1.Implement the following using UDP sockets:

Server is a calculator, the client request the server to provide the operations it can perform and now the server gives the options like multiplication, addition, division and square root.(10)

SERVER

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include<arpa/inet.h>
#include <string.h>
#include<time.h>
#include<math.h>

#define BUFF_SIZE 4095
#define SA struct sockaddr_in

int create_server(SA address){

  int sock_fd ;
  if ((sock_fd = socket(AF_INET, SOCK_DGRAM, 0)) == 0)
  {
    fprintf(stderr , "failed to accept .");
    exit(EXIT_FAILURE);
  }

  if (bind(sock_fd, (struct sockaddr *)&address,
               sizeof(address))<0)
  {
    fprintf(stderr , "failed to bind");
    exit(EXIT_FAILURE);
  }

  return sock_fd ;
}
int main(int argc, char  **argv)
{

    if(argc < 2){
        fprintf(stdout , "USAGE ./%s <port>",argv[0]);
        exit(EXIT_FAILURE);
    }

    int PORT = atoi(argv[1]);

  int server_fd, new_socket;
  SA address;
  int addrlen = sizeof(address);


  char *calc = "\n1) Addition \n"
               "2) Multiplication\n"
               "3) Division\n"
               "4) Square Root\n"
               "5) EXIT\n"
              "\nEnter Operation: " ;


  address.sin_family = AF_INET;
  address.sin_port = htons( PORT );
    inet_pton(AF_INET, "127.0.0.1", &address.sin_addr);

  server_fd = create_server(address);
```

```c
    printf("server file descriptor : %d \n",server_fd);
    printf("Multithreaded calculated server listening on : %d\n",PORT);
    char buffer[BUFF_SIZE] = {0} ;

    SA host_addr, client_addr;
    socklen_t length = sizeof(SA);

    int recvbytes = recvfrom(server_fd, buffer, sizeof(buffer), 0, (struct sockaddr*)&client_addr,
&length);
    int sentbytes = sendto(server_fd, calc, strlen(calc) + 1, 0, (struct sockaddr*)&client_addr, length);

    while(1){

            bzero(buffer , BUFF_SIZE);
            recvbytes = recvfrom(server_fd, buffer, sizeof(buffer), 0, (struct sockaddr*)&client_addr,
&length);
            int operator = atoi(buffer) ;

            bzero(buffer , BUFF_SIZE);
            recvbytes = recvfrom(server_fd, buffer, sizeof(buffer), 0, (struct sockaddr*)&client_addr,
&length);
            int num1 = atoi(buffer) ;

            bzero(buffer , BUFF_SIZE);
            recvbytes = recvfrom(server_fd, buffer, sizeof(buffer), 0, (struct sockaddr*)&client_addr,
&length);
            int num2 = atoi(buffer) ;
            double n = num1 ;

            int result = 0;
            switch (operator)
                {
                    case 1:
                        result = num1 + num2 ;
                        break;
                    case 2:
                        result = num1 * num2 ;
                        break;
                    case 3:
                        result = num1/num2 ;
                        break;
                    case 4:
                        result = sqrt(n) ;
                        break;
                    default:
                        result = 0 ;
                }
            printf("The server answer is %d\n" , result);
            bzero(buffer , BUFF_SIZE);
            sprintf(buffer , "The server answer is %d" , result);
            sentbytes = sendto(server_fd, buffer, strlen(buffer) + 1, 0, (struct sockaddr*)&client_addr,
length);

            fflush(stdout);


