A REPORT

ON

DNS Client Application

By

Alladi Santhosh	2011A7PS086H
Himshi Bachchas	2011A7PS172H
Raghavender Sahdev	2011A7PS257H

At

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

HYDERABAD Campus

Contents:

Abstract

Our aim is to create a DNS client application where in client can perform a number of operations by sending messages to the internet. Based on a given domain name, it has to perform a number of functions such as retrieving all IP addresses, mail server records, host info, etc. The later part of the project deals with implementing iterative and recursive queries.

<u>Introduction</u>

DNS Basics

DNS stands for Domain Name System which is an essential part of any host connected to the internet. It is used by applications to map between host names and IP addresses. It is a distributed database used by TCP/IP applications as nothing on the internet is present at one single place. It acts like a protocol between client and servers which allows communication between them.

Working of DNS

DNS is accessed through a resolver which is run on Unix host. A resolver is a link between application and DNS server. 2 types of DNS models exist namely recursive and iterative. A request is sent to named server, which is then checked in the database, if it does not contain information, it will pass the query to the root server. The root servers contains the info about all the root servers. <u>Iterative</u> : the named server has the initial request, if not found there, the named server contacts the root server, the root server info about all the other named servers, if none of the named servers contain the info it will contact other root server and so on.

What are different DNS Commands, what options do they provide?

This is the simplest of the DNS commands. It is a quick way to determine the IP address of a hostname:

- Nslookup – it is used for querying the Domain Name System(DNS) to obtain domain name or IP address mapping or for any other specific DNS record
- Ping – Test connectivity
- Tracert – Trace IP address Route
- Netstat – displays the TCP/IP protocol sessions
- Route – Display Local Route
- Arp – Display reslved MAC addresses
- Hostname – doisplay computer name

- Netsetup.cpl – Network Setup wizard
- Control netconnections – Network connections
- Ipconfig – Internet protocol configuration

Which library functions are available?

gethostbyaddr();

gethostbyname();

dns_ip4(&out,&fqdn);

dns_ip4_packet(&out,buf,len);

dns_ip4_qualify(&out,&fqdn,&udn);

dns_name4(&out,ip);

dns_name_packet(&out,buf,len);

dns_name4_domain(q,ip);

dns_mx(&out,&fqdn);

dns_mx_packet(&out,buf,len);

dns_txt(&out,&fqdn);

dns_txt_packet(&out,buf,len);

Which system config file is used by DNS?

The DNS configuration files are stored in the /etc/bind directory. The primary configuration file is /etc/bind/named.conf. The include line specifies the filename which contains the DNS options. The directory line in the /etc/bind/named.conf.options file tells DNS where to look for files. All files BIND uses will be relative to this directory.

However, resolve.conf is used by the resolver.

DNS related software or online services?

- DNSCrypt - A tool for securing communications between a client and a DNS resolver.

- Open DNS - extends DNS adding features such as misspelling correction, phishing protection, and optional content filtering.
- BIND, Microsoft DNS, Dnsmasq , djbdns , Simple DNS Plus , NSD , Knot DNS, PowerDNS , MaraDNS , Nominum Authoritative Name Server , Nominum, Vantio , Posadis , Unbound , pdnsd , etc.

Project Design

DNS Message Format:

identification	flags
Number of questions	Number of answer RRs
Number of authority RRs	Number of additional RRs
Questions	
Answers	
Authority	
Additional information	

Fig. General format of DNS message

The above figure shows the overall format of a DNS message defined for both queries and responses.

The message has a fixed 12-byte header which is followed by four variable length fields.

The **identification** lets the client match responses to requests. It is set by client and returned by the server.

The **flag field** is 16-bit in size. It gives information such as :

The message is a query or response; opcode of the message; answer is authoritative or not; whether the message is truncated; iterative or recursive query; whether the servers provide recursion or not and a 4-bit return code specifying the type of error, if any. The flag field also contains a 3-bit field which must be 0.

The next four 16-bit fields specify the number of entries in the corresponding variable length fields. For a query, the number of questions is 1 or more(in general it is 1) and the other counts are 0. For a response the number of answers os atleast 1, with the remaining two counts 0 or more.

DNS Features:

The following are the DNS features:

I.Gets all IP addresses for a given domain name, and gets canonical name and aliases for a given IP address.

