**SSN College of Engineering, Kalavakkam**

**Department of Computer Science and Engineering**

**UCS1511 NETWORKS LAB**

**Exercise 5: DNS Lookup Using UDP**

**Name :** Kshitij Sharma **Roll No. :** 185001080 **Date :** 27/09/2020

**Aim:**

To simulate the concept of Domain Name Server using UDP..

# Algorithm

## Server

* 1. Initialise DNS table with URL and corresponding IP addresses.
  2. Prompt admin to add new URLs and IP after verifying valid IP address and ensuring it is not duplicate.
  3. Create a socket descriptor with **socket()** system call with AF INET (IPV4 domain), SOCK DGRAM (for UDP protocol), default protocol and store as sockfd.
  4. If sockfd is a negative number, socket creation failed, end program.
  5. Create sockaddr in object to assign IP address and Port number for socket. Set family to AF INET, IP address to INADDR ANY to accept connections from any client and port number required.
  6. Bind newly created socket to addresss given in sockaddr in.
  7. If bind is non zero, bind failed, print error message and terminate.
  8. Read URL requested into buffer using **recvfrom()** system call.
  9. LOOP through DNS table:
     + IF domain name matches requested URL, copy all mapped IP addresses into buffer
     + ELSE write ‘URL not Found Error!’ onto buffer
  10. Send response to client using **sendto()** system call, goto step 8.
  11. Close connections on socket using **close()** and terminate program.

## Client

* 1. Create a socket descriptor with **socket()** system call with AF INET (IPV4 domain), SOCK DGRAM, default protocol and store as sockfd.
  2. If sockfd is a negative number, socket creation failed, end program.
  3. Create sockaddr in object to assign IP address and Port number for socket. Set family to AF INET, IP address to localhost(127.0.0.1) to connect to server and port number required.
  4. BEGIN LOOP
     + Read URL from use onto buffer.
     + IF input is ‘END’ terminate program.
     + Send the URL to server using **sendto()** system call.
     + Read the response from server onto buffer using **recvfrom()** system call.
     + IF response is ‘URL not Found Error!’, print error message to user
     + ELSE, display all received addresses to user.
  5. Close the connections on socket using **close()** and terminate program.