    }

  return EXIT_SUCCESS;
}
```

CLIENT

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

#define SA struct sockaddr_in
#define BUFFER_SIZE 4095

int create_client(){

    int sock_fd ;
    if( (sock_fd = socket(AF_INET , SOCK_DGRAM , 0) )< 0){
        fprintf(stderr , "failed to open socket");
        exit(EXIT_FAILURE);
    }
    return sock_fd ;
}

int main(int argc ,  char **argv){

    if(argc < 2){
        fprintf(stderr , "USAGE %s port" , argv[0]);
        exit(EXIT_FAILURE);
    }
    int socketfd = create_client();
    int PORT = atoi(argv[1]);

    char buffer[BUFFER_SIZE] = {0};
    int n = 0 , operator , num1 , num2;

    struct sockaddr_in host_addr;
    host_addr.sin_family = AF_INET;
    host_addr.sin_port = htons(PORT);
    inet_pton(AF_INET, "127.0.0.1", &host_addr.sin_addr);



    int sentbytes = sendto(socketfd,"send options", strlen("send options") + 1, 0, (struct
sockaddr*)&host_addr, sizeof(host_addr));
    int recvbytes = recvfrom(socketfd, buffer, sizeof(buffer), 0, NULL, NULL);

    printf("%s",buffer);
    scanf("%d",&operator);

    if(operator == 3){
        printf("Enter number : ");
        scanf("%d",&num1);
        num2 = 0 ;
    }
    else{
        printf("Enter number 1 : ");
        scanf("%d",&num1);

        printf("Enter number 2 : ");
        scanf("%d",&num2);
    }

    bzero(buffer , BUFFER_SIZE);
    sprintf(buffer ,"%d", operator);

    sendto(socketfd,buffer, strlen(buffer) + 1, 0, (struct sockaddr*)&host_addr, sizeof(host_addr));

    bzero(buffer , BUFFER_SIZE);
    sprintf(buffer ,"%d" ,num1);
```

```c
    sendto(socketfd,buffer, strlen(buffer) + 1, 0, (struct sockaddr*)&host_addr, sizeof(host_addr));

    bzero(buffer , BUFFER_SIZE);
    sprintf(buffer ,"%d" ,num2);

    sendto(socketfd,buffer, strlen(buffer) + 1, 0, (struct sockaddr*)&host_addr, sizeof(host_addr));

    bzero(buffer , BUFFER_SIZE);

    recvbytes = recvfrom(socketfd, buffer, sizeof(buffer), 0, NULL, NULL);

    printf("%s\n",buffer);
    return EXIT_SUCCESS ;
}
```

OUTPUT :-

```
[s2019103573n@centos8-linux Tue Nov 30 06:01 PM ~]$ cd asses
[s2019103573n@centos8-linux Tue Nov 30 06:01 PM asses]$ gcc -lm server.c
[s2019103573n@centos8-linux Tue Nov 30 06:01 PM asses]$ ./a.out 3573
server file descriptor : 3
Multithreaded calculated server listening on : 3573
The server answer is 72
^C
[s2019103573n@centos8-linux Tue Nov 30 06:02 PM asses]$ gcc -lm server.c
[s2019103573n@centos8-linux Tue Nov 30 06:02 PM asses]$ ./a.out 3573
server file descriptor : 3
Multithreaded calculated server listening on : 3573
The server answer is 18
^C
[s2019103573n@centos8-linux Tue Nov 30 06:03 PM asses]$ gcc -lm server.c
[s2019103573n@centos8-linux Tue Nov 30 06:03 PM asses]$ ./a.out 3573
server file descriptor : 3
Multithreaded calculated server listening on : 3573
The server answer is 8
^C
[s2019103573n@centos8-linux Tue Nov 30 06:03 PM asses]$ gcc -lm server.c
[s2019103573n@centos8-linux Tue Nov 30 06:03 PM asses]$ ./a.out 3573
server file descriptor : 3
Multithreaded calculated server listening on : 3573
The server answer is 9
^C
[s2019103573n@centos8-linux Tue Nov 30 06:04 PM asses]$
```

```
[s2019103573n@centos8-linux Tue Nov 30 06:01 PM ~]$ cd asses
[s2019103573n@centos8-linux Tue Nov 30 06:02 PM asses]$ gcc -lm client.c
[s2019103573n@centos8-linux Tue Nov 30 06:02 PM asses]$ ./a.out 3573

1) Addition
2) Multiplication
3) Square Root
4) Division
5) EXIT

Enter Operation: 1
Enter number 1 : 27
Enter number 2 : 45
The server answer is 72
[s2019103573n@centos8-linux Tue Nov 30 06:02 PM asses]$ gcc -lm client.c
[s2019103573n@centos8-linux Tue Nov 30 06:02 PM asses]$ ./a.out 3573

1) Addition
2) Multiplication
3) Square Root
4) Division
5) EXIT

Enter Operation: 2
Enter number 1 : 3
Enter number 2 : 6
The server answer is 18
[s2019103573n@centos8-linux Tue Nov 30 06:03 PM asses]$ gcc -lm client.c
[s2019103573n@centos8-linux Tue Nov 30 06:03 PM asses]$ ./a.out 3573

1) Addition
2) Multiplication
3) Square Root
4) Division
5) EXIT

Enter Operation: 3
Enter number : 64
The server answer is 8
[s2019103573n@centos8-linux Tue Nov 30 06:03 PM asses]$ gcc -lm client.c
[s2019103573n@centos8-linux Tue Nov 30 06:04 PM asses]$ ./a.out 3573

1) Addition
2) Multiplication
3) Square Root
4) Division
5) EXIT

Enter Operation: 4
Enter number 1 : 81
Enter number 2 : 9
The server answer is 9
[s2019103573n@centos8-linux Tue Nov 30 06:04 PM asses]$
```

