20/10/2021 **CN LAB 6** 2019103573

IMPLEMENTING THE DNS ITERATIVE

ROOT DNS SERVER

```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 to socket.\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, "REQUEST FROM : %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

```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 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 %d\n", TLDPORT);

    while (1)
    {
        recvbytes = recvfrom(socketfd, buffer, sizeof(buffer), 0, (struct sockaddr*)
        &client_addr, &length);
        fprintf(stdout, "REQUEST FROM CLIENT : %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;
}
```

AUTHORIZATION SERVER CODE

```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 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[] = {
{"www.cricbuzz.com", 0},
{"mail.google.com", 0},
{"cric.cricbuzz.com", 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 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

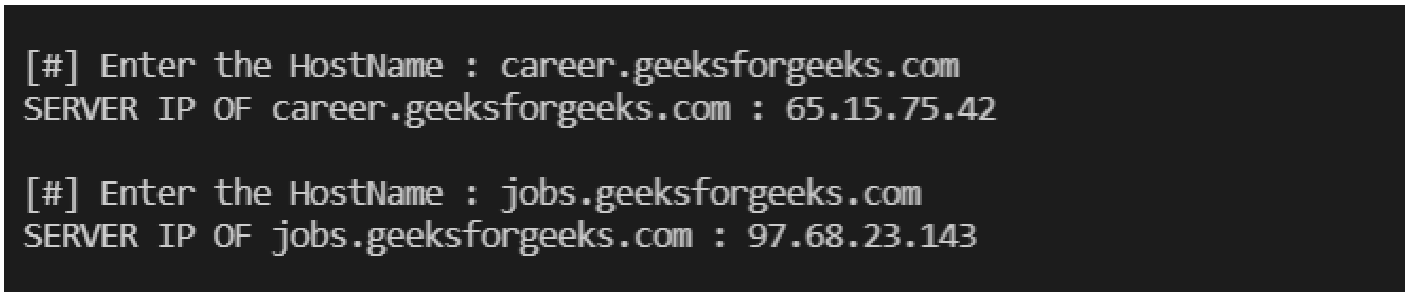
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>

#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 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;
        }