# Program

## Server Side

## #include <stdio.h>

## #include <netdb.h>

## #include <fcntl.h>

## #include <netinet/in.h>

## #include <stdlib.h>

## #include <string.h>

## #include <sys/socket.h>

## #include <sys/types.h>

## #define MAX 1024

## #define MAX\_ADDR 10

## #define MAX\_DOMAIN 20

## typedef char string[30];

## typedef struct Entry

## {

## string domain;

## string address[MAX\_ADDR];

## } Entry;

## int is\_IpTaken(Entry table[MAX\_DOMAIN], char \* address){

## for(int i=0;i<MAX\_DOMAIN;++i){

## for (int j = 0; j < MAX\_ADDR; ++j)

## {

## if(table[i].address[j][0] && strcmp(table[i].address[j],address)==0){

## printf("IP address already taken\n");

## return 0;

## }

## }

## }

## return 1;

## }

## int validate\_number(char \*str) {

## while (\*str) {

## if(!isdigit(\*str)){ //if the character is not a number, return false

## return 0;

## }

## str++; //point to next character

## }

## return 1;

## }

## int validate\_ip(char \*ip) { //check whether the IP is valid or not

## int i, num, dots = 0;

## char \*ptr;

## if (ip == NULL)

## return 0;

## ptr = strtok(ip, "."); //cut the string using dor delimiter

## if (ptr == NULL)

## return 0;

## while (ptr) {

## if (!validate\_number(ptr)) //check whether the sub string is holding only number or not

## return 0;

## num = atoi(ptr); //convert substring to number

## if (num >= 0 && num <= 255) {

## ptr = strtok(NULL, "."); //cut the next part of the string

## if (ptr != NULL)

## dots++; //increase the dot count

## } else

## return 0;

## }

## if (dots != 3) //if the number of dots are not 3, return false

## return 0;

## return 1;

## }

## int createEntry(Entry table[MAX\_DOMAIN], char \*domain, char \*address)

## {

## 

## int index = -1;

## int flag = 0;

## for (int i = 0; i < MAX\_DOMAIN; i++)

## {

## if (strcmp(table[i].domain, domain) == 0)

## {

## for (int j = 0; j < MAX\_DOMAIN; j++)

## if (!table[i].address[j][0])

## {

## strcpy(table[i].address[j], address);

## flag = 1;

## break;

## }

## break;

## }

## if (!table[i].domain[0] && index == -1)

## index = i;

## }

## if (!flag)

## {

## strcpy(table[index].domain, domain);

## strcpy(table[index].address[0], address);

## flag = 1;

## }

## return flag;

## }

## Entry \*getAddress(Entry \*table, char \*const domain)

## {

## static Entry result;

## bzero(&result, sizeof(Entry));

## strcpy(result.domain, domain);

## for (int i = 0; i < MAX\_DOMAIN; i++)

## {

## if (strcmp(table[i].domain, domain) == 0)

## {

## for (int j = 0; j < MAX\_ADDR; j++)

## {

## strcpy(result.address[j], table[i].address[j]);

## }

## break;

## }

## }

## return (&result);

## }

## int main(int argc, char \*\*argv)

## {

## Entry table[MAX\_DOMAIN], \*result;

## bzero(table, MAX\_DOMAIN \* sizeof(Entry));

## int sockfd, new\_fd, len;

## struct sockaddr\_in servaddr, cliadrr;

## char buff[30];

## string domain, address;

## int n;

## sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

## if (sockfd == -1)

## {

## fprintf(stderr, "Error: Socket creation failed!\n");

## exit(EXIT\_FAILURE);

## }

## else

## printf("Socket creation successfull!\n");

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

## // assign IP, PORT

## servaddr.sin\_family = AF\_INET;

## servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

## servaddr.sin\_port = htons(8080);

## // Binding newly created socket to given IP and verification

## if ((bind(sockfd, (struct sockaddr \*)&servaddr, sizeof(servaddr))) != 0)

## {

## fprintf(stderr, "Error: Socket bind failed!