2.Implement the following sequences of a DNS system using UDP sockets: (15)

Show the flow of all the sequences in the console

ROOT DNS

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#define ROOTPORT 8041
extern int errno;
int main()
{
    int socketfd = 0, clientfd = 0, sentbytes, recvbytes;
    socklen_t length = sizeof(struct sockaddr_in);
    struct sockaddr_in host_addr, client_addr;
    char buffer[20];
    socketfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (socketfd < 0)
    {
        fprintf(stderr, "Error in socket creation.\n");
        return -1;
    }
    host_addr.sin_family = AF_INET;
    host_addr.sin_port = htons(ROOTPORT);
    inet_pton(AF_INET, "127.0.0.1", &host_addr.sin_addr);
    if (bind(socketfd, (struct sockaddr *)&host_addr,sizeof(host_addr)) < 0)
    {
        fprintf(stderr, "Error in binding port tosocket.\n");
        return -1;
    }
    printf("ROOT DNS RESOLVER STARTED AT PORT :%d\n",ROOTPORT);
    while(1) {
        recvbytes = recvfrom(socketfd, buffer,
                             sizeof(buffer), 0, (struct sockaddr *)&client_addr, &length);
        fprintf(stdout, "DNS QUERY : %s\n", buffer);
        FILE *fd = fopen("rootdns.txt", "r");
        if (!fd)
        {
            fprintf(stderr, "Could not access DNS records.\n");
            sendto(socketfd, "ERROR", strlen("ERROR")+ 1, 0, (struct sockaddr*)&client_addr, length);
            continue;
        }
        char linebuff[40], filebuff[400], ip[20],tempbuff[40], lastbuff[40];
        char *temp, *iptemp;
        int flag = 0, i;
        linebuff[0] = '\0';
        lastbuff[0] = '\0';
        filebuff[0] = '\0';
        ip[0] = '\0';
        while (fgets(linebuff, sizeof(linebuff), fd))
        {
            strcpy(tempbuff, linebuff);
            temp = strtok(tempbuff, " ");
            if (flag == 0 && strncmp(temp, buffer,
                                     strlen(temp)) == 0)
            {
                flag = 1;
                strcpy(lastbuff, linebuff);
                iptemp = strtok(NULL, "\n");
                for (i = 0; *iptemp != '\0'; i++,
                    iptemp++)
                    ip[i] = *iptemp;
                ip[i] = '\0';
```

