1. SERVER AND CLIENT

**Client.c**

#include <netdb.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <strings.h>

#include <sys/socket.h>

#include <unistd.h>

#define MAX 80

#define PORT 8888

#define SA struct sockaddr

void func(int sockfd)

{

char buff[MAX];

int n;

for (;;) {

bzero(buff, sizeof(buff));

printf("Enter the string : ");

n = 0;

while ((buff[n++] = getchar()) != '\n')

;

write(sockfd, buff, sizeof(buff));

bzero(buff, sizeof(buff));

read(sockfd, buff, sizeof(buff));

printf("From Server : %s", buff);

if ((strncmp(buff, "exit", 4)) == 0) {

printf("Client Exit...\n");

break;

}

}

}

int main()

{

int sockfd, connfd;

struct sockaddr\_in servaddr, cli;

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

if (sockfd == -1) {

printf("socket creation failed...\n");

exit(0);

}

else

printf("Socket successfully created..\n");

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

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

servaddr.sin\_port = htons(PORT);

if (connect(sockfd, (SA\*)&servaddr, sizeof(servaddr))

!= 0) {

printf("connection with the server failed...\n");

exit(0);

}

else

printf("connected to the server..\n");

func(sockfd);

close(sockfd);

}

**Server.c**

#include <stdio.h>

#include <netdb.h>

#include <netinet/in.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <unistd.h>

#define MAX 80

#define PORT 8888

#define SA struct sockaddr

void func(int connfd)

{

char buff[MAX];

int n;

for (;;) {

bzero(buff, MAX);

read(connfd, buff, sizeof(buff));

printf("From client: %s\t To client : ", buff);

bzero(buff, MAX);

n = 0;

while ((buff[n++] = getchar()) != '\n')

;

write(connfd, buff, sizeof(buff));

if (strncmp("exit", buff, 4) == 0) {

printf("Server Exit...\n");

break;

}

}

}

int main()

{

int sockfd, connfd, len;

struct sockaddr\_in servaddr, cli;

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

if (sockfd == -1) {

printf("socket creation failed...\n");

exit(0);

}

else

printf("Socket successfully created..\n");

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

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

servaddr.sin\_port = htons(PORT);

if ((bind(sockfd, (SA\*)&servaddr, sizeof(servaddr))) != 0) {

printf("socket bind failed...\n");

exit(0);

}

else

printf("Socket successfully binded..\n");

if ((listen(sockfd, 5)) != 0) {

printf("Listen failed...\n");

exit(0);

}

else

printf("Server listening..\n");

len = sizeof(cli);

connfd = accept(sockfd, (SA\*)&cli, &len);

if (connfd < 0) {

printf("server accept failed...\n");

exit(0);

}

else

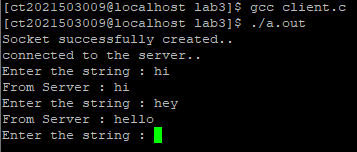
printf("server accept the client...\n");

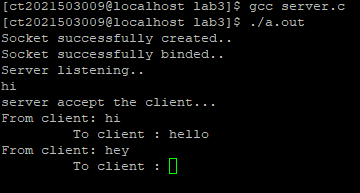
func(connfd);

close(sockfd);

}

**Ouput**





2)

**Client.java**

import java.io.\*;

import java.net.\*;

public class CLIENT {

public static void main(String[] args) {

String serverAddress = "localhost"; // Change this to the IP address or hostname of your server

int serverPort = 12345; // Port on which the server is listening

try {

Socket socket = new Socket(serverAddress, serverPort);

System.out.println("Connected to server at " + serverAddress + ":" + serverPort);

BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

String message;

while (true) {

System.out.print("Enter a message to send to the server (or 'exit' to quit): ");

message = userInput.readLine();

if (message.equalsIgnoreCase("exit")) {

break;

}

out.println(message); // Send the message to the server

System.out.println("Received from server: " + in.readLine());

}

// Close the socket when done

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Server.java**

import java.io.\*;

import java.net.\*;

public class SERVER {

public static void main(String[] args) {

int port = 12345; // Port on which the server listens

try {

ServerSocket serverSocket = new ServerSocket(port);

