Ex. No. 5	DNS Using UDP	Karthik D
16 August 2021	UCS1511 - Networks Lab	195001047

Aim

To simulate the working of Domain Name Server using socket programming with UDP in the C language, with the help of TCP-based multi-client server

Question

Simulate ARP using UDP socket programming, wherein:

The Server should perform the following:

- Maintain a DNS in the form of a table.
- The table contains IP address and the corresponding
- Server name and displays the table
- When a request is for an IP address (Given a server name), from a client is received, verify the table and send the corresponding IP address to the client
- Make server to accept multiple client request simultaneously
- Also modify the server

The Client should do the following:

- Request for an IP address is given to the server by the domain name
- Receive the corresponding IP address and display it

Algorithms

(a) Server-side

Step 1: Globally define a dynamic array to store the DNS table. This is initially populated with a few domain names and their corresponding IP addresses

Step 2: Create a network socket with parameters suitable for an end-point of UDP based communication

Step 3: Bind the socket to INADDR_ANY which is defined as a *zero* address, allowing the socket to be reachable by all active interfaces on the device. Set the port to a preset value, known to the targeted clients as well

Step 4: Initiate a separate thread to await user input on the server-side to update the DNS table.

i: If user requests an update by pressing ‘u’

ii: Accept a domain name

iii: Accept a corresponding IP address. If the IP address is in an invalid format, reject the request and repeat from step-(i)

iv: If the IP address is already matched to a domain name, reject the request and repeat from step-(i)

v: Add an entry into the DNS table for the supplied domain name and IP address

Step 5: On the main thread, prepare a memory buffer to read and store messages from the connection.

Step 6: Start an infinite loop to perform the following operations,

i: Wait for a UDP-based DNS request from a client. If a predetermined timeout duration passes before a request arrives, terminate the loop

ii: Read the message buffer to obtain the domain name requested by the client

iii: Lookup the domain name on the DNS table

iv: If the domain is found, send all corresponding IP addresses to the client using the *write()* system call, one IP address per message. Continue to step-(vi)

v: If the domain is not found, continue to step-(vi)

vi: Terminate the sequence by sending a predetermined string to indicate the end of response

(Repeat till timeout when waiting for a DNS request in step-(i))

Step 7: Close the created sockets using the *close()* system call and terminate the process

(b) **Client-side**

Step 1: Create a network socket with parameters suitable for an end-point of UDP based communication

Step 2: Accept the Domain Name Server’s IP address as input from the user

Step 3: Using the accepted IP address and a preset port number agreed upon between the server and client

Step 4: Prepare a memory buffer to read and store messages from the connection.

Step 5: Start an infinite loop to perform the following operations,

i: Accept a domain name as input from the user

ii: Use the *write()* system call send the accepted domain name to the server using the accepted IP address and a preset port number agreed upon between the server and client

iii: Use the *read()* system call to block and read a message sent from the server repeatedly until the server sends the response termination string. Store and display each the message into and from the buffer, each time

iv: Ask the user the choice to continue with more requests or stop. If the user chooses to continue, repeat from step-(i). Otherwise, terminate the loop

(Repeat till user chooses to terminate)

Step 6: Close the created sockets using the *close()* system call and terminate the process