```
            }
            else
            {
                strcat(filebuff, linebuff);
            }
        }
        fclose(fd);
        if (flag == 0)
        {
            sentbytes = sendto(socketfd, "404",
                               strlen("404") + 1, 0, (struct sockaddr *)&client_addr, length);
        }
        else
        {
            int fdes = open("rootdns.txt", O_WRONLY);
            strcat(filebuff, lastbuff);
            write(fdes, filebuff, strlen(filebuff));
            close(fdes);
            fprintf(stdout, "TOP LEVEL DOMAIN IP :%s\n\n", ip);
            sentbytes = sendto(socketfd, ip,
            strlen(ip) + 1, 0, (struct sockaddr*)&client_addr, length);
        }
    }
    close(socketfd);
    return 0;
}
```

TOP LEVEL DOMAIN

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#define TLDPORT 8042
extern int errno;
int main()
{
    int socketfd = 0, clientfd = 0, sentbytes, recvbytes;
    socklen_t length = sizeof(struct sockaddr_in);
    struct sockaddr_in host_addr, client_addr;
    char buffer[20];
    socketfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (socketfd < 0)
    {
        fprintf(stderr, "Error in socket creation.\n");
        return -1;
    }
    host_addr.sin_family = AF_INET;
    host_addr.sin_port = htons(TLDPORT);
    inet_pton(AF_INET, "127.0.0.1", &host_addr.sin_addr);
    if (bind(socketfd, (struct sockaddr *)&host_addr,sizeof(host_addr)) < 0)
    {
        fprintf(stderr, "Error in binding port to socket.\n");
        return -1;
    }
    printf("TOP LEVEL DOMAIN SERVER FOR EDU STARTED AT %d\n", TLDPORT);
    while (1)
    {
        recvbytes = recvfrom(socketfd, buffer,
                             sizeof(buffer), 0, (struct sockaddr *)&client_addr, &length);
        fprintf(stdout, "DNS QUERY : %s\n", buffer);
        FILE *fd = fopen("tlddns.txt", "r");
```

```
        if (!fd)
        {
            fprintf(stderr, "Could not access DNS records.\n");
            sendto(socketfd, "ERROR", strlen("ERROR") + 1, 0, (struct sockaddr*)&client_addr, length);
            continue;
        }
        char linebuff[40], filebuff[400], ip[20],
            tempbuff[40], lastbuff[40];
        char *temp, *iptemp;
        int flag = 0, i;
        linebuff[0] = '\0';
        lastbuff[0] = '\0';
        filebuff[0] = '\0';
        ip[0] = '\0';
        while (fgets(linebuff, sizeof(linebuff), fd))
        {
            strcpy(tempbuff, linebuff);
            temp = strtok(tempbuff, " ");
            if (flag == 0 && strncmp(temp, buffer,
                                     strlen(temp)) == 0)
            {
                flag = 1;
                strcpy(lastbuff, linebuff);
                iptemp = strtok(NULL, "\n");
                for (i = 0; *iptemp != '\0'; i++,
                    iptemp++)
                    ip[i] = *iptemp;
                ip[i] = '\0';
            }
            else
            {
                strcat(filebuff, linebuff);
            }
        }
        fclose(fd);
        if (flag == 0)
        {
            sentbytes = sendto(socketfd, "404",
                               strlen("404") + 1, 0, (struct sockaddr *)&client_addr, length);
        }
        else
        {
            int fdes = open("tlddns.txt", O_WRONLY);
            strcat(filebuff, lastbuff);
            write(fdes, filebuff, strlen(filebuff));
            close(fdes);
            fprintf(stdout, "AUTHORITATIVE SERVER IP: %s\n\n", ip);
            sentbytes = sendto(socketfd, ip,strlen(ip) + 1, 0, (struct sockaddr*)&client_addr, length);
        }
    }
    close(socketfd);
    return 0;
}
```

