS,MX,CNAME 四种查询类型
   ，所以只做了匹配第一个字母的简单处理*/
   switch(pos[0])
   {
   case'A':
   {
      *((unsigned short*)rr_ptr) = htons(1);
      rr_ptr += 2;
      pos += 2;
      flag = 1;
      break;
   }
   case'N':
   {
         unsigned char* _ptr = dnsmessage;
      _ptr += 6;
      *((unsigned short*)_ptr) = htons(htons(*((unsigned short*)_ptr)) - 1);
      _ptr += 2;
      *((unsigned short*)_ptr) = htons(htons(*((unsigned short*)_ptr)) + 1);
```

```
    *((unsigned short*)rr_ptr) = htons(2);
    rr_ptr += 2;
    pos += 3;
    break;
}
case'C':
{
    *((unsigned short*)rr_ptr) = htons(5);
    rr_ptr += 2;
    pos += 6;
    break;
}
case'M':
{
    *((unsigned short*)rr_ptr) = htons(15);
    rr_ptr += 2;
    pos += 3;
    flag = 2;
    break;
}
}
*((unsigned short*)rr_ptr) = htons(1);
rr_ptr += 2;
*((unsigned short*)rr_ptr) = htonl(0);
rr_ptr += 4;
len = strlen(pos);
len = len - 2;//len - 2 是因为从文件中读取的字符串最后两位是回车加换行
if (flag == 1)
{
    *((unsigned short*)rr_ptr) = htons(4);
    rr_ptr += 2;
    struct in_addr addr;
    char ip[32];
    memset(ip, 0, sizeof(ip));
    memcpy(ip, pos, len);
    inet_aton(ip, &addr);
    *((unsigned long*)rr_ptr) = addr.s_addr;
    rr_ptr += 4;
}
else if(flag == 2)
{
        *((unsigned short*)rr_ptr) = htons(len);
    rr_ptr += 2;
    memcpy(rr_ptr, pos - 3, 2);
    rr_ptr += 2;
    *rr_ptr = (unsigned char)len;
    rr_ptr += 1;
    memcpy(rr_ptr, pos, len);
    rr_ptr += len;
    memset(rr_ptr, 0, 1);
```

```
        rr_ptr++;
    }
    else
    {
        *((unsigned short*)rr_ptr) = htons(len);
        rr_ptr += 2;
        memcpy(rr_ptr, pos - 1, len + 1);
        rr_ptr += (len + 1);
    }
}

void setAddRR()
{
    rr_ptr = getMessage(&header, &query, dnsmessage, &len_header_query);
    rr_ptr = getRR(rr, &header, rr_ptr);
    rr_ptr++;
    int i, j;
    for(j = 0; j < header.answerNum; j++)
    {
        if(rr[i].type == 15)//找到 MX 对应 data 对应的 IP 地址
        {
            unsigned char temp_rr[256];
            unsigned char type;//记录第二个空格后的字符，也就是 RR 类型的首字母
            FILE *fp;
            fp = fopen(filename, "r");
            if(fp == NULL)
            {
                printf("the file cannot be opened: %d", errno);
                exit(0);
            }
            while(fgets(temp_rr, sizeof(temp_rr), fp) != NULL)//逐行查询
            {
                unsigned char rname[128];//记录一条资源记录中第一个空格前的部分
                memset(rname, 0, sizeof(rname));
                int len = strlen(temp_rr);
                for(i = 0; i < len; i++)
                {
                    if(temp_rr[i] == ' ')
                        break;
                }
                memcpy(rname, temp_rr, i);
                int numofspace = 0;
                for(i = 0; i < len; i++)
                {
                    if(temp_rr[i] == ' ')
                        numofspace++;
                    if(temp_rr[i] == ' ' && numofspace == 2)
                        break;
                }
                type = temp_rr[i + 1];
```