C Program Code

1. udp_socket.h - UDP connection helper functions

```
#ifndef udp_socket
#define udp_socket

#include<sys/socket.h>
#include<arpa/inet.h>
#include<unistd.h>
#include<string.h>
#include<errno.h>

#define SERVER_PORT 8088
#define LOCALHOST_IP "127.0.0.1"
#define ADDRESS_FAMILY AF_INET
#define ADDRESS_BUFFER_SIZE 30
#define MSG_BUFFER_SIZE 100
#define IP_STRING_LEN 24
#define MSG_DELIMITER ';'
```

```c
#define MSG_WAIT_TIMEOUT 20
#define RESPONSE_END_STRING "END_OF_RESPONSE"

/*
Use BLOCKING sockets (default configuration)
Alternating send-receive only
No need to initiate messages on the server!
(i.e) Synchronous send/receive
*/

int make_socket(){
   // AF_INTER specifies IPv4
   // SOCK_STREAM specifies two-way byte-stream
   // 0 selects default protocol
   int sock_fd = socket(ADDRESS_FAMILY, SOCK_DGRAM, IPPROTO_UDP);
   if (sock_fd == -1){
       return -2;    // Could not create socket
   }
   return sock_fd;
}

short bind_server_socket(int sock_fd){
   struct sockaddr_in bind_address;
   // Set family to IPv4
   bind_address.sin_family = ADDRESS_FAMILY;
   // Set port in network byte-order to a non-privileged port (>1023)
   bind_address.sin_port = htons(SERVER_PORT);
   // Set address to 0.0.0.0 to connect to bind to all local interfaces
   bind_address.sin_addr.s_addr = htonl(INADDR_ANY);
   // Convert to generic socket address format and bind
   if (!bind(sock_fd, (struct sockaddr *)&bind_address,
sizeof(bind_address))){
       return 0;    // Success
   }
   else{
       printf("%d", errno);
       return -3;   // Could not bind server-socket
   }
}
```

```
struct sockaddr_in wrap_address(char *ip_address, int port){
   struct sockaddr_in address;
   bzero((char*)&address, sizeof(address));
   // Set family to IPv4
   address.sin_family = ADDRESS_FAMILY;
   // Set port in network byte-order to a non-privileged port (>1023)
   address.sin_port = htons(port);
   // Set the ip address in byte format
   address.sin_addr.s_addr = inet_addr(ip_address);
   return address;
}

void destroy_socket(int sock_fd){
   close(sock_fd);
}

#endif
```

2. msg_io.h - Message transfer helper functions

```
#ifndef msg_io
#define msg_io

#include<sys/types.h>

#include "udp_socket.h"

struct timeval prepare_time_structure(int duration_sec, int
duration_usec){
   struct timeval time;
   time.tv_sec = duration_sec;
   time.tv_usec = duration_usec;
   return time;
}

int wait_for_message(int *server_fds, int num_fds, fd_set *avl_fds){
   fd_set read_fds;
   FD_ZERO(&read_fds);
   int max_fd = -1;
```

```
    for(int i=0;i<num_fds;i++){
        FD_SET(*(server_fds+0), &read_fds);
        if (*(server_fds+0) > max_fd){
            max_fd = *(server_fds+0);
        }
    }
    struct timeval timeout = prepare_time_structure(MSG_WAIT_TIMEOUT, 0);
    int avl_fds_count = select(max_fd+1, &read_fds, NULL, NULL, &timeout);
    if(avl_fds_count==-1){
        return -8;    // Error when waiting for socket
    }
    else if(avl_fds_count==0){
        return -9;    // Timed-out when waiting for message
    }
    *avl_fds = read_fds;
    return avl_fds_count;
}

ssize_t receive_message(int socket, char *msg_buffer, struct sockaddr_in
*sender_addr, int *sender_addr_len){
    int addr_buffer_size = sizeof(struct sockaddr_in);
    int msg_size = recvfrom(socket, msg_buffer, MSG_BUFFER_SIZE,
MSG_WAITALL, (struct sockaddr*)sender_addr, sender_addr_len);
    if (*sender_addr_len > addr_buffer_size){
        *sender_addr_len = -1;  // Warning: Client address was truncated to
fit in buffer
    }
    return msg_size;
}

ssize_t send_message(int socket, char *msg, struct sockaddr_in
*destn_addr, int destn_addr_len){
    // Using MSG_CONFIRM to prevent ARP reprobing, since this is a reply
message
    int msg_size = sendto(socket, msg, MSG_BUFFER_SIZE, MSG_CONFIRM,
(struct sockaddr*)destn_addr, destn_addr_len);
    if(msg_size == -1){
        return -6;    // Couldn't send message
    }
    return msg_size;
```

```
}

ssize_t send_reply(int socket, char *msg, struct sockaddr_in *destn_addr,
int destn_addr_len){
   // Using MSG_CONFIRM to prevent ARP reprobing, since this is a reply
message
   int msg_size = sendto(socket, msg, MSG_BUFFER_SIZE, MSG_CONFIRM,
(struct sockaddr*)destn_addr, destn_addr_len);
   if(msg_size == -1){
       return -7;    // Couldn't send reply
   }
   return msg_size;
}

#endif
```