System.out.println("Server is listening on port " + port);

while (true) {

Socket clientSocket = serverSocket.accept(); // Wait for a client to connect

System.out.println("Client connected from " + clientSocket.getInetAddress().getHostAddress());

// Create a thread to handle the client

Thread clientThread = new ClientHandler(clientSocket);

clientThread.start();

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

class ClientHandler extends Thread {

private Socket clientSocket;

public ClientHandler(Socket socket) {

this.clientSocket = socket;

}

public void run() {

try {

// Create input and output streams for the client

BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

String message;

while ((message = in.readLine()) != null) {

System.out.println("Received from client: " + message);

out.println("Server: " + message); // Send a response to the client

}

// Close the client connection

clientSocket.close();

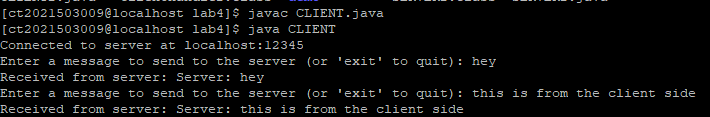
} catch (IOException e) {

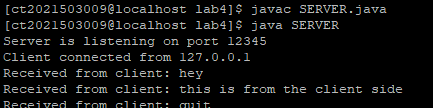
e.printStackTrace();

}

}

}





3)

**CLIENT2.java**

import java.io.\*;

import java.net.\*;

class CLIENT2 {

public static void main(String args[]) throws Exception {

// Create client socket

Socket s = new Socket("localhost", 1289);

// To send data to the server

PrintWriter pw = new PrintWriter(s.getOutputStream(), true);

// To read data coming from the server

BufferedReader br = new BufferedReader(new InputStreamReader(s.getInputStream()));

// To read data from the keyboard

BufferedReader kb = new BufferedReader(new InputStreamReader(System.in));

String str, str1;

// Repeat as long as "exit" is not typed at the client

while (!(str = kb.readLine()).equals("exit")) {

// Send to the server

pw.println(str);

// Receive from the server

str1 = br.readLine();

System.out.println("Server: " + str1);

}

// Close the connection.

pw.close();

br.close();

kb.close();

s.close();

}

}

**SERVER2.java**

import java.io.\*;

import java.net.\*;

class SERVER2 {

public static void main(String args[]) throws Exception {

// Create server Socket

ServerSocket ss = new ServerSocket(1289);

while (true) {

// Connect to a client socket

Socket s = ss.accept();

System.out.println("Connection established");

// Create input and output streams for this client

PrintStream ps = new PrintStream(s.getOutputStream());

BufferedReader br = new BufferedReader(new InputStreamReader(s.getInputStream()));

BufferedReader kb = new BufferedReader(new InputStreamReader(System.in));

String str, str1;

// Read from client and respond

while ((str = br.readLine()) != null) {

System.out.println("Client: " + str);

// Read from the server's keyboard and send to the client

str1 = kb.readLine();

ps.println(str1);

}

// Close the streams and the client socket

br.close();

ps.close();

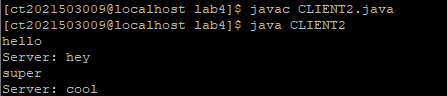
s.close();

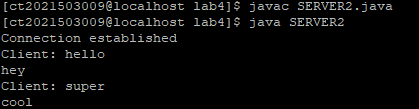
System.out.println("Connection closed.");

}

}

}





1. **FTP**

**Server.c**

#include <stdio.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#define SERV\_TCP\_PORT 8889

#define MAX 60

int main(int argc, char \*argv[]) {

int sockfd, newsockfd, clength;

struct sockaddr\_in serv\_addr, cli\_addr;

char str[MAX];

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("Socket creation failed");

exit(1);

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = INADDR\_ANY;

serv\_addr.sin\_port = htons(SERV\_TCP\_PORT);

printf("\nBinding...");

if (bind(sockfd, (struct sockaddr \*) &serv\_addr, sizeof(serv\_addr)) == -1) {

perror("Binding error");

close(sockfd);

exit(1);