II. Gets all IPv6 addresses for a given domain name, and gets canonical name and aliases for a given IPv6 address.

III. Retrieves mail server records, Name Server records, host info, Start of Authority record.

IV. Handles multiple domain names or IP addresses.

V. Provides no. of tries option with same/different IP addresses.

VI. Provides timeout feature to stop querying.

VII. Provides debug information.

VIII. Writes every successful call to a log file.

IX. Implements both recursive and iterative queries.

X. Handles responses larger than 512 bytes.

Protocols used:

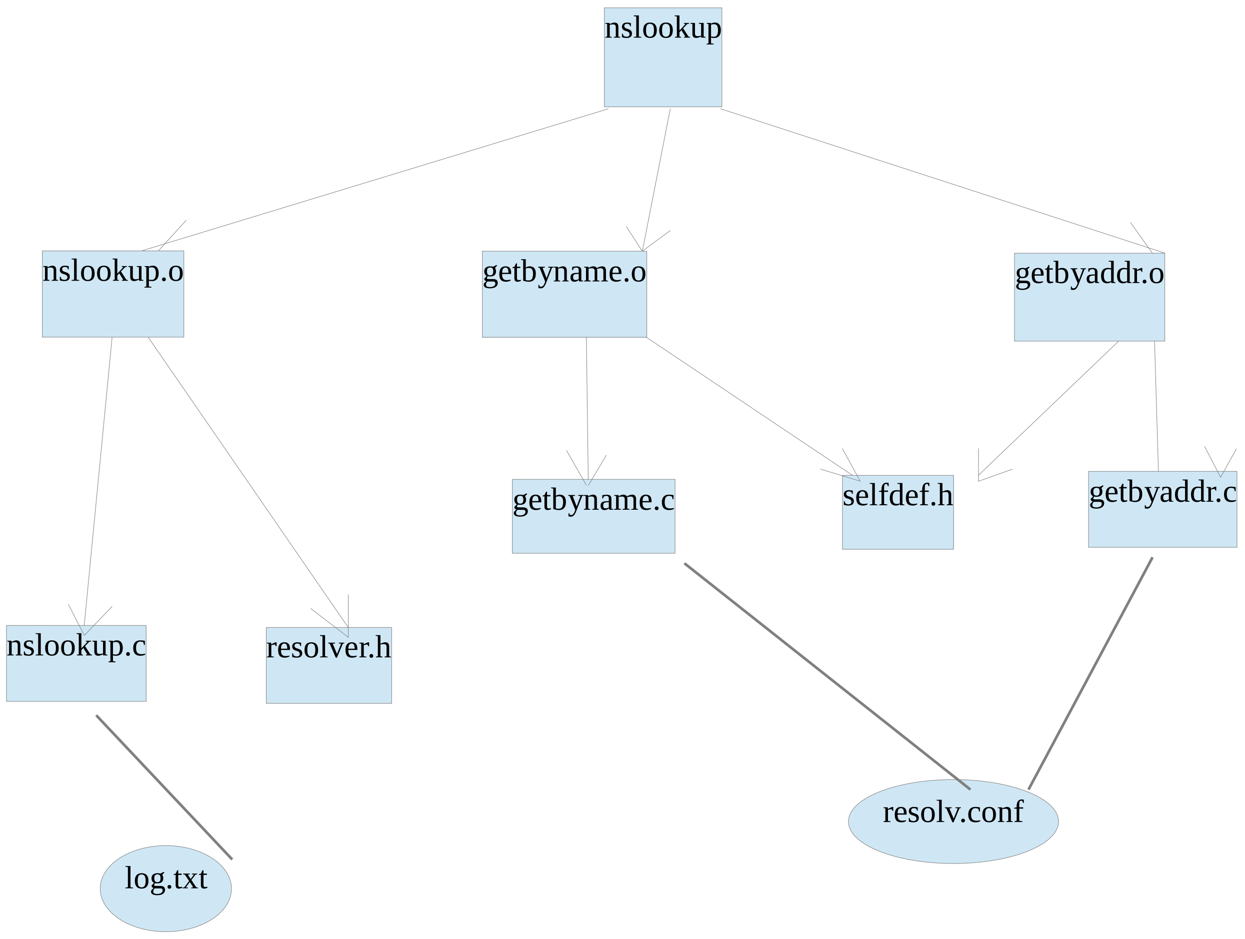
We have used the following protocols depending on the requirements:

1. UDP Protocol: At first, the resolver queries the DNS server using UDP protocol. But UDP has its limitation since it can't handle data of greater than 512 bytes in one go. This leads to the use of the next protocol.

2. TCP Protocol: If the response of DNS server is greater than 512 bytes in size, we establish reliable connection using TCP protocol for the query.

Dependency Graph:

We have the main dependency graph on the client application side.