3. DNSTable.h - ADT for managing the DNS Table

```
#ifndef DNS_Table_h
#define DNS_Table_h

#define DOMAIN_NAME_SIZE 100
#define IP_ADDRESS_SIZE 20

#include<stdlib.h>
#include<ctype.h>

// NULL-based Linked-List
struct dns_table{
   char *domain_name;
   char **ips;
   int num_ips;
   struct dns_table *next;
};
typedef struct dns_table DNS_Table;


DNS_Table* make_dns_entry(char *domain_name, char *ip){
   DNS_Table *dns_table = (DNS_Table*)malloc(sizeof(DNS_Table));
```

```
    dns_table->domain_name = (char*)malloc(sizeof(char)*DOMAIN_NAME_SIZE);
    dns_table->ips = (char**)malloc(sizeof(char*));
    *(dns_table->ips+0) = (char*)malloc(sizeof(char)*IP_ADDRESS_SIZE);
    memcpy(dns_table->domain_name, domain_name, DOMAIN_NAME_SIZE);
    memcpy(*(dns_table->ips+0), ip, IP_ADDRESS_SIZE);
    dns_table->num_ips = 1;
    dns_table->next = NULL;
    return dns_table;
}


DNS_Table* get_dns_entry(char *domain_name, DNS_Table *table){
    DNS_Table *handle = table;
    while(handle!=NULL){
        if(strcmp(domain_name, handle->domain_name)==0){
            return handle;
        }
        handle = handle->next;
    }
    return NULL;
}


DNS_Table* add_dns_ip(char *domain_name, char *ip, DNS_Table *table){
    DNS_Table *handle = table;
    DNS_Table *entry = NULL;
    DNS_Table *prev = NULL;
    char *ip_parser;
    while(handle!=NULL){
        if(strcmp(handle->domain_name, domain_name)==0){
            entry = handle;
        }
        for(int i=0;i<handle->num_ips;i++){
            if(strcmp(*(handle->ips+i), ip)==0){
                printf("HERE");
                return NULL;     // IP already exists
            }
        }
        prev = handle;
        handle = handle->next;
```

```
    }
    if(entry!=NULL){
        entry->ips = (char**)realloc(entry->ips,
sizeof(char*)*(entry->num_ips+1));
        *(entry->ips+entry->num_ips) =
(char*)malloc(sizeof(char)*IP_ADDRESS_SIZE);
        *(entry->ips+entry->num_ips) = ip;
        entry->num_ips++;
    }
    else{
        entry = make_dns_entry(domain_name, ip);
        if(prev==NULL){
            table = entry;
        }
        else{
            prev->next = entry;
        }
    }
    return table;
}


void display_dns_table(DNS_Table *table){
    DNS_Table *handle = table;
    printf("\n     DNS TABLE");
    printf("\n------------------------------------------------------------ ");
    printf("\n|       Domain Name        |        IP Address        |");
    printf("\n------------------------------------------------------------ ");
    while(handle!=NULL){
        printf("\n|    %-22s", handle->domain_name);
        for(int i=0;i<handle->num_ips;i++){
            if(i==0){
                printf("|    %-22s|", *(handle->ips+i));
            }
            else{
                printf("\n|    %-22s|    %-22s|", " ", *(handle->ips+i));
            }
        }
        printf("\n------------------------------------------------------------
");
```

```
        handle = handle->next;
    }
    printf("\n\n");
}

short validate_IP(char *ip){
    /* Return 1 if valid IP, 0 otherwise */
    int quad_1 = -1;
    int quad_2 = -1;
    int quad_3 = -1;
    int quad_4 = -1;
    sscanf(ip,
        "%d.%d.%d.%d",
        &quad_1,
        &quad_2,
        &quad_3,
        &quad_4
    );
    if(quad_1<0 || quad_1>255 || quad_2<0 || quad_2>255 ||
         quad_3<0 || quad_3>255 || quad_4<0 || quad_4>255){
            return 0;       // Invalid IP
    }
    else{
        char *parser = ip;
        // Assume dot at beginning of string
        int dot = 1;
        int zero_start = 0;
        while(*parser!='\0'){
            if(zero_start && isdigit(*parser)){
                return 0;     // Invalid IP
            }
            // zero-start flag
            if(dot && *parser=='0'){
                zero_start = 1;
            }
            else{
                zero_start = 0;
            }
            // dot flag
            if(*parser=='.'){
```

```
                dot = 1;
            }
            else{
                dot = 0;
            }
            parser++;
        }
    }
    return 1;
}

#endif
```