}

printf("\nListening...");

listen(sockfd, 5);

clength = sizeof(cli\_addr);

newsockfd = accept(sockfd, (struct sockaddr \*) &cli\_addr, &clength);

close(sockfd);

if (newsockfd == -1) {

perror("Accepting connection failed");

exit(1);

}

read(newsockfd, str, MAX);

printf("\nClient message\n File Name: %s\n", str);

FILE \*f1 = fopen(str, "r");

if (f1 == NULL) {

perror("File opening failed");

close(newsockfd);

exit(1);

}

char buff[MAX];

while (fgets(buff, MAX, f1) != NULL) {

write(newsockfd, buff, strlen(buff)); // Use strlen to send only the actual content

printf("\n");

}

fclose(f1);

printf("\nFile Transferred\n");

close(newsockfd);

return 0;

}

**Client.c**

#include <netdb.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <stdio.h>

#include <fcntl.h> // Added for file operations

#define SERV\_TCP\_PORT 8889

#define MAX 60

int main(int argc, char \*argv[]) {

int sockfd, n;

struct sockaddr\_in serv\_addr;

char send[MAX], recvline[MAX];

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

if (sockfd == -1) {

perror("Socket creation failed");

exit(1);

}

serv\_addr.sin\_family = AF\_INET;

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

serv\_addr.sin\_port = htons(SERV\_TCP\_PORT);

if (connect(sockfd, (struct sockaddr \*) &serv\_addr, sizeof(serv\_addr)) == -1) {

perror("Connection failed");

close(sockfd);

exit(1);

}

printf("\nEnter the source file name: \n");

scanf("%s", send);

if (write(sockfd, send, MAX) == -1) {

perror("Write error");

close(sockfd);

exit(1);

}

// Open a new file on the client side with the same name

int file\_fd = open(send, O\_WRONLY | O\_CREAT, 0666);

if (file\_fd == -1) {

perror("File creation failed");

close(sockfd);

exit(1);

}

while ((n = read(sockfd, recvline, MAX)) > 0) {

// Write received data to the new file

if (write(file\_fd, recvline, n) == -1) {

perror("Write to file error");

close(sockfd);

close(file\_fd);

exit(1);

}

}

if (n < 0) {

perror("Read error");

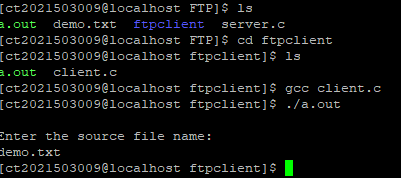
}

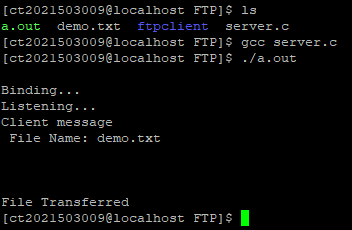
close(file\_fd); // Close the new file

close(sockfd);

return 0;

}





1. **UDP**

**Server.c**

#include <arpa/inet.h>

#include <netinet/in.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <unistd.h>

#define IP\_PROTOCOL 0

#define PORT\_NO 8887

#define NET\_BUF\_SIZE 32

#define cipherKey 'S'

#define sendrecvflag 0

#define nofile "File Not Found!"

// function to clear buffer

void clearBuf(char\* b)

{

int i;

for (i = 0; i < NET\_BUF\_SIZE; i++)

b[i] = '\0';

}

// function to encrypt

char Cipher(char ch)

{

return ch ^ cipherKey;

}

// function sending file

int sendFile(FILE\* fp, char\* buf, int s)

{

int i, len;

if (fp == NULL) {

strcpy(buf, nofile);

len = strlen(nofile);

buf[len] = EOF;

for (i = 0; i <= len; i++)

buf[i] = Cipher(buf[i]);

return 1;

}

char ch, ch2;

for (i = 0; i < s; i++) {

ch = fgetc(fp);

ch2 = Cipher(ch);

buf[i] = ch2;

if (ch == EOF)

return 1;

}

return 0;

}

// driver code

int main()

{

int sockfd, nBytes;

struct sockaddr\_in addr\_con;

int addrlen = sizeof(addr\_con);

addr\_con.sin\_family = AF\_INET;

addr\_con.sin\_port = htons(PORT\_NO);

addr\_con.sin\_addr.s\_addr = INADDR\_ANY;

char net\_buf[NET\_BUF\_SIZE];