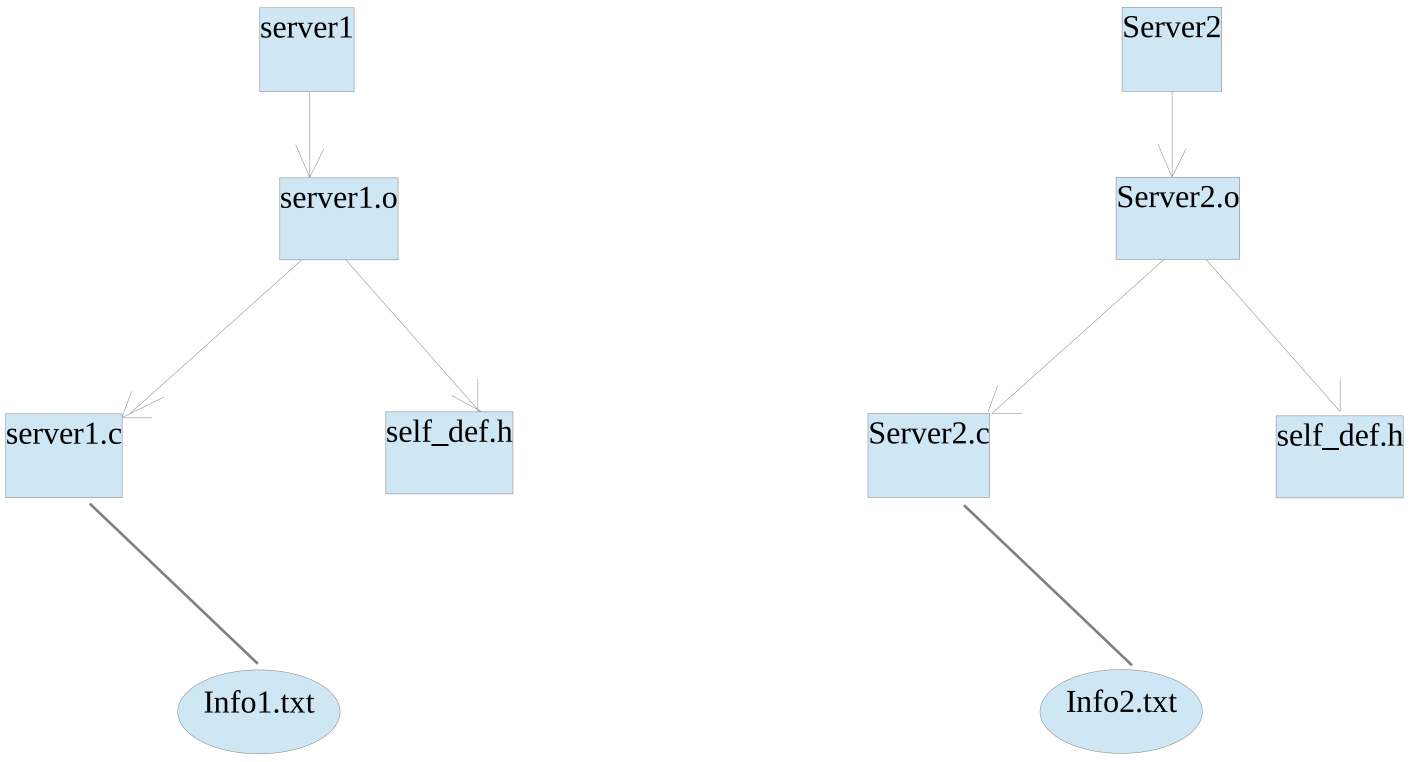
In the above dependency graph, we have the executable file named as ***nslookup***.

We have ***nslookup.c*** file which has the application code. It depends on ***resolver.h*** file which has the prototype of the two functions getbyname and getbyaddr as used by the resolver.

In the resolver, we have two files. ***getbyname.c*** is the implementation of ''*gethostbyname*'' function and ***getbyaddr.c*** is the implementation of ''*gethostbyaddr*'' function. Both of these files include the ***selfdef.h*** file which consists of definitions of structs used.

Apart from the C files, the nslookup.c file also requires *log.txt* to write successful logs. Similarly the getbyname and getbyaddr need the *resolv.conf* file for the details of name server to function properly. Hence, these dependencies are also shown with dashed lines.

We can have more dependency graphs for the server side. Here we show one example for 1st server.



Explanation of the code

Phase 1: First of all user can enter any number of domain names as command line arguments, user can also specify his/her choice regarding recursive/ iterative query, number of tries, wait for reply, etc. with the help of command line options. We have the following command line options: -R (this sets iterative query option, default is recursive), –r (number of tries, user enters only integer - assumption), -w (this is for the number of seconds to wait for response - integer assumption). The user can specify debug mode by simply entering –d. Number of tries 1, wait for replies- default value - 15 seconds, we are assuming that the user does not enter option after specifying domain names, the user will specify all options before entering any names.

Then with the help of argc the number of arguments, we look for number of domain names and in each iteration we call the function make _ domain name argument.

Make_call() function – In this we have used a structure DNS records, it sores all the info that we need to retrieve like canonical name, ipv4/v6 address, aliases, host info, etc. First we call getbyname function and pass the string argument and a pointer to DNS record. We check for the return value of getbyname() function, if its less than 0 we tell the user that there is an error in communication with th user, otherwise we write the date , time and typoe of request in the log file. The successful call to getbyname() returns all the required info in the foirm of dns record structure, so we use that structure to print all info in the formatted way.