4. server.c - Server-side script

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<regex.h>

#ifndef udp_socket
    #include "udp_socket.h"
#endif

#ifndef msg_io
    #include "msg_io.h"
#endif

#ifndef DNS_Table_h
    #include "DNSTable.h"
#endif

// Global DNS Table
DNS_Table* dns_table = NULL;
short updating = 0;
short updated = 0;

void* update_dns_table(){
    DNS_Table *result = NULL;
```

```
    char response;
    char *domain_name = (char*)malloc(sizeof(char)*DOMAIN_NAME_SIZE);
    char *ip_address = (char*)malloc(sizeof(char)*IP_ADDRESS_SIZE);
    do{
        scanf(" %c", &response);
        if(response=='u'||response=='U'){
            updating++;
            printf("\nEnter Domain Name: ");
            scanf(" %s", domain_name);
            printf("Enter IP Address: ");
            scanf(" %s", ip_address);
            if(!validate_IP(ip_address)){
                printf("\nInvalid IP Address\n");
            }
            else{
                result = add_dns_ip(domain_name, ip_address, dns_table);
                if(result==NULL){
                    printf("\nIP Address already exists\n");
                }
                else{
                    dns_table = result;
                }
            }
            display_dns_table(dns_table);
            updating--;
            updated++;
        }
        response = 'z';
    }while(1==1);
}


void main(){

    // Populate DNS table
    dns_table = add_dns_ip("www.google.com", "192.168.0.1", dns_table);
    dns_table = add_dns_ip("www.google.com", "192.167.0.1", dns_table);
    dns_table = add_dns_ip("www.yahoo.com", "192.67.0.1", dns_table);
    display_dns_table(dns_table);
```

```
    int self_socket = make_socket();
    if(self_socket<0){
        printf("\nCould not create socket. Retry!\n");
        return;
    }

    if (bind_server_socket(self_socket)<0){
        printf("\nCould not bind server socket. Retry!\n");
        destroy_socket(self_socket);
        return;
    }

    pthread_t updater_thread_id;
    if(pthread_create(&updater_thread_id, NULL, (void*)(update_dns_table),
NULL)){
        printf("\nError while creating thread for DNS Table Updation\n");
    }

    // Keep track of all server sockets.
    int num_sockets = 1;
    int *server_sockets = (int*)malloc(sizeof(int)*num_sockets);
    *(server_sockets+0) = self_socket;
    // To store sender-client address
    struct sockaddr_in *client_addr = malloc(sizeof(struct sockaddr_in));
    int client_addr_len = sizeof(struct sockaddr_in);
    char *client_addr_ip_str =
(char*)malloc(sizeof(char)*ADDRESS_BUFFER_SIZE);
    int client_addr_port;
    // Data transmission storage locations
    DNS_Table *dns_entry;
    char *msg_buffer = (char*)malloc(sizeof(char)*MSG_BUFFER_SIZE);
    fd_set readable_fds;
    int msg_size = 0;
    int response;
    int read_fd;
    do{
        // BLOCK till some client sends message