FILE\* fp;

// socket()

sockfd = socket(AF\_INET, SOCK\_DGRAM, IP\_PROTOCOL);

if (sockfd < 0)

printf("\nfile descriptor not received!!\n");

else

printf("\nfile descriptor %d received\n", sockfd);

// bind()

if (bind(sockfd, (struct sockaddr\*)&addr\_con, sizeof(addr\_con)) == 0)

printf("\nSuccessfully binded!\n");

else

printf("\nBinding Failed!\n");

while (1) {

printf("\nWaiting for file name...\n");

// receive file name

clearBuf(net\_buf);

nBytes = recvfrom(sockfd, net\_buf,

NET\_BUF\_SIZE, sendrecvflag,

(struct sockaddr\*)&addr\_con, &addrlen);

fp = fopen(net\_buf, "r");

printf("\nFile Name Received: %s\n", net\_buf);

if (fp == NULL)

printf("\nFile open failed!\n");

else

printf("\nFile Successfully opened!\n");

while (1) {

// process

if (sendFile(fp, net\_buf, NET\_BUF\_SIZE)) {

sendto(sockfd, net\_buf, NET\_BUF\_SIZE,

sendrecvflag,

(struct sockaddr\*)&addr\_con, addrlen);

break;

}

// send

sendto(sockfd, net\_buf, NET\_BUF\_SIZE,

sendrecvflag,

(struct sockaddr\*)&addr\_con, addrlen);

clearBuf(net\_buf);

}

if (fp != NULL)

fclose(fp);

}

return 0;

}

**Client.c**

#include <arpa/inet.h>

#include <netinet/in.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <unistd.h>

#define IP\_PROTOCOL 0

#define IP\_ADDRESS "127.0.0.1" // localhost

#define PORT\_NO 8887

#define NET\_BUF\_SIZE 32

#define cipherKey 'S'

#define sendrecvflag 0

// function to clear buffer

void clearBuf(char\* b)

{

int i;

for (i = 0; i < NET\_BUF\_SIZE; i++)

b[i] = '\0';

}

// function for decryption

char Cipher(char ch)

{

return ch ^ cipherKey;

}

// function to receive file

int recvFile(char\* buf, int s)

{

int i;

char ch;

for (i = 0; i < s; i++) {

ch = buf[i];

ch = Cipher(ch);

if (ch == EOF)

return 1;

else

printf("%c", ch);

}

return 0;

}

// driver code

int main()

{

int sockfd, nBytes;

struct sockaddr\_in addr\_con;

int addrlen = sizeof(addr\_con);

addr\_con.sin\_family = AF\_INET;

addr\_con.sin\_port = htons(PORT\_NO);

addr\_con.sin\_addr.s\_addr = inet\_addr(IP\_ADDRESS);

char net\_buf[NET\_BUF\_SIZE];

FILE\* fp;

// socket()

sockfd = socket(AF\_INET, SOCK\_DGRAM,

IP\_PROTOCOL);

if (sockfd < 0)

printf("\nfile descriptor not received!!\n");

else

printf("\nfile descriptor %d received\n", sockfd);

while (1) {

printf("\nPlease enter file name to receive:\n");

scanf("%s", net\_buf);

sendto(sockfd, net\_buf, NET\_BUF\_SIZE,

sendrecvflag, (struct sockaddr\*)&addr\_con,

addrlen);

printf("\n---------Data Received---------\n");

while (1) {

// receive

clearBuf(net\_buf);

nBytes = recvfrom(sockfd, net\_buf, NET\_BUF\_SIZE,

sendrecvflag, (struct sockaddr\*)&addr\_con,

&addrlen);

// process

if (recvFile(net\_buf, NET\_BUF\_SIZE)) {

break;

}

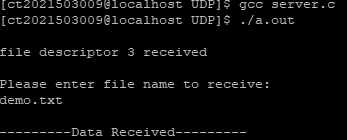
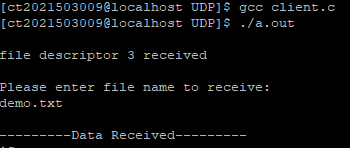
}