DNS packet structure we have used a structure dns_pkt which captures all the information of the DNS header as used in the standard format.

Getbyname() – First of all we make the query, to make the query we divide the string argument with the delimiter *being.* After forming the query we enter all the required values for DNS packet structure. Then we read the resolv.conf file to get the address of the named server. Once done with it we establish the connection with UDP protocol. Once the connection is establish we set the query time of the query like 1 for ip address, 2 for named serer records, 13 for hostinfo, etc. Each time after setting the query we send the packet to the server to check for any error in receiving the response

Once we are done with the entire query we return record structure all of its fields set as per the reply from server to make call function.

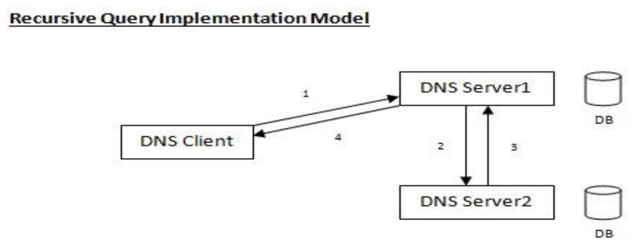
Before communicating with the server we set the loop for no. of tries and also alarm for number of seconds for response.

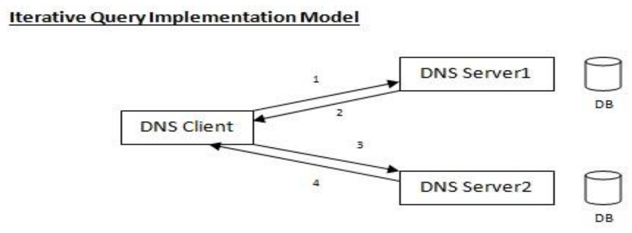
We have also implemented the TCP protocol, each time a server replies with the UDP packet of size greater than 512 bytes by checking TC flag. If set implies size greater than 512 byes then TCP connection and retry with query

Self_types.h is a header file under in which we have defined our own structure like DNS record and dns_pkt and the flags we need.

Phase 2: In phase 2 we have created 2 different servers and a client application. The client sends a query string such as google.com, the query is then received by the named Server and the query string is searched in the database (the text file info1.txt). If the required string is found, the named server sends a string containing all the required information about the domain name (the IP addresses, canonical name, host info, etc.) to the client. If not found in info1.txt, the request is forwarded to the second server by the first server and the query string is searched in the second server database (info2.txt). The server then returns the required information to the client in the form of a string. This is how we have implemented the recursive query model.

For the iterative query model request is first sent to the named server, if found string is returned to the client, else the query is sent to the second server by the client. A **Make file** is also a there for this part in the appendix.





Conclusion:

DNS is a basic unit of internet. This report describes about a DNS server-client application. The client side queries for some information and then sends it to the resolver which inturn sends it to the server. The functionality of DNS server client application is implemented without using DNS library functions.s

Note: Make file for phase 2 is there in Appendix Section

Appendix:

Phase 1 code:

Log.txt file – initially empty, later it gets filled

Main Application file (main.c)

```
#include<stdio.h>

#include<string.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<sys/signal.h>

#include<sys/errno.h>

#include<netdb.h>                    //hostent struct and h_errno

#include<stdlib.h>                   //atoi()

#include<time.h>

#include<arpa/inet.h>

#include "def_types.h"


#define T_A 1


u_char* ReadName(unsigned char* reader,unsigned char* buffer,int* count);


static void sig_alrm(int signo)

{

        printf("connection timeout error!!\n");

        return;

}


int getbyname(char* str, struct DNSrecord* host)
```

```
{
        extern int NUMBER_OF_TRIES;
        extern int WAIT_FOR_REPLY;

        unsigned char buf[65536], *ans;

        struct dns_header *dns = NULL;
        struct question* query = NULL;
        struct resource_record answers[20];

        //making fully qualified name based on question field

        dns = (struct dns_header*)&buf;
        dns->identification_no = (unsigned short) htons(rand()%65000);
        dns->QR= 0;
        dns->opcode= 0;
        dns->AA= 0;
        dns->TC= 0;
        dns->RD= 0;
        dns->RA= 0;
        dns->zero1= 0;
        dns->zero2= 0;
        dns->zero3= 0;
        dns->rcode= 0;
        dns->no_of_questions = htons(1);
        dns->no_of_answers = 0;
        dns->no_of_authoritative_RR = 0;
```

```
		dns->no_of_additional_RR = 0;


		unsigned char* ques = (unsigned char*)&buf[sizeof(struct dns_header)];
		int lock = 0 , i, j;
		strcat((char*)str,".");
		for(i = 0, j = 0 ; j < strlen((char*)str) ; i++, j++)
		{	if(i == lock) i++;
			if(str[j]!='.')
			{
				*(ques+i) = str[j];
			}
			else
			{
				char dig = (char)(((int)'0')+(i-lock-1));
				*(ques+lock) = dig;
				lock = i;
			}
		}
		*(ques+j)='\0';
		strcat((char*)ques,"0");
	printf("\nVALUE OF str %s\n",ques);


		query = (struct question*)&buf[(sizeof(struct dns_header)) + (strlen((const char*)ques) + 1)];
		query->query_type = htons(T_A);
		query->query_class = htons(1);


		int pkt_size = 65536;
```

```
        int rt;


        int sockfd;

        struct sockaddr_in servaddr;


        sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);


        //bzero(&servaddr, sizeof(servaddr));

        servaddr.sin_family = AF_INET;

        servaddr.sin_port = htons(53);          //port 53 must i think


        char word[20];

        FILE* fp = fopen("/home/mypc/Downloads/fwd/resolv.conf","r");


        int r;

        do

        {

                r = fscanf(fp,"%s",word);

        }while((strcmp(word,"nameserver") != 0) &&  (r != EOF));

        fscanf(fp,"%s",word);


        servaddr.sin_addr.s_addr = inet_addr(word);

printf("\nVALUE OF word %s %d\n",word,(int)strlen(word));


        int tries = NUMBER_OF_TRIES;

        int len = sizeof(servaddr);

        signal(SIGALRM, sig_alrm);
        int check = 0;


        while(tries > 0)

        {

        //          char rbuf[pkt_size];

                    int n;

                    short int m = 0;


                    alarm(WAIT_FOR_REPLY);


                    rt = sendto(sockfd, (char*)buf, (sizeof(struct dns_header)) + (strlen((const
char*)ques)+1) + (sizeof(struct question)), 0,(struct sockaddr*) &servaddr, len);

        printf("\n rt = %d",rt);




                    if((n = recvfrom(sockfd, (char*)buf,65536, 0,(struct sockaddr*) &servaddr,
(socklen_t*)&len)) < 0)

                    {

                                if(errno == EINTR)

                                            printf("connection timeout");

                                else

                                            printf("recvfrom error");

                    }

                    else{


                                //alarm(0);

                                dns = (struct dns_header*)buf;

                                if((dns->TC) == 1)
```

```
                {
                        check = 1;
                        break;
                }

                ans = &buf[sizeof(struct dns_header) + (strlen((const char*)ques)+1) +
sizeof(struct question)];
                printf("\n\n Received Packet: %s ",buf);
                printf("\nThe response contains : ");
                printf("\n %d Questions.",ntohs(dns->no_of_questions));
                printf("\n %d Answers.",ntohs(dns->no_of_answers));
                printf("\n %d Authoritative Servers.",ntohs(dns->no_of_authoritative_RR));
                printf("\n %d Additional records.\n\n",ntohs(dns->no_of_additional_RR));
                //if(*(ans) == '\0') printf("no ans");
                int count = 0;

/*              for(i=0;i<ntohs(dns->no_of_answers);i++)
                {
                        count += strlen(s) + 1;
                        read_str(buf,,count);
        v               char* t = &ans
//                      answers[i].type = ;
                        count += (2*sizeof(unsigned short) + sizeof(unsigned int));
//                      answers[i].res_data = ;
                        if(ntohs(answers[i].type) == 1)
                                strcpy(host->IPv4[i],answers[i].res_data);
                        else
                                strcpy(host->IPv6[i],answers[i].res_data);
                }
*/

                int stop = 0, j;
                for(i=0;i<ntohs(dns->no_of_answers);i++)
                {
                answers[i].domain_name=ReadName(ans,buf,&stop);
                ans = ans + stop;

                    answers[i].res = (struct RR*)(ans);
                    ans = ans + sizeof(struct RR);

                    if(ntohs(answers[i].res->type) == 1) //if its an ipv4 address
                {
                      answers[i].res_data = (unsigned char*)malloc(ntohs(answers[i].res-
>len));

                      for(j=0 ; j<ntohs(answers[i].res->len) ; j++)
                  {
                        answers[i].res_data[j]=ans[j];
                  }

                      answers[i].res_data[ntohs(answers[i].res->len)] = '\0';

                      ans = ans + ntohs(answers[i].res->len);
                }
                else
                    {
                                        answers[i].res_data = ReadName(ans,buf,&stop);
                                    ans = ans + stop;
                                }
                                }
                        tries--;}
            }

            if(tries < 0)
                        return tries;
            else
                        return 0;
}



void make_call(char* str)
{printf("check0");
            struct  DNSrecord host;

            printf("check");
                        bzero(&host,sizeof(struct DNSrecord));
                        int ret =  getbyname(str,& host);
                        if(ret < 0)
                        {
                                    printf("error in communicating server! Number of tries passed!!!\n");
                        }
                        else
                        {
```

```
                        FILE* fp = fopen("log.txt","a");
                        //write date time type of request reply into the file
                        time_t mytime;
                        mytime = time(NULL);
                        fwrite(str,sizeof(str),1,fp);
                        fwrite(&mytime,sizeof(mytime),1,fp);

                        printf(";;QUESTION SECTION:\n");
                        printf(";%s      IN      A\n\n",str);
                        printf(";;ANSWER SECTION:\n");

                        printf("\n%s",host.IPv4[0]);
/*                      printf("\n%s",host.IPv6[0]);
                        printf("\n%s",host.aliases[0]);
                        printf("\n%s",host.hostinfo);
                        printf("\n%s",host.mail_server);
                        printf("\n%s",host.name_server);        */
                }
}

int main(int argc, char* argv[])
{
        int n = 1;
        extern int RES_NON_REC;
        extern int NUMBER_OF_TRIES;
        extern int WAIT_FOR_REPLY;
        extern int RES_DEBUG;
```

```
	if(argc==1)

	{

		printf("Enter the name/address you want to know about. You can also explore the following options:\n");

		printf(" -R for iterative/non-recursive\n");

		printf(" -r <number> for number of tries\n");

		printf(" -w <wait seconds> for wait time for response\n");

		return -1;

	}

	do

	{

//assuming must enter int with r and w

		if(strcmp(argv[n],"-R") == 0)

		{

			RES_NON_REC = 1;//set some flag}

		}

		else if(strcmp(argv[n],"-r") == 0)

		{

			NUMBER_OF_TRIES = atoi(argv[++n]);

		}

		else if(strcmp(argv[n],"-w") == 0)

		{

			WAIT_FOR_REPLY = atoi(argv[++n]);

		}

		else if(strcmp(argv[n],"d") == 0)

		{//set some flag

			RES_DEBUG = 1;
```

```
                }
                else if(strncmp(argv[n],"-",1) == 0)
                {
                        printf("Illegal options!\n");
                        return -1;
                }
                else
                        break;          //assuming user doesn't enter any option after a name/add}
        }while(++n < argc);

        char str[20];
        while(n < argc)
        {
                strcpy(str,argv[n]);
        printf("3st %s\n",str);
                make_call(str);
                n++;
        }

        return 0;
}

u_char* ReadName(unsigned char* reader,unsigned char* buffer,int* count)
{
  unsigned char *name;
  unsigned int p=0,jumped=0,offset;
  int i , j;
```

```
*count = 1;
name = (unsigned char*)malloc(256);

name[0]='\0';

//read the names in 3www6google3com format
while(*reader!=0)
{
   if(*reader>=192)
   {
      offset = (*reader)*256 + *(reader+1) - 49152; //49152 = 11000000 00000000 ;)
      reader = buffer + offset - 1;
      jumped = 1; //we have jumped to another location so counting wont go up!
   }
   else
   {
      name[p++]=*reader;
   }

   reader = reader+1;

   if(jumped==0)
   {
      *count = *count + 1; //if we havent jumped to another location then we can count up
   }
}
```

```
    name[p]='\0'; //string complete
    if(jumped==1)
    {
        *count = *count + 1; //number of steps we actually moved forward in the packet
    }

    //now convert 3www6google3com0 to www.google.com
    for(i=0;i<(int)strlen((const char*)name);i++)
    {
        p=name[i];
        for(j=0;j<(int)p;j++)
        {
            name[i]=name[i+1];
            i=i+1;
        }
        name[i]='.';
    }
    name[i-1]='\0'; //remove the last dot
    return name;
}
```