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

```
        printf("\nWaiting for DNS requests... Hit 'u' to update DNS
table\n");

        response = wait_for_message(server_sockets, num_sockets,
&readable_fds);
        if(response == -9){
            if(updated>0){
                updated--;
                continue;
            }
            if(updating){
                continue;
            }
            printf("\nTimed out when waiting for requests\nExiting...\n");
            break;
        }
        else if(response == -8){
            printf("\nError occurred when monitoring socket for
messages\nRetry!\n");
        }

        // Handle all available readable descriptors
        for(int read_idx=0; read_idx<num_sockets; read_idx++){
            read_fd = *(server_sockets+read_idx);

            if (FD_ISSET(read_fd, &readable_fds)==0){
                // This socket is not readable
                continue;
            }

            msg_size = receive_message(self_socket, msg_buffer,
client_addr, &client_addr_len);
            if (msg_size==0){
                printf("\nEmpty message\n");
            }
            else if(client_addr_len == -1){
                printf("\nRequest received\nCould not read sender
address!\n");
            }
            else{
```

```
                // Alternatively, use inet_ntoa
                inet_ntop(ADDRESS_FAMILY, (void*)&client_addr->sin_addr,
client_addr_ip_str, ADDRESS_BUFFER_SIZE);
                client_addr_port = (int)ntohs(client_addr->sin_port);
                if (client_addr_ip_str == NULL) {
                    printf("\nRequest received\nCould not read sender
address!\n");
                }
                else{
                    printf("\nRequest received from %s:%d",
client_addr_ip_str, client_addr_port);
                }

                printf("\n\nDomain Requested: %s", msg_buffer);
                dns_entry = get_dns_entry(msg_buffer, dns_table);
                if(dns_entry==NULL){
                    printf("\nNo IP-Address found");
                    memcpy(msg_buffer, "No IP-Address found",
MSG_BUFFER_SIZE);
                    msg_size = send_reply(self_socket, msg_buffer,
client_addr, client_addr_len);
                }
                else{
                    printf("\nIP-Addresses Responded");
                    for(int i=0;i<dns_entry->num_ips;i++){
                        memcpy(msg_buffer, *(dns_entry->ips+i),
MSG_BUFFER_SIZE);
                        msg_size = send_reply(self_socket, msg_buffer,
client_addr, client_addr_len);
                    }
                }
                msg_size = send_reply(self_socket, RESPONSE_END_STRING,
client_addr, client_addr_len);
            }
        }
    }while(1==1);

    destroy_socket(self_socket);
    return;
}
```