        fprintf(stdout, "Request from client : %s\n\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\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\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;
}
```

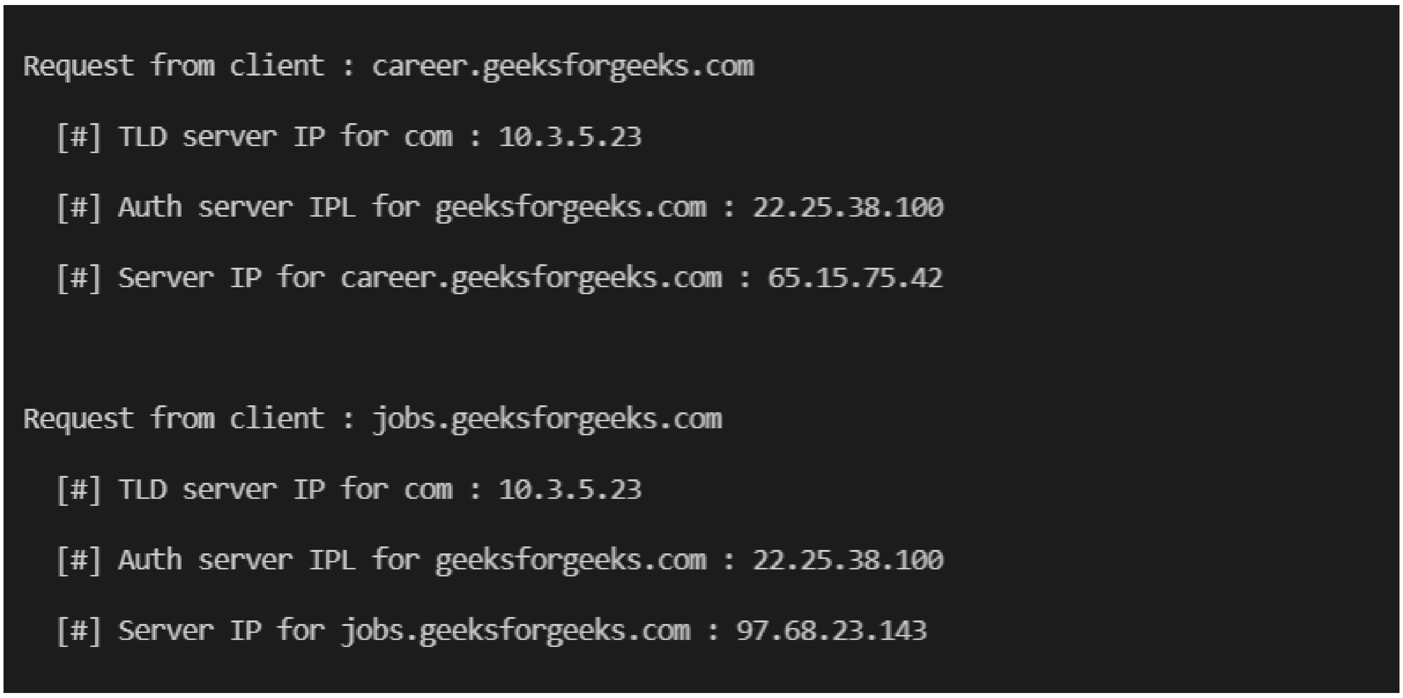
OUTPUT :-

CLIENT



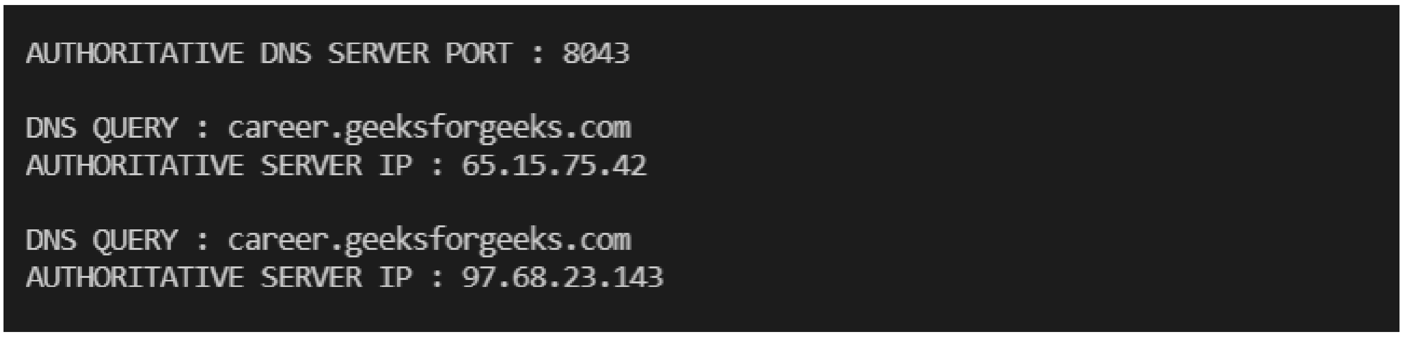
```
[#] Enter the HostName : career.geeksforgeeks.com
SERVER IP OF career.geeksforgeeks.com : 65.15.75.42

[#] Enter the HostName : jobs.geeksforgeeks.com
SERVER IP OF jobs.geeksforgeeks.com : 97.68.23.143
```

LOCAL DNS



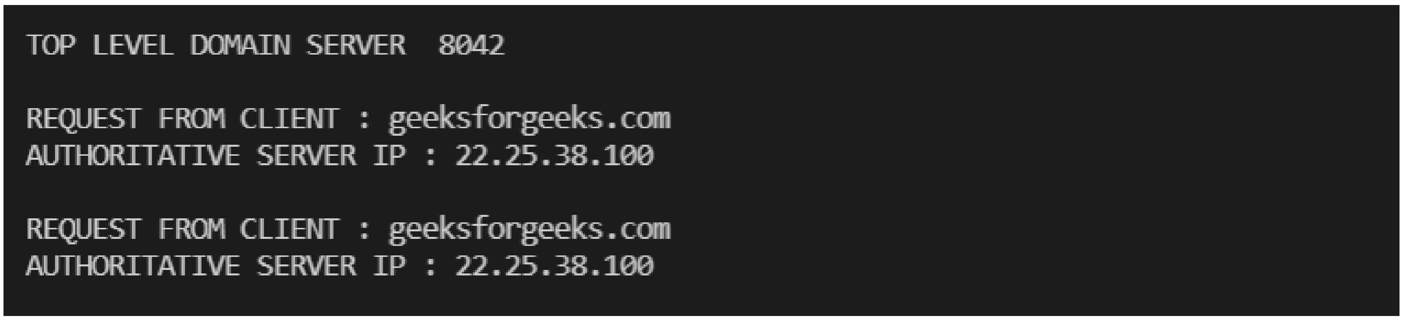
```
Request from client : career.geeksforgeeks.com