Def_types.h header file

```
int NUMBER_OF_TRIES = 1;
int  WAIT_FOR_REPLY = 15;
int RES_DEBUG = 0;
int RES_NON_REC = 0;
```

```
struct DNSrecord

{

	char* IPv4[20];

	char* IPv6[20];

	char* aliases[15];

	char* hostinfo;

	char* name_server;

	char* mail_server;

};



//int getbyname(char* str, struct DNSrecord* host);

//struct DNSrecord* getbyaddr(char* str, int x, int y);


struct question

{

	unsigned short query_type;

	unsigned short query_class;

};


struct RR

{

	unsigned short type;

	unsigned short RR_class;

	unsigned int TTL;

	unsigned short len;

};
```

```
struct resource_record

{

            char* domain_name;

            struct RR* res;

            char* res_data;

};


struct dns_header

{

            unsigned short identification_no;

            unsigned char QR :1;

            unsigned char opcode: 4;

            unsigned char AA :1;

            unsigned char TC :1;

            unsigned char RD :1;

            unsigned char RA :1;

            unsigned char zero1: 1;

            unsigned char zero2: 1;

            unsigned char zero3: 1;

            unsigned char rcode: 4;

            unsigned short no_of_questions;

            unsigned short no_of_answers;

            unsigned short no_of_authoritative_RR;

            unsigned short no_of_additional_RR;

};
```