AUTHORATATIVE SERVER

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#define IPLOOKUP_TABLE_COUNT 4
#define IP_FOR_EACH_DNS_RECORDS 3
#define AUTHPORT 8043
extern int errno;
typedef struct
{
    char *key;
    int value;
} keyValuePairs;
keyValuePairs ip_lookuptable[] = {
    {"zomato.co.in", 0},
    {"customer.zomato.co.in", 0},
};
int rotate_dns_ip(char *domain_name)
{
    for (int i = 0; i < IPLOOKUP_TABLE_COUNT; i++)
    {
        if (strcmp(domain_name, ip_lookuptable[i].key) == 0)
        {
            int value = ip_lookuptable[i].value;
            ip_lookuptable[i].value++;
            return value;
        }
    }
    return -1;
}
int main()
{
    int socketfd = 0, clientfd = 0, sentbytes, recvbytes;
    socklen_t length = sizeof(struct sockaddr_in);
    struct sockaddr_in host_addr, client_addr;
    char buffer[20];
    socketfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (socketfd < 0)
    {
        fprintf(stderr, "Error in socket creation.\n");
        return -1;
    }
    host_addr.sin_family = AF_INET;
    host_addr.sin_port = htons(AUTHPORT);
    inet_pton(AF_INET, "127.0.0.1", &host_addr.sin_addr);
    if (bind(socketfd, (struct sockaddr *)&host_addr,sizeof(host_addr)) < 0)
    {
        fprintf(stderr, "Error in binding port to socket.\n");
        return -1;
    }
printf("AUTHORITATIVE DNS SERVER FOR CO.IN STARTED AT PORT : %d\n",AUTHPORT);
while(1) {
        recvbytes = recvfrom(socketfd, buffer,
                             sizeof(buffer), 0, (struct sockaddr *)&client_addr, &length);
        fprintf(stdout, "DNS QUERY : %s\n", buffer);
        FILE *fd = fopen("authdns.txt", "r");
        if (!fd)
        {
            fprintf(stderr, "Could not access DNS records.\n");
            sendto(socketfd, "ERROR", strlen("ERROR")+ 1, 0, (struct sockaddr*)&client_addr, length);
            continue;
        }
        char linebuff[80], filebuff[400], ip[40],
            tempbuff[80], lastbuff[80];
```