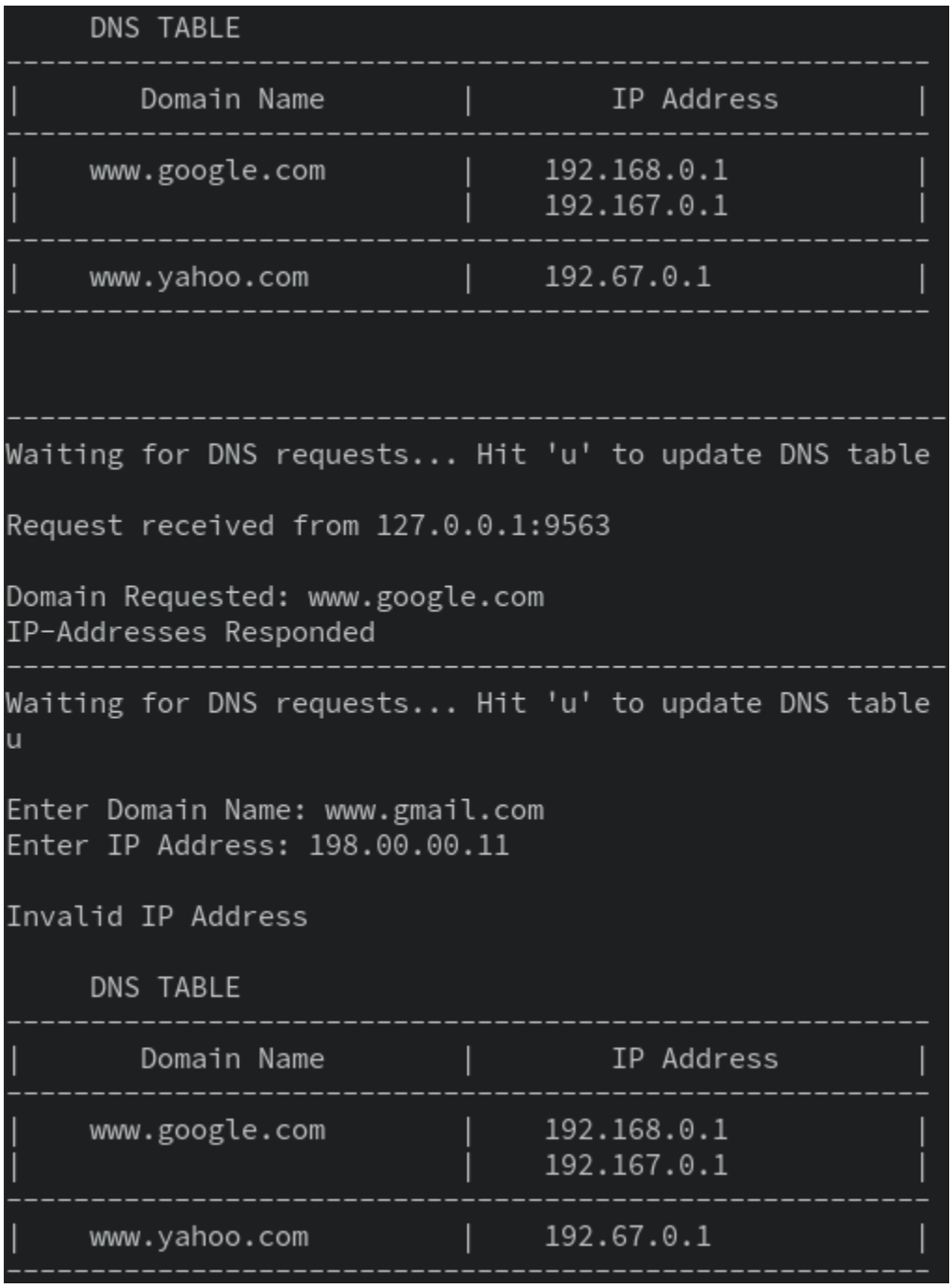
5. client.c - Client-side script

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

#ifndef udp_socket
   #include "udp_socket.h"
#endif

#ifndef msg_io
   #include "msg_io.h"
#endif

void main(){

   int self_socket = make_socket();
   if(self_socket<0){
       printf("\nCould not create socket. Retry!\n");
       return;
   }

   char *server_ip = (char*)malloc(sizeof(char)*IP_STRING_LEN);
   printf("\nEnter DNS Server IP Address: ");
   scanf(" %s", server_ip);
   struct sockaddr_in server_addr = wrap_address(server_ip, SERVER_PORT);
   int server_addr_len = sizeof(server_addr);

   // Structure to store the message source address
   struct sockaddr_in *source_addr = malloc(sizeof(struct sockaddr_in));
   int source_addr_len = sizeof(struct sockaddr_in);

   char *msg_buffer = (char*)malloc(sizeof(char)*MSG_BUFFER_SIZE);
   int msg_size = 0;
   char ch;
   do {
       bzero(msg_buffer, MSG_BUFFER_SIZE);
       printf("\nEnter Domain Name: ");
       scanf(" %s", msg_buffer);
```

```
       msg_size = send_message(self_socket, msg_buffer, &server_addr,
server_addr_len);
       bzero(msg_buffer, MSG_BUFFER_SIZE);
       do{
           msg_size = receive_message(self_socket, msg_buffer,
source_addr, &source_addr_len);
           if(strcmp(msg_buffer, RESPONSE_END_STRING)==0){
               break;
           }
           printf("%s\n", msg_buffer);
       }while(1==1);
       printf("\nMore queries? (y/n): ");
       scanf(" %c", &ch);
   }while(ch!='n');