Resolv.conf

```
nameserver 208.67.222.222

nameserver2 208.67.222.220
```

Phase 2 Code:

Make file :

```
PROGS =         dnscli dnsser dnsser2

all:    ${PROGS}

dnscli: dnscli.o

                ${CC} ${CFLAGS} -o $@ dnscli.o ${LIBS}

dnsser: dnsser.o

                ${CC} ${CFLAGS} -o $@ dnsser.o ${LIBS}

dnsser2:        dnsser2.o

                ${CC} ${CFLAGS} -o $@ dnsser2.o ${LIBS}

clean:

                rm -f ${PROGS} ${CLEANFILES}
```

Client Side code - dnscli.c

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/stat.h>

#include <sys/uio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>


#define  MAXLINE                4096
```

```
#define  SA          struct sockaddr


int main(int argc, char **argv)
{
        int                                                 n,sockfd,flag=0;
        struct sockaddr_in           servaddr;
        char query[MAXLINE], query1[MAXLINE], recvline[MAXLINE + 1];
        if (argc != 2)
                {
                printf("usage: udpcli <IPaddress>");
                exit(1);


                }
        bzero(&servaddr, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = htons(1500);
        inet_pton(AF_INET, argv[1], &servaddr.sin_addr);


        sockfd = socket(AF_INET, SOCK_DGRAM, 0);
        while (fgets(query, MAXLINE, stdin) != NULL) {
                fputs("Recursive(1) or Iterative(0) : ", stdout);
                fgets(query1, MAXLINE, stdin);
        //printf("a%da\n",query1);
                sendto(sockfd, query, strlen(query), 0,(SA *) &servaddr, sizeof(servaddr));
                sendto(sockfd, query1, strlen(query1), 0,(SA *) &servaddr, sizeof(servaddr));
                n = recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);


                recvline[n] = 0;
                if((strcmp(recvline,"No record found") != 0 ))
```

```
		{
			fputs(recvline, stdout);
			flag=1;
		}
		if(flag != 1 && (strcmp(query1,"0\n") == 0))
		{
			//Fputs("Iterative", stdin);
			int	n1,sockfd1;
			struct sockaddr_in	servaddr1;



			bzero(&servaddr1, sizeof(servaddr1));
			servaddr1.sin_family = AF_INET;
			servaddr1.sin_port = htons(1501);
			inet_pton(AF_INET, "127.0.0.1", &servaddr1.sin_addr);
			sockfd1 = socket(AF_INET, SOCK_DGRAM, 0);
			sendto(sockfd1, query, strlen(query), 0, (SA *) &servaddr1, sizeof(servaddr1));
			sendto(sockfd1, query1, strlen(query1), 0, (SA *) &servaddr1, sizeof(servaddr1));
			n1 = recvfrom(sockfd1, recvline, MAXLINE, 0, NULL, NULL);

			recvline[n1] = 0;

			fputs(recvline, stdout);
		}
	}
	exit(0);
}
```

Dnsser.c – the first server's code:

```
#include <stdio.h>
```

```
#include <stdlib.h>

#include <string.h>

#include <sys/stat.h>

#include <sys/uio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>


#define  MAXLINE          4096

#define  SA        struct sockaddr


char * retri123(char *mesg, char *mesg2)

{


        FILE *fp;

        fp=fopen("inf1.txt","r");

        char line[200], sendline[MAXLINE];

        //strcpy(msg,"No record found");

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

        {

                if(strcmp(line,"")==0)

                break;

                if(strcmp(line,mesg) == 0)

                {

                        fgets(line,sizeof(line),fp);

                        while(strcmp(line,"\n")!=0)

                        {

                                strcat(sendline, line);
                        fgets(line,sizeof(line),fp);
                    }
                    return sendline;
            }
    }
//printf("%s1\n",msg);
    return mesg;
}




int
main(int argc, char **argv)
{
    int     sockfd;
    socklen_t len1;
    struct sockaddr_in      servaddr, cliaddr;
    char mesg[MAXLINE], mesg1[MAXLINE], send_line[MAXLINE];
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family     = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port       = htons(1500);

    bind(sockfd, (SA *) &servaddr, sizeof(servaddr));
```

```
        for ( ; ; ) {
                len1 = sizeof(cliaddr);
                recvfrom(sockfd, mesg, MAXLINE, 0, (SA *) &cliaddr, &len1);
                recvfrom(sockfd, mesg1, MAXLINE, 0, (SA *) &cliaddr, &len1);
                if(strcmp(mesg1, "1\n")==0)
                        printf("Recursive\n");
                strcpy(send_line,retri123(mesg, mesg1));
                fputs(send_line,stdout);
                if(strcmp(send_line, mesg)!=0)
                {
                        sendto(sockfd, (char *)send_line, sizeof(send_line), 0, (SA *) &cliaddr, len1);