```
        char *temp, *iptemp;
        int flag = 0, i;
        linebuff[0] = '\0';
        lastbuff[0] = '\0';
        filebuff[0] = '\0';
        ip[0] = '\0';
        while (fgets(linebuff, sizeof(linebuff), fd))
        {
            strcpy(tempbuff, linebuff);
            temp = strtok(tempbuff, " ");
            if (flag == 0 && strncmp(temp, buffer,
                                     strlen(temp)) == 0)
            {
                flag = 1;
                strcpy(lastbuff, linebuff);
                iptemp = strtok(NULL, " ");
                int counter = 0;
                int curr_pointer =
                    rotate_dns_ip(buffer) % IP_FOR_EACH_DNS_RECORDS;
                int i = 0;
                while (1)
                {
                    for (i = 0; *iptemp != ' ' &&
                                *iptemp != '\0';
                         i++, iptemp++)
                        ip[i] = *iptemp;
                    if (*iptemp == '\n' || counter == curr_pointer)
                        break;
                    counter++;
                    iptemp = strtok(NULL, " ");
                }
                ip[i] = '\0';
            }
            else
            {
                strcat(filebuff, linebuff);
            }
        }
        fclose(fd);
        if (flag == 0)
        {
            sentbytes = sendto(socketfd, "404",
                               strlen("404") + 1, 0, (struct sockaddr *)&client_addr, length);
        }
        else
        {
            int fdes = open("authdns.txt", O_WRONLY);
            strcat(filebuff, lastbuff);
            write(fdes, filebuff, strlen(filebuff));
            close(fdes);
            fprintf(stdout, "AUTHORITATIVE SERVER IP: %s\n\n", ip);
            sentbytes = sendto(socketfd, ip,strlen(ip) + 1, 0, (struct sockaddr*)&client_addr, length);
        }
}
close(socketfd);
return 0;
}
```

LOCAL DNS

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#define ROOTPORT 8041
#define TLDPORT 8042
#define AUTHPORT 8043
#define PORT 8044
int main()
{
    int socketfd = 0, localfd = 0;
    int rootfd = 0, tldfd = 0, authfd = 0;
    socklen_t length = sizeof(struct sockaddr_in);
    struct sockaddr_in host_addr, root_addr, tld_addr,
        auth_addr, client_addr;
    char buffer[512], root[20], tld[30], auth[100];
    char rootip[30], tldip[30], authip[30];
    int recvbytes, sentbytes;
    socketfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (socketfd < 0)
    {
        fprintf(stderr, "Error in socket creation.\n");
        return -1;
    }
    host_addr.sin_family = AF_INET;
    host_addr.sin_port = htons(PORT);
    inet_pton(AF_INET, "127.0.0.1", &host_addr.sin_addr);
    if (bind(socketfd, (struct sockaddr *)&host_addr,
             sizeof(host_addr)) < 0)
    {
        fprintf(stderr, "Error in binding port to socket.\n");
        return -1;
    }
    printf("LOCAL DNS STARTED AT PORT : %d\n", PORT);
    while (1)
    {
        recvbytes = recvfrom(socketfd, buffer,
                             sizeof(buffer), 0, (struct sockaddr *)&client_addr, &length);
        if (strncmp(buffer, "exit", sizeof("exit")) == 0)
        {
            fprintf(stdout, "exiting");
            break;
        }
        printf("\n-----------------------------------------\n");
        fprintf(stdout, "Request from client : %s\n",
                buffer);
        strcpy(auth, buffer);
        int i = 0, j = 0, k = 0;
        while (buffer[i++] != '.')
            ;
        while (buffer[i] != '.')
        {
            tld[j++] = buffer[i++];
        }
        tld[j++] = buffer[i++];
        while (buffer[i] != ' ' && buffer[i] != '\0')
        {
            tld[j++] = buffer[i];
            root[k++] = buffer[i];
            i++;
        }
        tld[j] = '\0';
        root[k] = '\0';
        fprintf(stdout, "\t\t[RESOLVING DNS QUERY]\n\n");
```

```
        rootfd = socket(AF_INET, SOCK_DGRAM, 0);
        if (rootfd < 0)
        {
            fprintf(stderr, "Error in socket creation.\n");
            return -1;
        }
        root_addr.sin_family = AF_INET;
        root_addr.sin_port = htons(ROOTPORT);
        inet_pton(AF_INET, "127.0.0.1",&root_addr.sin_addr);
        sentbytes = sendto(rootfd, root, strlen(root) + 1, 0, (struct sockaddr *)&root_addr, length);
        recvbytes = recvfrom(rootfd, rootip,sizeof(rootip), 0, NULL, NULL);
        fprintf(stdout, "[ROOT DNS SERVER]\n\n");
        fprintf(stdout, "TLD server IP for %s:%s\n\t|\n\t|\n\t↓\n", root, rootip);
        close(rootfd);
        tldfd = socket(AF_INET, SOCK_DGRAM, 0);
        if(tldfd < 0) {
        fprintf(stderr, "Error in socket creation.\n");
        return -1;
    }
        tld_addr.sin_family = AF_INET;
        tld_addr.sin_port = htons(TLDPORT);
        inet_pton(AF_INET, "127.0.0.1",&tld_addr.sin_addr);
        sentbytes = sendto(tldfd, tld, strlen(tld) + 1,0, (struct sockaddr*)&tld_addr, length);
        recvbytes = recvfrom(tldfd, tldip, sizeof(tldip),0, NULL, NULL);
        fprintf(stdout,"[TLD SERVER]\n\n");
        fprintf(stdout, "Auth server IP for %s:%s\n\t|\n\t|\n\t↓\n", tld, tldip);
        close(tldfd);
        authfd = socket(AF_INET, SOCK_DGRAM, 0);
        if(authfd < 0) {
        fprintf(stderr, "Error in socket creation.\n");
        return -1;
    }
        auth_addr.sin_family = AF_INET;
        auth_addr.sin_port = htons(AUTHPORT);
        inet_pton(AF_INET, "127.0.0.1",&auth_addr.sin_addr);
        sentbytes = sendto(authfd, auth, strlen(auth) +1, 0, (struct sockaddr*)&auth_addr, length);
        recvbytes = recvfrom(authfd, authip,
        sizeof(authip), 0, NULL, NULL);
        fprintf(stdout,"[AUTHORITATIVE SERVER]\n\n");
        if(strcmp(authip,"404") == 0)
            fprintf(stdout,"DNS
        RECORDS NOT FOUND : %s \n",auth);
        else
            fprintf(stdout, "Server ip for %s: %s\n\n",
        auth, authip);
        close(authfd);
        sentbytes = sendto(socketfd, authip,strlen(authip) + 1, 0, (struct sockaddr*)&client_addr,
length);
    }
    close(socketfd);
    return 0;
}
```

CLIENT

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define LOCALDNS 8044
int main()
{
    int socketfd = 0, sentbytes, recvbytes;
    struct sockaddr_in host_addr;
    char input[20], buffer[20];
    socketfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (socketfd < 0)
    {
        fprintf(stderr, "Error in socket creation.\n");
        return -1;
    }
    host_addr.sin_family = AF_INET;
    host_addr.sin_port = htons(LOCALDNS);
    inet_pton(AF_INET, "127.0.0.1", &host_addr.sin_addr);
    while (1)
    {
        fprintf(stdout, "\n[+]Enter the HostName: ");
        scanf("%s", input);
        sentbytes = sendto(socketfd, input, strlen(input) + 1, 0, (struct sockaddr *)&host_addr,
sizeof(host_addr));
        if (strncmp(input, "exit", sizeof("exit")) == 0)
            break;
        recvbytes = recvfrom(socketfd, buffer,
                             sizeof(buffer), 0, NULL, NULL);
        if (strcmp("404", buffer) == 0)
        {
            printf("DNS RECORDS NOT FOUND FOR %s\n",input);
        }
        else
            printf("SERVER IP OF %s : %s\n", input,buffer);
    }
    close(socketfd);
    return 0;
}
```

OUTPUT :-

Client

```
[s2019103573n@centos8-linux Tue Nov 30 06:17 PM as]$ gcc client.c -o client
[s2019103573n@centoss-linux Tue Nov 30 06:18 PM as]$ ./client
[+]Enter the HostName: zomato.co.in
SERVER IP OF zomato.co.in : 132.55.4.1
```

localDNS

```
[s2019103573n@centos8-linux Tue Nov 30 06:18 PM as]$ gcc localDNS.c -o localDNS
[s2019103573n@centos8-linux Tue Nov 30 06:18 PM as]$ ./localDNS
LOCAL DNS STARTED AT PORT : 3044
Request from client : zomato.co.in
[RESOLVING DNS QUERY]
[ROOT DNS SERVER]
TLD server IP for in: 12.11.80.6
        |
        |
        |
[TLD SERVER]

Auth server IP for co.in: 132.55.4.3
        |
        |
        |
[AUTHORITATIVE SERVER]
Server ip for zomato.co.in: 132.55.4.1
```

rootDNS

```
[s2019103573n@centos8-linux Tue Nov 30 06:15 PM as]$ gcc rootDNS.c -o rootDNS
[s2019103573n@centos8-linux Tue Nov 30 06:15 PM as]$ ./rootDNS
ROOT DNS RESOLVER STARTED AT PORT : 3041
DNS QUERY : in

[s2019103573n@centos8-linux Tue Nov 30 06:16 PM as]$ gcc rootDNS.c -o rootDNS
[s2019103573n@centos8-linux Tue Nov 30 06:18 PM as]$ ./rootDNS
ROOT DNS RESOLVER STARTED AT PORT : 3041
DNS QUERY : in
TOP LEVEL DOMAIN IP : 12.11.80.6
```

toplevelDomain

```
[s2019103573n@centos8-linux Tue Nov 30 06:16 PM as]$ gcc toplevelDomain.c -o toplevelDomain
[s2019103573n@centos8-linux Tue Nov 30 06:18 PM as]$ ./toplevelDomain
TOP LEVEL DOMAIN SERVER FOR EDU STARTED AT 3042
DNS QUERY : co.in
AUTHORITATIVE SERVER IP : 132.55.4.3
```

authDNS

```
[s2019103573n@centos8-linux Tue Nov 30 06:16 PM as ]$ gcc authServer.c -o authServer
[s2019103573n@centos8-linux Tue Nov 30 06:18 PM as ]$ ./authServer
AUTHORITATIVE DNS SERVER FOR CO.IN STARTED AT PORT : 3043
DNS QUERY : zomato.co.in
```

DNS records :

```
[s2019103573n@centos8-linux Tue Nov 30 07:03 PM lab]$ cat rootdns.txt
org 6.32.1.9
com 10.32.1.4
edu 8.1.32.1
12.11.80.6
[s2019103573n@centos8-linux Tue Nov 30 07:03 PM lab]$ cat tlddns.txt
srm.edu 185.219.1.1
iit.edu 142.139.161.1
nit.edu 143.139.161.1
annauniv.edu 144.139.161.1
co.in 132.55.4.3
[s2019103573n@centos8-linux Tue Nov 30 07:03 PM lab]$ cat authdns.txt
customer.zomato.co.in 132.55.4.4
zomato.co.in 132.55.4.3
```