   destroy_socket(self_socket);
   return;
}
```

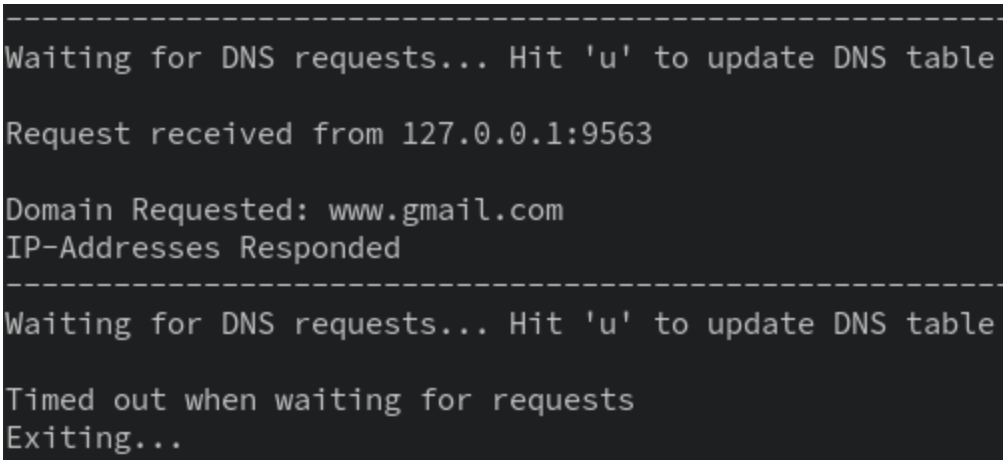
Sample Output

- Server-Side

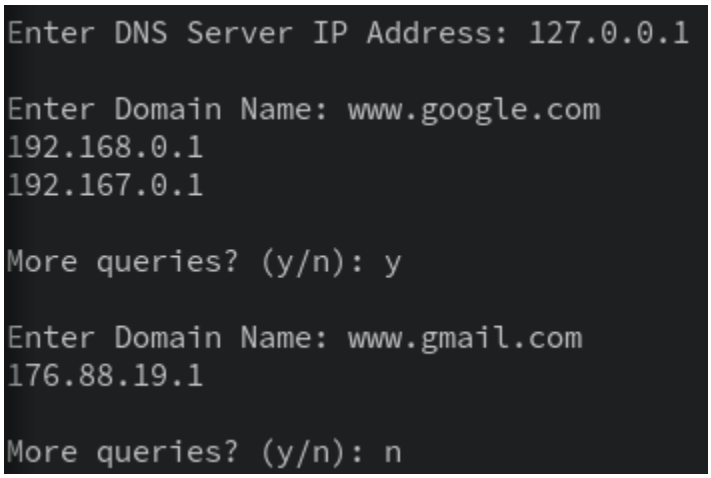


```
     DNS TABLE
-----------------------------------------------------------
|       Domain Name         |        IP Address           |
-----------------------------------------------------------
|    www.google.com         |    192.168.0.1              |
|                           |    192.167.0.1              |
-----------------------------------------------------------
|    www.yahoo.com          |    192.67.0.1               |
-----------------------------------------------------------



-----------------------------------------------------------
Waiting for DNS requests... Hit 'u' to update DNS table

Request received from 127.0.0.1:9563

Domain Requested: www.google.com
IP-Addresses Responded
-----------------------------------------------------------
Waiting for DNS requests... Hit 'u' to update DNS table
u

Enter Domain Name: www.gmail.com
Enter IP Address: 198.00.00.11

Invalid IP Address

     DNS TABLE
-----------------------------------------------------------
|       Domain Name         |        IP Address           |
-----------------------------------------------------------
|    www.google.com         |    192.168.0.1              |
|                           |    192.167.0.1              |
-----------------------------------------------------------
|    www.yahoo.com          |    192.67.0.1               |
-----------------------------------------------------------
```

```
Waiting for DNS requests... Hit 'u' to update DNS table
u

Enter Domain Name: www.gmail.com
Enter IP Address: 192.168.0.1
HERE
IP Address already exists

     DNS TABLE
-----------------------------------------------------
|       Domain Name        |        IP Address        |
-----------------------------------------------------
|    www.google.com        |    192.168.0.1           |
|                          |    192.167.0.1           |
-----------------------------------------------------
|    www.yahoo.com         |    192.67.0.1            |
-----------------------------------------------------



-----------------------------------------------------
Waiting for DNS requests... Hit 'u' to update DNS table
u

Enter Domain Name: www.gmail.com
Enter IP Address: 176.88.19.1

     DNS TABLE
-----------------------------------------------------
|       Domain Name        |        IP Address        |
-----------------------------------------------------
|    www.google.com        |    192.168.0.1           |
|                          |    192.167.0.1           |
-----------------------------------------------------
|    www.yahoo.com         |    192.67.0.1            |
-----------------------------------------------------
|    www.gmail.com         |    176.88.19.1           |
-----------------------------------------------------
```



- Client-Side



Result

Implemented a socket program in C language using UDP to simulate DNS lookup in network communication. A connectionless multi-client server is developed, wherein a server awaits a DNS request from any client, looks up the IP address and responds with the list of addresses to the client. Through this implementation, the following aspects were understood:

1. Basic functioning of the DNS and client
2. A possible algorithmic implementation of DNS lookup
3. Implementation details of socket programming using C language for UDP protocol