                }
                else if(strcmp(mesg1,"1\n")!=0)
                {


                        sendto(sockfd, (char *) "No record found", sizeof("No record found"), 0, (SA *)
    &cliaddr, len1);
                        printf("ohk!!\n");
                }
                else
                {
                printf("searching in second dns server\n");


                        int                                     n1,sockfd1;
                        struct sockaddr_in          servaddr1;


                        bzero(&servaddr, sizeof(servaddr));
                        servaddr1.sin_family = AF_INET;
                        servaddr1.sin_port = htons(1501);

                        inet_pton(AF_INET, "127.0.0.1", &servaddr1.sin_addr);


                        sockfd1 = socket(AF_INET, SOCK_DGRAM, 0);

                        sendto(sockfd1, mesg, strlen(mesg), 0,(SA *) &servaddr1, sizeof(servaddr1));

                        sendto(sockfd1, "0", strlen(mesg1), 0,(SA *) &servaddr1, sizeof(servaddr1));

                        n1 = recvfrom(sockfd1, send_line, MAXLINE, 0, NULL, NULL);

                        send_line[n1] = 0;

                        sendto(sockfd, (char *)send_line, sizeof(send_line), 0, (SA *) &cliaddr, len1);

                }

        }

}
```

Dnsser2.c code – the second server's code:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/stat.h>

#include <sys/uio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>


#define  MAXLINE           4096

#define  SA       struct sockaddr


char * retri123(char *mesg, char *mesg2)

{
    FILE *fp;

    fp=fopen("inf2.txt","r");

    char line[200], sendline[MAXLINE];

    //strcpy(msg,"No record found");

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

    {

        if(strcmp(line,"")==0)

        break;

        if(strcmp(line,mesg) == 0)

        {

            fgets(line,sizeof(line),fp);

            while(strcmp(line,"\n")!=0)

            {

                strcat(sendline, line);


                fgets(line,sizeof(line),fp);

            }

            return sendline;

        }

    }

//printf("%s1\n",msg);

    return mesg;

}





int

main(int argc, char **argv)

{
```

```
int         sockfd;

socklen_t len1;

struct sockaddr_in            servaddr, cliaddr;

char mesg[MAXLINE], mesg1[MAXLINE], send_line[MAXLINE];

sockfd = socket(AF_INET, SOCK_DGRAM, 0);


bzero(&servaddr, sizeof(servaddr));

servaddr.sin_family      = AF_INET;

servaddr.sin_addr.s_addr = htonl(INADDR_ANY);

servaddr.sin_port       = htons(1501);


bind(sockfd, (SA *) &servaddr, sizeof(servaddr));



for ( ; ; ) {

            len1 = sizeof(cliaddr);

            recvfrom(sockfd, mesg, MAXLINE, 0, (SA *) &cliaddr, &len1);

            recvfrom(sockfd, mesg1, MAXLINE, 0, (SA *) &cliaddr, &len1);

            if(strcmp(mesg1, "1\n")==0)

                        printf("Recursive\n");

            strcpy(send_line,retri123(mesg, mesg1));

            fputs(send_line,stdout);

            if(strcmp(send_line, mesg)!=0)

            {

                        sendto(sockfd, (char *)send_line, sizeof(send_line), 0, (SA *) &cliaddr, len1);


            }

            else if(strcmp(mesg1,"1\n")!=0)

            {
```

```
                        sendto(sockfd, (char *) "No record found", sizeof("No record found"), 0, (SA *)
&cliaddr, len1);

                        printf("ohk!!\n");

                }

                else

                {

                printf("searching in second dns server\n");


                        int                                             n1,sockfd1;

                        struct sockaddr_in              servaddr1;


                        bzero(&servaddr, sizeof(servaddr));

                        servaddr1.sin_family = AF_INET;

                        servaddr1.sin_port = htons(1500);

                        inet_pton(AF_INET, "127.0.0.1", &servaddr1.sin_addr);


                        sockfd1 = socket(AF_INET, SOCK_DGRAM, 0);

                        sendto(sockfd1, mesg, strlen(mesg), 0,(SA *) &servaddr1, sizeof(servaddr1));

                        sendto(sockfd1, "0", strlen(mesg1), 0,(SA *) &servaddr1, sizeof(servaddr1));

                        n1 = recvfrom(sockfd1, send_line, MAXLINE, 0, NULL, NULL);

                        send_line[n1] = 0;

                        sendto(sockfd, (char *)send_line, sizeof(send_line), 0, (SA *) &cliaddr, len1);

                }

        }

}
```