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

}

return 0;

}



Dijkstra:

import java.util.\*;

public class DijkstraAlgorithm {

public static void main(String[] args) {

// Create a sample graph with weights

int V = 5; // Number of vertices

List<List<Edge>> graph = new ArrayList<>();

for (int i = 0; i < V; i++) {

graph.add(new ArrayList<>());

}

// Add edges with weights

addEdge(graph, 0, 1, 2);

addEdge(graph, 0, 2, 4);

addEdge(graph, 1, 2, 1);

addEdge(graph, 1, 3, 7);

addEdge(graph, 2, 4, 3);

addEdge(graph, 3, 4, 2);

int source = 0; // Source vertex

int[] dist = dijkstra(graph, source, V);

// Print the shortest distances from the source vertex

for (int i = 0; i < V; i++) {

System.out.println("Shortest distance from vertex " + source + " to vertex " + i + " is " + dist[i]);

}

}

static class Edge {

int destination;

int weight;

Edge(int destination, int weight) {

this.destination = destination;

this.weight = weight;

}

}

static void addEdge(List<List<Edge>> graph, int source, int destination, int weight) {

graph.get(source).add(new Edge(destination, weight));

graph.get(destination).add(new Edge(source, weight)); // For undirected graph

}

static int[] dijkstra(List<List<Edge>> graph, int source, int V) {

int[] dist = new int[V];

Arrays.fill(dist, Integer.MAX\_VALUE);

dist[source] = 0;

PriorityQueue<Pair> minHeap = new PriorityQueue<>(Comparator.comparingInt(p -> p.distance));

minHeap.offer(new Pair(source, 0));

while (!minHeap.isEmpty()) {

Pair current = minHeap.poll();

int u = current.vertex;

int uDist = current.distance;

if (uDist > dist[u]) {

continue;

}

for (Edge edge : graph.get(u)) {

int v = edge.destination;

int weight = edge.weight;

if (dist[u] + weight < dist[v]) {

dist[v] = dist[u] + weight;

minHeap.offer(new Pair(v, dist[v]));

}

}

}

return dist;

}

static class Pair {

int vertex;

int distance;

Pair(int vertex, int distance) {

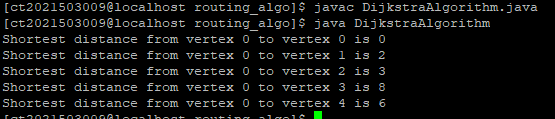
this.vertex = vertex;

this.distance = distance;

}

}

}



Distance Vector:

import java.io.\*;

public class DVR

{

static int graph[][];

static int via[][];

static int rt[][];

static int v;

static int e;

public static void main(String args[]) throws IOException

{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.println("Please enter the number of Vertices: ");

v = Integer.parseInt(br.readLine());

System.out.println("Please enter the number of Edges: ");

e = Integer.parseInt(br.readLine());

graph = new int[v][v];

via = new int[v][v];

rt = new int[v][v];

for(int i = 0; i < v; i++)

for(int j = 0; j < v; j++)

{

if(i == j)

graph[i][j] = 0;

else

graph[i][j] = 9999;

}

for(int i = 0; i < e; i++)

{

System.out.println("Please enter data for Edge " + (i + 1) + ":");

System.out.print("Source: ");

int s = Integer.parseInt(br.readLine());

s--;

System.out.print("Destination: ");

int d = Integer.parseInt(br.readLine());

d--;

System.out.print("Cost: ");

int c = Integer.parseInt(br.readLine());

graph[s][d] = c;

graph[d][s] = c;

}

dvr\_calc\_disp("The initial Routing Tables are: ");

System.out.print("Please enter the Source Node for the edge whose cost has changed: ");

int s = Integer.parseInt(br.readLine());

s--;

System.out.print("Please enter the Destination Node for the edge whose cost has changed: ");

int d = Integer.parseInt(br.readLine());

d--;

System.out.print("Please enter the new cost: ");

int c = Integer.parseInt(br.readLine());

graph[s][d] = c;

graph[d][s] = c;

dvr\_calc\_disp("The new Routing Tables are: ");