  [#] TLD server IP for com : 10.3.5.23

  [#] Auth server IPL for geeksforgeeks.com : 22.25.38.100

  [#] Server IP for career.geeksforgeeks.com : 65.15.75.42


Request from client : jobs.geeksforgeeks.com

  [#] TLD server IP for com : 10.3.5.23

  [#] Auth server IPL for geeksforgeeks.com : 22.25.38.100

  [#] Server IP for jobs.geeksforgeeks.com : 97.68.23.143
```

AUTHORIZATION



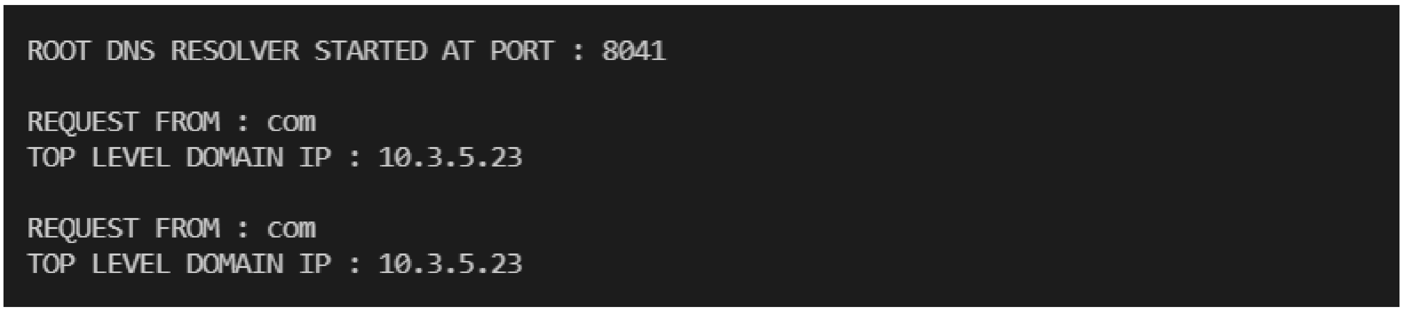
```
AUTHORITATIVE DNS SERVER PORT : 8043

DNS QUERY : career.geeksforgeeks.com
AUTHORITATIVE SERVER IP : 65.15.75.42

DNS QUERY : career.geeksforgeeks.com
AUTHORITATIVE SERVER IP : 97.68.23.143
```

TOP LEVEL DOMAIN



```
TOP LEVEL DOMAIN SERVER  8042

REQUEST FROM CLIENT : geeksforgeeks.com
AUTHORITATIVE SERVER IP : 22.25.38.100

REQUEST FROM CLIENT : geeksforgeeks.com
AUTHORITATIVE SERVER IP : 22.25.38.100
```

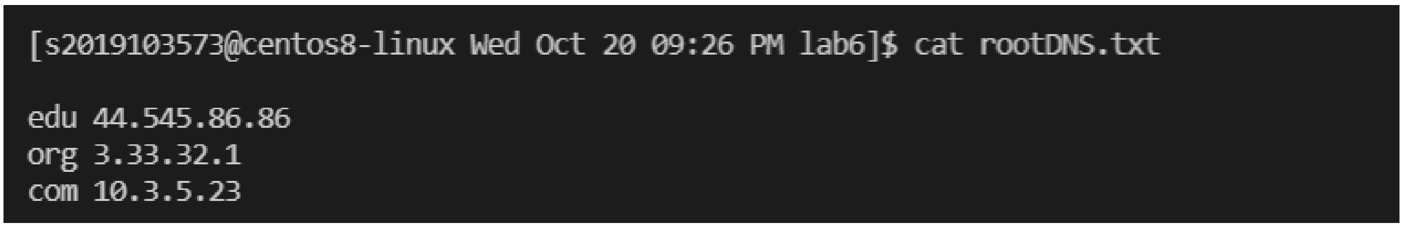
ROOT