\n");

## exit(EXIT\_FAILURE);

## }

## else

## printf("Socket bind successfull\n");

## len = sizeof(cliadrr);

## createEntry(table, "google.com", "192.168.1.1");

## createEntry(table, "yahoo.com", "194.12.34.12");

## createEntry(table, "google.com", "17.10.23.123");

## printf("+-----------------+----------------------+\n");

## printf("| Domain Name | Address |\n");

## printf("+-----------------+----------------------+\n");

## for (int i = 0; i < MAX\_DOMAIN; i++)

## {

## if (table[i].domain[0])

## {

## printf("| %-15s | %-20s |\n", table[i].domain, table[i].address[0]);

## for (int j = 1; j < MAX\_ADDR && table[i].address[j][0]; j++)

## printf("| %-15s | %-20s |\n", "", table[i].address[j]);

## printf("+-----------------+----------------------+\n");

## }

## }

## printf("\n");

## char ch;

## char dn[100],ip[100];

## printf("Do you want to update the table y or n: ");

## scanf(" %c",&ch);

## while(ch == 'y')

## { int flag = 0;

## char x[100];

## printf("\nEnter the domain name: ");

## scanf(" %[^\n]", dn);

## while(flag == 0)

## {

## printf("Enter ip address: ");

## scanf(" %[^\n]", ip);

## strcpy(x,ip);

## flag = validate\_ip(x);

## 

## 

## if(flag == 0)

## {

## printf("IP address entered is wrong\n");

## }

## else

## {

## flag = is\_IpTaken(table,ip);

## }

## 

## }

## 

## createEntry(table,dn,ip);

## printf("\nDo you want to update the table y or n: ");

## scanf(" %c",&ch);

## }

## 

## printf("\nUpdated Table\n");

## printf("+-----------------+----------------------+\n");

## printf("| Domain Name | Address |\n");

## printf("+-----------------+----------------------+\n");

## for (int i = 0; i < MAX\_DOMAIN; i++)

## {

## if (table[i].domain[0])

## {

## printf("| %-15s | %-20s |\n", table[i].domain, table[i].address[0]);

## for (int j = 1; j < MAX\_ADDR && table[i].address[j][0]; j++)

## printf("| %-15s | %-20s |\n", "", table[i].address[j]);

## printf("+-----------------+----------------------+\n");

## }

## }

## printf("\n Waiting for Client...");

## 

## while (1)

## {

## recvfrom(sockfd, buff, sizeof(buff), MSG\_WAITALL, (struct sockaddr \*)&cliadrr, &len);

## result = getAddress(table, buff);

## sendto(sockfd, result, sizeof(Entry), MSG\_CONFIRM, (struct sockaddr \*)&cliadrr, len);

## }

## close(sockfd);

}

## Client Side

## #include <netdb.h>

## #include <stdio.h>

## #include <stdlib.h>

## #include <fcntl.h>

## #include <string.h>

## #include <sys/socket.h>

## #define SA struct sockaddr

## #define MAX 1024

## #define MAX\_ADDR 10

## #define MAX\_DOMAIN 20

## typedef char string[30];

## typedef struct Entry

## {

## string domain;

## string address[MAX\_ADDR];

## } Entry;

## int main(int argc, char \*\*argv)

## {

## 

## Entry query;

## int sockfd, connfd;

## struct sockaddr\_in servaddr, cli;

## char buff[30] = {0};

## sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

## if (sockfd == -1)

## {

## fprintf(stderr, "Error: Socket creation failed!\n");

## exit(EXIT\_FAILURE);

## }

## else

## printf("Socket creation successfull!\n");

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

## // assign IP, PORT

## servaddr.sin\_family = AF\_INET;

## servaddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

## servaddr.sin\_port = htons(8080);

## int len = sizeof(Entry);

## while(1)

## {

## bzero(&query, sizeof(Entry));

## printf("Enter the domain name: ");

## scanf(" %[^\n]", query.domain);

## if (strcmp(query.domain, "END") == 0)

## break;

## sendto(sockfd, query.domain, sizeof(query.domain), MSG\_CONFIRM, (struct sockaddr \*)&servaddr, sizeof(servaddr));

## recvfrom(sockfd, &query, sizeof(Entry), MSG\_WAITALL, (struct sockaddr \*)&servaddr, &len);

## if (!query.address[0][0])

## printf("No entry in DNS!\n");

## else

## {

## printf("The IP Address is: \n");

## for (int i = 0; i < MAX\_ADDR; i++)

## {

## if (query.address[i][0])

## printf("%s\n", query.address[i]);

## }

## printf("\n");

## }

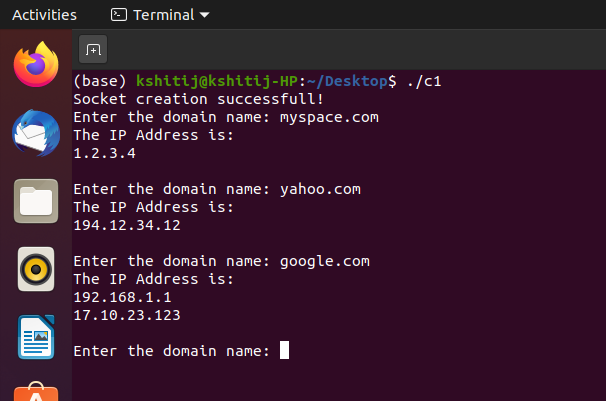
## }

## close(sockfd);

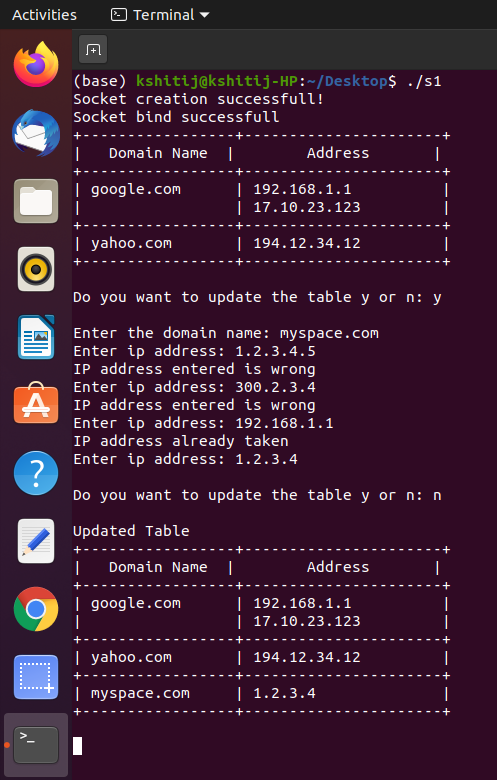
## }

# Output

# Client:



**Server:**



# Learning Outcomes:

* + We learn how to create a UDP client server connection.
  + We learn to maintain a table for performing DNS functions.
  + We learn to validate IP addresses.
  + We learn to work with connection-less protocol.