**30/11/2021 CN ASSESMENT 2019103573**

**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**

#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);

    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**

#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);

    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));

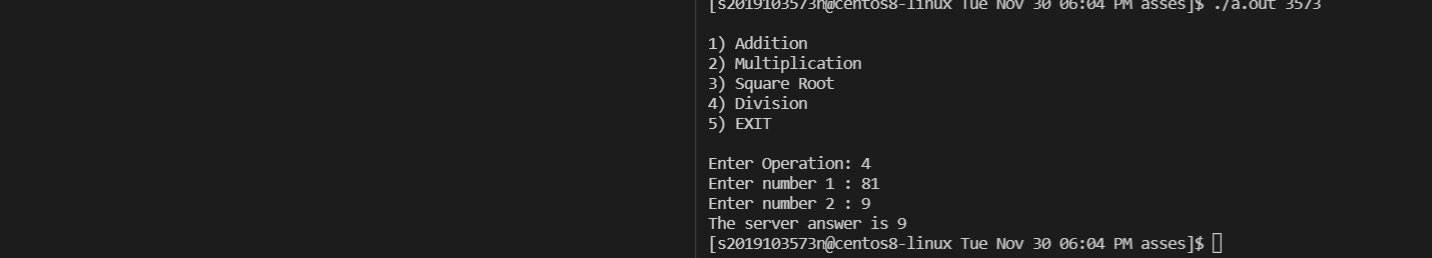
    bzero(buffer , BUFFER\_SIZE);

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

    printf("%s\n",buffer);

    return EXIT\_SUCCESS ;

}

**OUTPUT :-**

**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**

#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**

#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**

#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);