```
            if(containStr(rr[j].rdata, rname, type) == 1)
            {
               addRR(temp_rr, rname);
               unsigned char* ptr = dnsmessage;
               ptr += 6;
               /*因为添加 additional rr 也是用的添加 RR 的函数，所以
               需要报头的资源记录数减 1，然后附加资源记录数加 1*/
               *((unsigned short*)ptr) = htons(htons(*((unsigned short*)ptr)) - 1);
               ptr += 4;
               *((unsigned short*)ptr) = htons(htons(*((unsigned short*)ptr)) + 1);
            }
            memset(temp_rr, 0, sizeof(temp_rr));
         }
         err = fclose(fp);
         if(err == EOF)
         {
            printf("The file close failed: %d", errno);
            exit(0);
         }
         break;
      }
   }
}

void recvfromSvr(int flag)
{
         memset(dnsmessage, 0, 1024);
         switch(flag)
         {
                  case 0:
                  {
                           struct sockaddr_in addr;
                  int len = sizeof(addr);
                  err = recvfrom(socketudp, dnsmessage, sizeof(dnsmessage), 0, (struct
sockaddr*)&addr, &len);
                           break;
                  }
                  case 1:
                  {
                           int len = sizeof(clientAddr);
                  err = recvfrom(socketudp, dnsmessage, sizeof(dnsmessage), 0, (struct
sockaddr*)&clientAddr, &len);
                           break;
                  }
         }
   if(err <= 0)//等于 0 时表示连接已终止
   {
      printf("UDP socket receive failed: %d\n", errno);
      exit(0);
   }
```

```
   int i;
}

void sendtoSvr(const unsigned char* svr, int flag)
{
        switch(flag)
        {
                case 0:
                {
        unsigned char* ptr = dnsmessage;
        ptr += 2;
        if (*((unsigned short*)ptr) == htons(0x8080))
        {
           *((unsigned short*)ptr) = htons(0x0080);
        }
        else if(*((unsigned short*)ptr) == htons(0x8180))
        {
           *((unsigned short*)ptr) = htons(0x0180);
        }
                        struct sockaddr_in destSvr;
                    memset(&destSvr, 0, sizeof(destSvr));
                    destSvr.sin_family = AF_INET;
                    destSvr.sin_port = htons(PORT);
                    destSvr.sin_addr.s_addr = inet_addr(svr);
                    int len = sizeof(dnsmessage);
                    err = sendto(socketudp, dnsmessage, len, 0, (struct
sockaddr*)&destSvr, sizeof(struct sockaddr));
                        break;
                }
                case 1:
                {
        unsigned char* ptr = dnsmessage;
        ptr += 2;
        if (*((unsigned short*)ptr) == htons(0x0080))
        {
           *((unsigned short*)ptr) = htons(0x8080);
        }
        else if(*((unsigned short*)ptr) == htons(0x0180))
        {
           *((unsigned short*)ptr) = htons(0x8180);
        }
                        err = sendto(socketudp, dnsmessage, sizeof(dnsmessage), 0,
(struct sockaddr*)&clientAddr, sizeof(struct sockaddr));
                }
        }
        if(err <= 0)
   {
      printf("send question to next dns failed: %d\n", errno);
      exit(0);
   }
```

```
}

void iterantion()
{
   printf("\nITERATION\n");
   sendtoSvr("", 1);
}

void recursion()
{
   printf("\nRECURSION\n");
   rr_ptr = getRR(rr, &header, get_rr_ptr);
   int i;
   for(i = 0; i < header.answerNum; i++)
   {
      if(rr[i].type == 2)
      {
         sendtoSvr(rr[i].rdata, 0);
         recvfromSvr(0);
         sendtoSvr("", 1);
      }
      else//如果查询类型不为 A 表示已经查到结果
      {
         sendtoSvr("", 1);
      }
   }
}

void process()
{
         while(1)
   {
      recvfromSvr(1);
      setRR();
      setAddRR();
      /*判断使用何种解析方式*/
      if(header.tag == 0x0080)
      {
         iterantion();
      }
      else if(header.tag == 0x0180)
      {
         recursion();
      }
   }
}
int main()
{
   initSocket(EDU, "education.txt");
   process();
```

```
    return 0;
}
```