```
ROOT DNS RESOLVER STARTED AT PORT : 8041

REQUEST FROM : com
TOP LEVEL DOMAIN IP : 10.3.5.23

REQUEST FROM : com
TOP LEVEL DOMAIN IP : 10.3.5.23
```

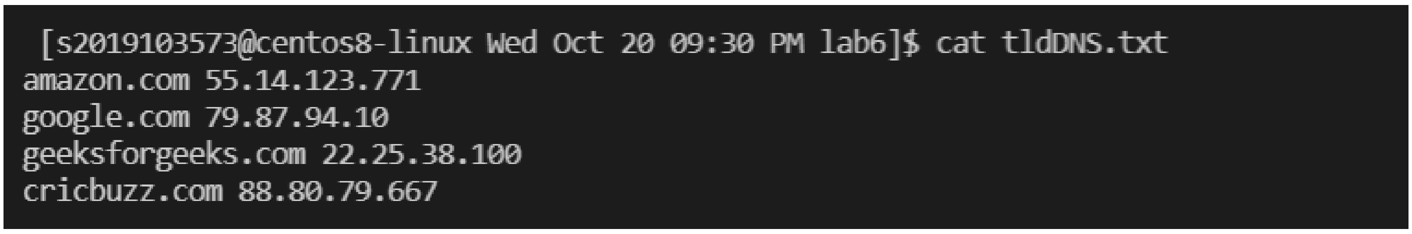
TEXT FILES

ROOT - rootDNS.txt



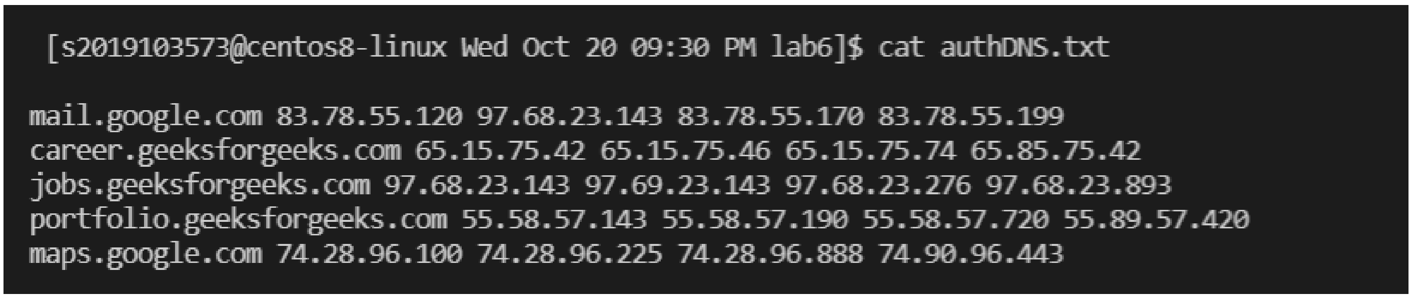
```
[s2019103573@centos8-linux Wed Oct 20 09:26 PM lab6]$ cat rootDNS.txt

edu 44.545.86.86
org 3.33.32.1
com 10.3.5.23
```

TLP - tldDNS.txt



```
 [s2019103573@centos8-linux Wed Oct 20 09:30 PM lab6]$ cat tldDNS.txt
amazon.com 55.14.123.771
google.com 79.87.94.10
geeksforgeeks.com 22.25.38.100
cricbuzz.com 88.80.79.667
```

AUTHORIZATION - authDNS.txt



```
 [s2019103573@centos8-linux Wed Oct 20 09:30 PM lab6]$ cat authDNS.txt

mail.google.com 83.78.55.120 97.68.23.143 83.78.55.170 83.78.55.199
career.geeksforgeeks.com 65.15.75.42 65.15.75.46 65.15.75.74 65.85.75.42
jobs.geeksforgeeks.com 97.68.23.143 97.69.23.143 97.68.23.276 97.68.23.893
portfolio.geeksforgeeks.com 55.58.57.143 55.58.57.190 55.58.57.720 55.89.57.420
maps.google.com 74.28.96.100 74.28.96.225 74.28.96.888 74.90.96.443
```