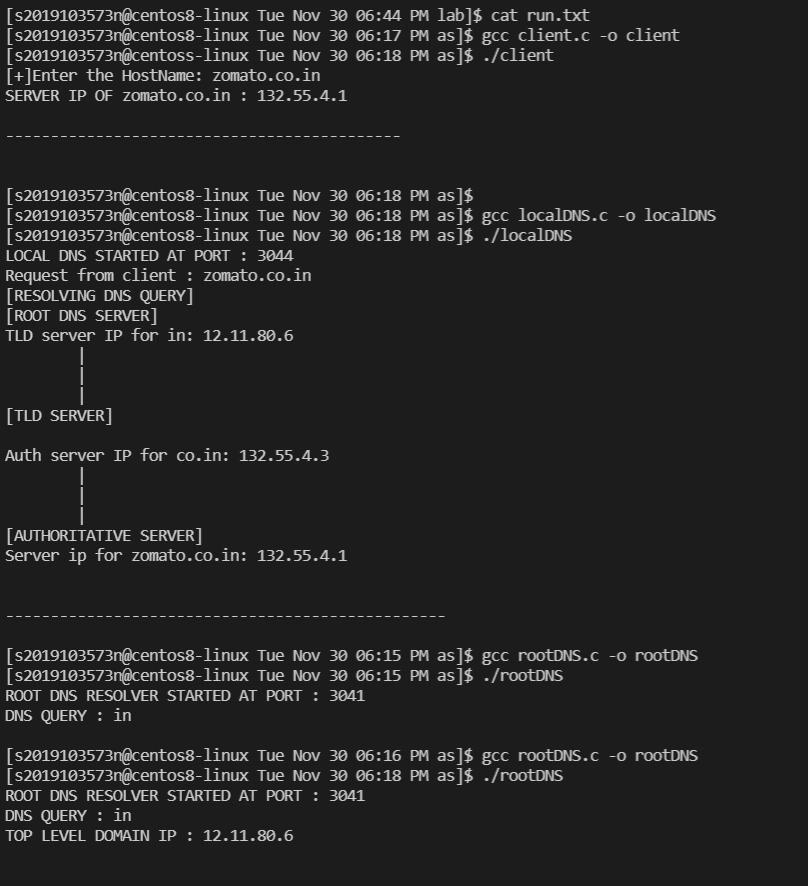
    }

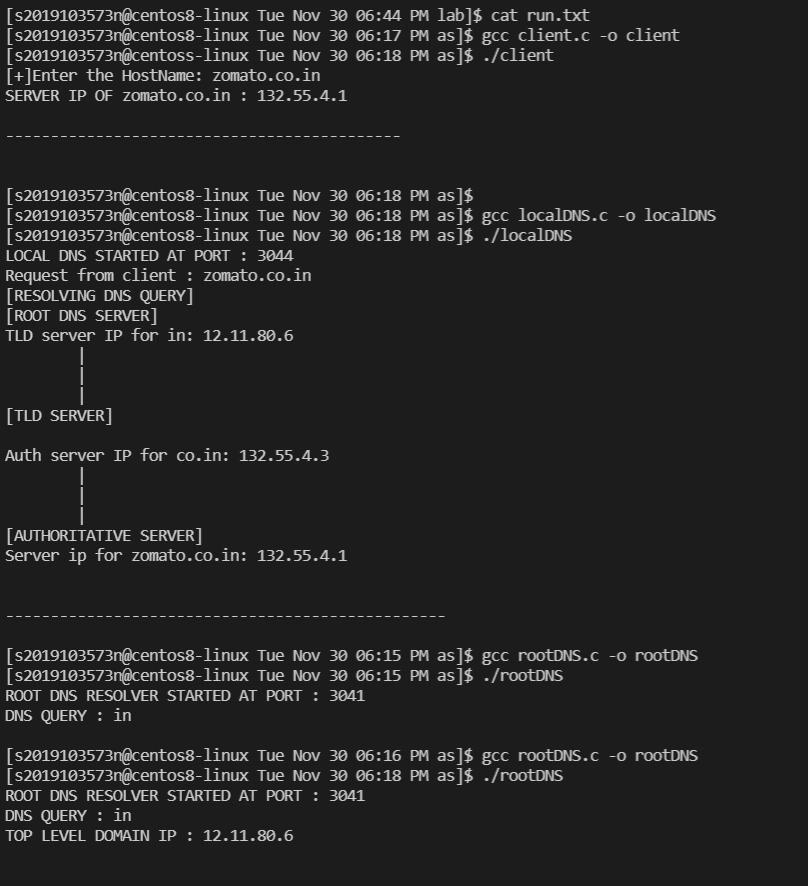
    close(socketfd);

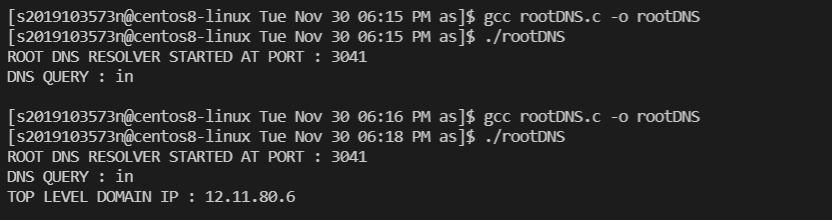
    return 0;

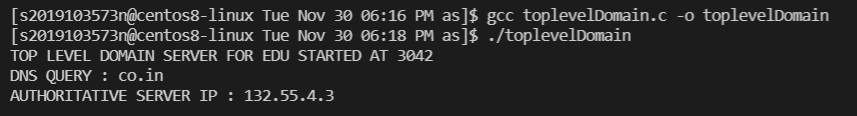
}

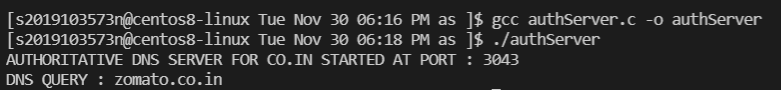
**OUTPUT :-**

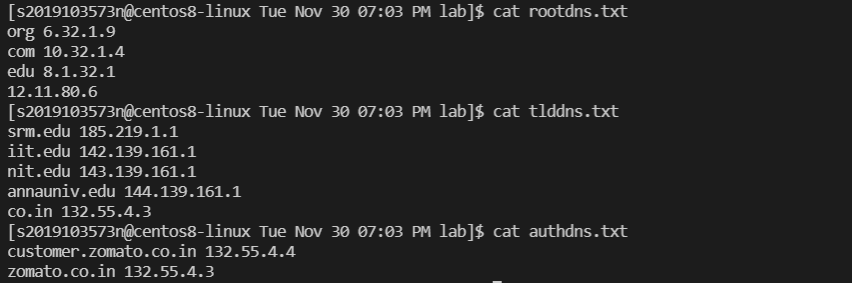
**Client**

**localDNS**

**rootDNS**

**toplevelDomain**

**authDNS**

**DNS records :**