}

static void dvr\_calc\_disp(String message)

{

System.out.println();

init\_tables();

update\_tables();

System.out.println(message);

print\_tables();

System.out.println();

}

static void update\_table(int source)

{

for(int i = 0; i < v; i++)

{

if(graph[source][i] != 9999)

{

int dist = graph[source][i];

for(int j = 0; j < v; j++)

{

int inter\_dist = rt[i][j];

if(via[i][j] == source)

inter\_dist = 9999;

if(dist + inter\_dist < rt[source][j])

{

rt[source][j] = dist + inter\_dist;

via[source][j] = i;

}

}

}

}

}

static void update\_tables()

{

int k = 0;

for(int i = 0; i < 4\*v; i++)

{

update\_table(k);

k++;

if(k == v)

k = 0;

}

}

static void init\_tables()

{

for(int i = 0; i < v; i++)

{

for(int j = 0; j < v; j++)

{

if(i == j)

{

rt[i][j] = 0;

via[i][j] = i;

}

else

{

rt[i][j] = 9999;

via[i][j] = 100;

}

}

}

}

static void print\_tables()

{

for(int i = 0; i < v; i++)

{

for(int j = 0; j < v; j++)

{

System.out.print("Dist: " + rt[i][j] + " ");

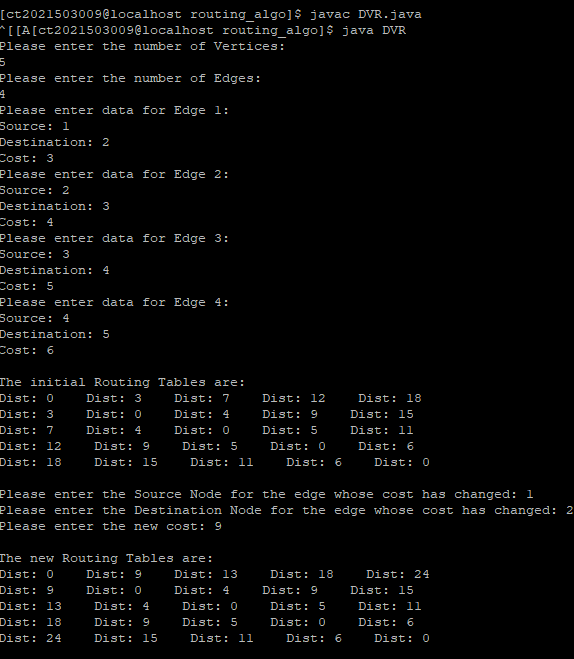
}

System.out.println();

}

}

}



Reverse DNS:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <netdb.h>

#include <arpa/inet.h>

int main() {

const char\* ip\_address = "8.8.8.8"; // Replace with the IP address you want to look up

// Reverse DNS lookup

struct hostent \*he;

struct in\_addr ipv4;

if (inet\_pton(AF\_INET, ip\_address, &ipv4) != 1) {

fprintf(stderr, "Invalid IP address.\n");

return 1;

}

he = gethostbyaddr(&ipv4, sizeof(ipv4), AF\_INET);

if (he == NULL) {

fprintf(stderr, "Reverse DNS lookup failed.\n");

return 1;

}

printf("Reverse DNS: %s - %s\n", ip\_address, he->h\_name);

return 0;

}



Forward DNS:

#include <stdio.h>

#include <stdlib.h>

#include <netdb.h>

#include <arpa/inet.h>

int main(int argc, char \*argv[]) {

struct hostent \*hen;

if (argc != 2) {

fprintf(stderr, "Enter the hostname\n");

exit(1);

}

hen = gethostbyname(argv[1]);

if (hen == NULL) {

fprintf(stderr, "Host not found\n");

exit(1);

}

//printf("\nHostname : %s\n", hen->h\_name);

if (hen->h\_addrtype == AF\_INET) {

char ip[INET\_ADDRSTRLEN];

inet\_ntop(AF\_INET, hen->h\_addr, ip, INET\_ADDRSTRLEN);

printf("IP address : %s\n\n", ip);

} else {

printf("This host does not have an IPv4 address.\n");

}

return 0;

}



http:

