

Computer Network Lab

LAB 5 Name-Aditya Agarwal

Roll-14

Section-CSE B

example1: TCP Client-Server Echo Program

1. Introduction

2. Objectives

To create a TCP server that listens for client.

To develop a TCP client that connects to the server, sends a message, and displays the echoed message from the server.

To understand and implement basic socket programming using the C language.

3. Implementation

3.1 Server-Side Implementation

Server Code Overview:

The server program is designed to:

1. **Create a TCP socket:** This socket is used to communicate with the client.
2. **Bind the socket to a specific IP address and port:** The server's socket is associated with an IP address and port, allowing it to listen for incoming connections.
3. **Listen for connections:** The server waits for clients to connect.
4. **Accept incoming connections:** When a client connects, the server accepts the connection and starts communication.
5. **Echo received messages:** The server reads a message from the client, prints it, and sends the same message back to the client.

Server Code:

c

Copy code

```
#include<stdio.h>
```

```
#include<string.h>
```

```
#include<sys/types.h>
```

```

#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<unistd.h>

#define PORTNO 10200

int main() {
    int sockfd, newsockfd, clilen, n=1;
    struct sockaddr_in seraddr, cliaddr;
    char buf[256];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("Error opening socket");
        return 1;
    }

    seraddr.sin_family = AF_INET;
    seraddr.sin_addr.s_addr = inet_addr("172.16.48.92");
    seraddr.sin_port = htons(PORTNO);

    if (bind(sockfd, (struct sockaddr *)&seraddr, sizeof(seraddr)) < 0) {
        perror("Error binding socket");
        return 1;
    }

    listen(sockfd, 5);
    printf("Server waiting for client...\n");

    while (1) {
        clilen = sizeof(cliaddr);
        newsockfd = accept(sockfd, (struct sockaddr *)&cliaddr,
        &clilen); if (newsockfd < 0) {
            perror("Error on accept");
            return 1;
        }
        n = read(newsockfd, buf, sizeof(buf));
    }
}

```

```

    if (n < 0) {
        perror("Error reading from socket");
        return 1;
    }
    printf("Message from Client: %s\n", buf);

    n = write(newsockfd, buf, sizeof(buf));
    if (n < 0) {
        perror("Error writing to socket");
        return 1;
    }
}

return 0;
}

```

3.2 Client-Side

1. **TCP socket:** The client creates a socket for communication with the server.
2. **Connect to the server**
3. **Send a message**
4. **Receive the message**

Client Code:

C

Copy code

```

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

int main() {
    int len, result, sockfd, n=1;
    struct sockaddr_in address;
    char ch[256], buf[256];

```

```

sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0) {
    perror("Error opening socket");
    return 1;
}

```

```

address.sin_family = AF_INET;
address.sin_addr.s_addr = inet_addr("172.16.48.92");
address.sin_port = htons(10200);
len = sizeof(address);

```

```

result = connect(sockfd, (struct sockaddr *)&address,
len); if(result == -1) {
    perror("CLIENT ERROR");
    exit(1);
}

```

```

printf("Enter a string: ");
fgets(ch, sizeof(ch), stdin);
ch[strlen(ch) - 1] = '\0';
write(sockfd, ch, strlen(ch));

```

```

n = read(sockfd, buf, sizeof(buf));
buf[n] = '\0';
printf("String sent back from server: %s\n", buf);

```

```

close(sockfd);

```

```

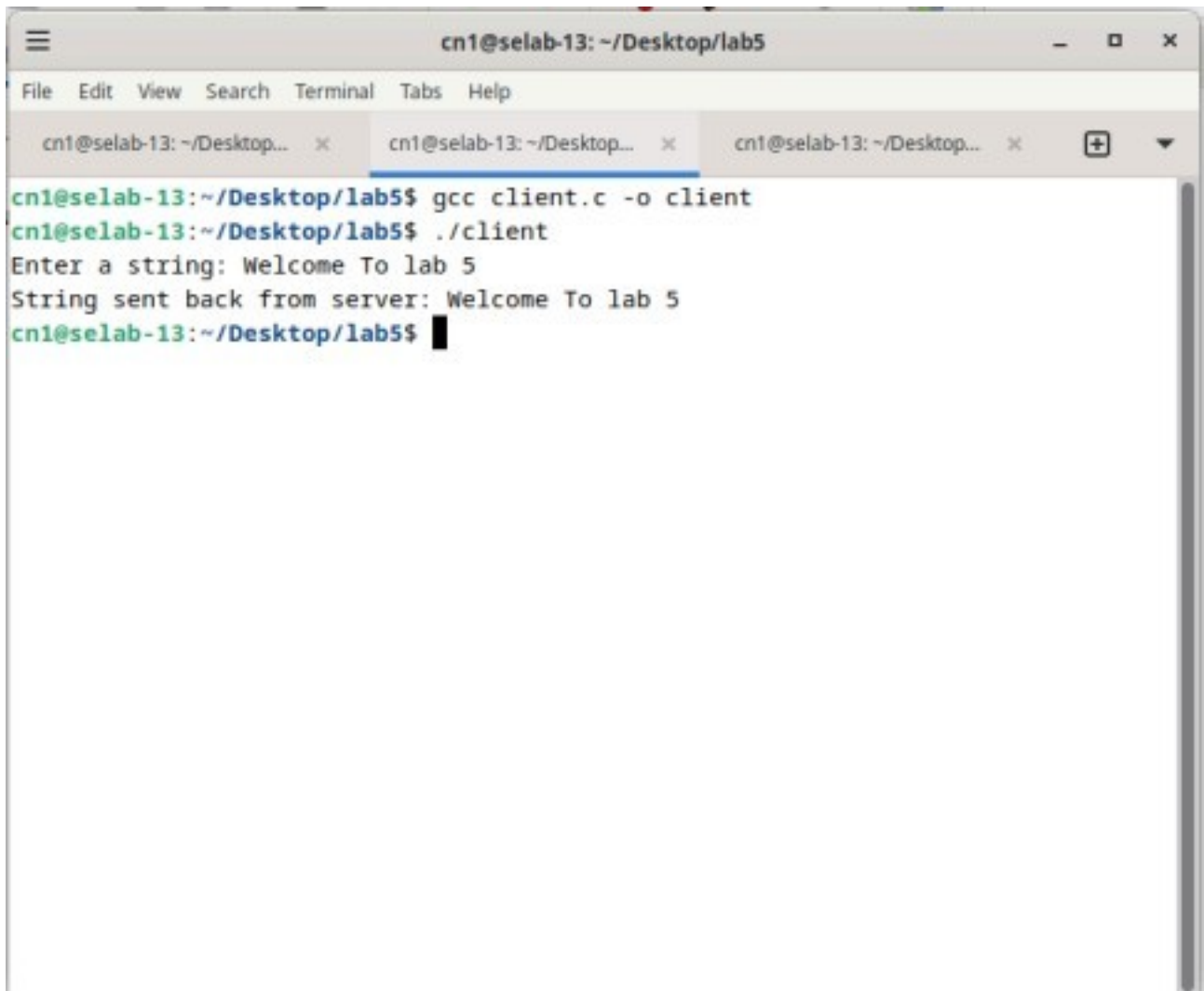
return 0;
}

```

5. Testing and Results

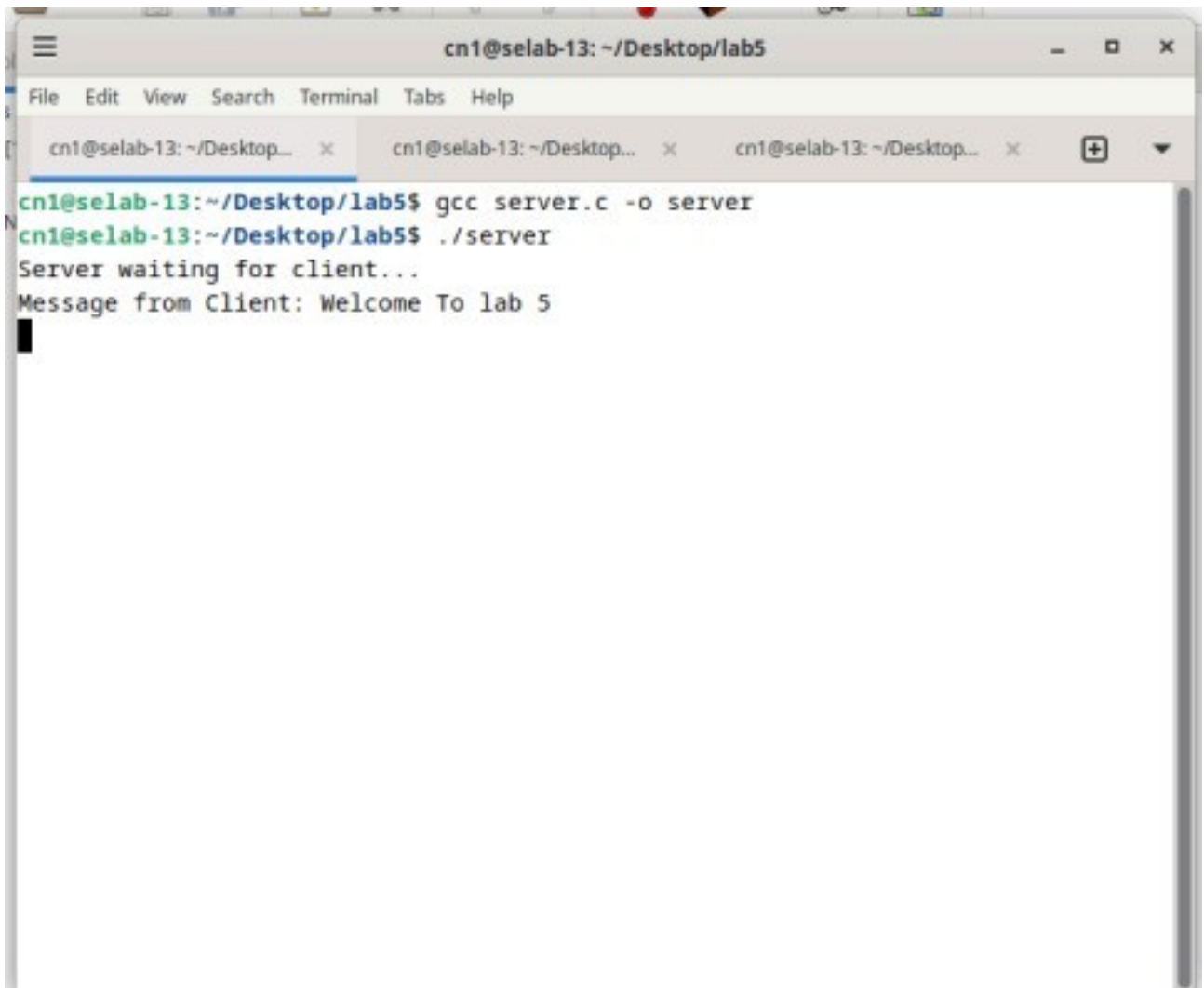
The client sends a message "Welcome to Lab 5" to the server. The server echoes this message back to the client.

Client Output:

A terminal window titled 'cn1@selab-13: ~/Desktop/lab5' with a menu bar (File, Edit, View, Search, Terminal, Tabs, Help) and three tabs. The active tab shows the following commands and output:

```
cn1@selab-13:~/Desktop/lab5$ gcc client.c -o client
cn1@selab-13:~/Desktop/lab5$ ./client
Enter a string: Welcome To lab 5
String sent back from server: Welcome To lab 5
cn1@selab-13:~/Desktop/lab5$
```

Server Output:

A screenshot of a terminal window titled 'cn1@selab-13: ~/Desktop/lab5'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', 'Tabs', and 'Help'. Below the menu bar, there are three tabs, all showing the same path: 'cn1@selab-13: ~/Desktop...'. The terminal content shows the following commands and output:

```
cn1@selab-13:~/Desktop/lab5$ gcc server.c -o server
cn1@selab-13:~/Desktop/lab5$ ./server
Server waiting for client...
Message from Client: Welcome To lab 5
```

6. Conclusion

This TCP client-server program successfully demonstrates basic socket programming in C. The server listens for client connections and echoes received messages back to the client. This simple communication model is fundamental to understanding how TCP/IP protocols operate in network programming, providing a foundation for more complex networked applications.

Q1. Write an iterative TCP client-server program where the client accepts a sentence from the user and sends it to the server. The server will check for duplicate words in the string. Server will find number of occurrences of duplicate words present and remove the duplicate words by retaining single occurrence of the word and send the resultant sentence to the client. The client displays the received data on the client screen. The process repeats until the user enter the string "Stop".

Client Code (**client.c**):

c

Copy code

// client.c

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

```
int main() {
    int len, result, sockfd, n = 1;
    struct sockaddr_in address;
    char ch[1024], buf[1024];

    // Create a socket for the client
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("Error opening socket");
        return 1;
    }
}
```

```

// Name the socket as agreed with the server
address.sin_family = AF_INET;
address.sin_addr.s_addr = inet_addr("127.0.0.1"); //
Use localhost for testing
address.sin_port = htons(10200);
len = sizeof(address);

// Connect your socket to the server's socket
result = connect(sockfd, (struct sockaddr
*)&address, len);
if (result == -1) {
    perror("CLIENT ERROR");
    exit(1);
}

// Communicate with the server
while (1) {
    printf("Enter a sentence (or type 'Stop' to end):
");
    fgets(ch, sizeof(ch), stdin);
    ch[strlen(ch) - 1] = '\0';           // Remove newline
character

// Send the sentence to the server
write(sockfd, ch, strlen(ch));

// Stop the client if "Stop" is entered
if (strcmp(ch, "Stop") == 0) {
    printf("Client stopping...\n");
    close(sockfd);
}

```



```

        exit(0);
    }

    // Read the modified sentence from the server
    n = read(sockfd, buf, sizeof(buf));
    buf[n] = '\0';
    printf("Received from server: %s\n", buf);
}

return 0;
}

```

Server Code (**server.c**):

```

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

#define PORTNO 10200

void remove_duplicates(char *str);

```

```
int main() {  
    int sockfd, newsockfd, clilen, n=1;  
    struct sockaddr_in seraddr, cliaddr;  
    char buf[1024];  
  
    // Create an unnamed socket for the server  
    sockfd = socket(AF_INET, SOCK_STREAM, 0);  
    if (sockfd < 0) {  
        perror("Error opening socket");  
        exit(1);  
    }  
  
    // Name the socket  
    seraddr.sin_family = AF_INET;  
    seraddr.sin_addr.s_addr = INADDR_ANY; // Bind to all available  
interfaces  
    seraddr.sin_port = htons(PORTNO);  
  
    // Bind the socket  
    if (bind(sockfd, (struct sockaddr *)&seraddr, sizeof(seraddr)) < 0) {  
        perror("Error binding socket");  
        exit(1);  
    }  
  
    // Create a connection queue and wait for clients  
    listen(sockfd, 5);  
    printf("Server waiting for client...\n");  
  
    while (1) {
```

```
clilen = sizeof(cliaddr);
newsockfd = accept(sockfd, (struct sockaddr *)&cliaddr, &clilen);
if (newsockfd < 0) {
    perror("Error on accept");
    exit(1);
}
```

```
// Read from client
```

```
while ((n = read(newsockfd, buf, sizeof(buf)-1)) > 0) {
    buf[n] = '\0'; // Null-terminate the string
    if (strcmp(buf, "Stop") == 0) {
        printf("Server stopping...\n");
        close(newsockfd);
        close(sockfd);
        exit(0);
    }
}
```

```
printf("Received from client: %s\n", buf);
```

```
// Remove duplicate words
```

```
remove_duplicates(buf);
```

```
// Send the modified sentence back to the client
```

```
write(newsockfd, buf, strlen(buf));
}
```

```
close(newsockfd);
```

```
}
```

```

        return 0;
    }

// Function to remove duplicate words from a sentence
void remove_duplicates(char *str) {
    char *token;
    char result[1024] = "";
    char *words[256];
    int word_count = 0;

    // Tokenize the input string and store the words
    token = strtok(str, " ");
    while (token != NULL) {
        int duplicate = 0;
        // Check if the word is already in the result
        for (int i = 0; i < word_count; i++) {
            if (strcmp(words[i], token) == 0) {
                duplicate = 1;
                break;
            }
        }
        // If not a duplicate, add it to the result
        if (!duplicate) {
            words[word_count] = token;
            word_count++;
            strcat(result, token);
            strcat(result, " ");
        }
        token = strtok(NULL, " ");
    }
}

```

```
}
```

```
// Copy the result back to the original string
```

```
strcpy(str, result);
```

```
str[strlen(str) - 1] = '\0'; // Remove the trailing space
```

```
}
```



```
cn1@selab-13: ~/Desktop/lab5/q1
File Edit View Search Terminal Tabs Help
cn1@selab-13: ~/Desktop/lab5/q1 x cn1@selab-13: ~/Desktop/lab5/q1 x cn1@selab-13: ~/Desktop/lab5/q1 x
cn1@selab-13:~/Desktop/lab5/q1$ gcc -o client client.c
cn1@selab-13:~/Desktop/lab5/q1$ ./client
CLIENT ERROR: No route to host
cn1@selab-13:~/Desktop/lab5/q1$ gcc -o client client.c
cn1@selab-13:~/Desktop/lab5/q1$ ./client
Enter a sentence (or type 'Stop' to end): he quick brown fox jumps over the lazy dog.
Received from server: he quick brown fox jumps over the lazy dog.
Enter a sentence (or type 'Stop' to end): hi hi i am am ayush
Received from server: hi i am ayush
Enter a sentence (or type 'Stop' to end): Stop
Client stopping...
cn1@selab-13:~/Desktop/lab5/q1$
```



```
cn1@selab-13: ~/Desktop/lab5/q1
File Edit View Search Terminal Tabs Help
cn1@selab-13: ~/Desktop/lab5/q1 x cn1@selab-13: ~/Desktop/lab5/q1 x cn1@selab-13: ~/Desktop/lab5/q1 x
cn1@selab-13:~/Desktop/lab5/q1$ gcc -o server server.c
cn1@selab-13:~/Desktop/lab5/q1$ ./server
Error binding socket: Cannot assign requested address
cn1@selab-13:~/Desktop/lab5/q1$ gcc -o server server.c
cn1@selab-13:~/Desktop/lab5/q1$ ./server
Server waiting for client...
Received from client: he quick brown fox jumps over the lazy dog.
Received from client: hi hi i am am ayush

```

Conclusion:

- Client: The client accepts sentences from the user and sends them to

the server. It displays the processed sentence received from the server. The program terminates when the user types "Stop."

- Server: The server listens for connections from the client, processes the sentence by removing duplicate words, and sends the resultant sentence back to the client. Both client and server terminate when "Stop" is received.

Q2 To implement an iterative UDP-based client-server application where the client sends rows of a matrix, and the server combines them into a full matrix.

udp_server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
```

```
#define PORT 10200
#define MAXLINE 1024
```

```
int main() {
    int sockfd;
    char buffer[MAXLINE];
    struct sockaddr_in servaddr, cliaddr;

    // Creating socket file descriptor
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));

    // Filling server information
    servaddr.sin_family = AF_INET; // IPv4
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);
```

```

// Bind the socket with the server address
if (bind(sockfd, (const struct sockaddr *)&servaddr,
sizeof(servaddr)) < 0) {
    perror("Bind failed");
    exit(EXIT_FAILURE);
}

printf("Server is running and waiting for data...\n");

int n, len, row_count = 0;
int matrix[100][100]; // Assume maximum size of matrix is 100x100
len = sizeof(cliaddr);

while (1) {
    n = recvfrom(sockfd, buffer, MAXLINE, MSG_WAITALL, (struct
sockaddr *)&cliaddr, &len);
    buffer[n] = '\0';

    // Stop the server if "Stop" is received
    if (strcmp(buffer, "Stop") == 0) {
        printf("Server stopping...\n");
        break;
    }

    // Convert received string to integers and store in the matrix
    int i = 0, col_count = 0;
    char *token = strtok(buffer, " ");
    while (token != NULL) {
        matrix[row_count][col_count++] = atoi(token);
        token = strtok(NULL, " ");
    }

    printf("Received row %d: ", row_count + 1);
    for (i = 0; i < col_count; i++) {
        printf("%d ", matrix[row_count][i]);
    }
    printf("\n");
    row_count++;
}

```

```

        printf("Final matrix received:\n");
        for (int i = 0; i < row_count; i++) {
            for (int j = 0; j < 100 && matrix[i][j] != 0; j++) { // Only print till
non-zero elements
                printf("%d ", matrix[i][j]);
            }
            printf("\n");
        }

        close(sockfd);
        return 0;
}

```

udp_client.c

```

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <arpa/inet.h>


#define PORT 10200

#define MAXLINE 1024


int main() {

    int sockfd;

    char buffer[MAXLINE];

```



```

struct sockaddr_in servaddr;

// Creating socket file descriptor

if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {

    perror("Socket creation failed");

    exit(EXIT_FAILURE);

}

memset(servaddr, 0, sizeof(servaddr));

// Filling server information

servaddr.sin_family = AF_INET;

servaddr.sin_port = htons(PORT);

servaddr.sin_addr.s_addr = inet_addr("127.0.0.1"); // Use localhost for
testing

int n, len;

while (1) {

    printf("Enter a row of the matrix (or type 'Stop' to end): ");

    fgets(buffer, sizeof(buffer), stdin);

    buffer[strlen(buffer) - 1] = '\0'; // Remove newline character

    // Send row to the server

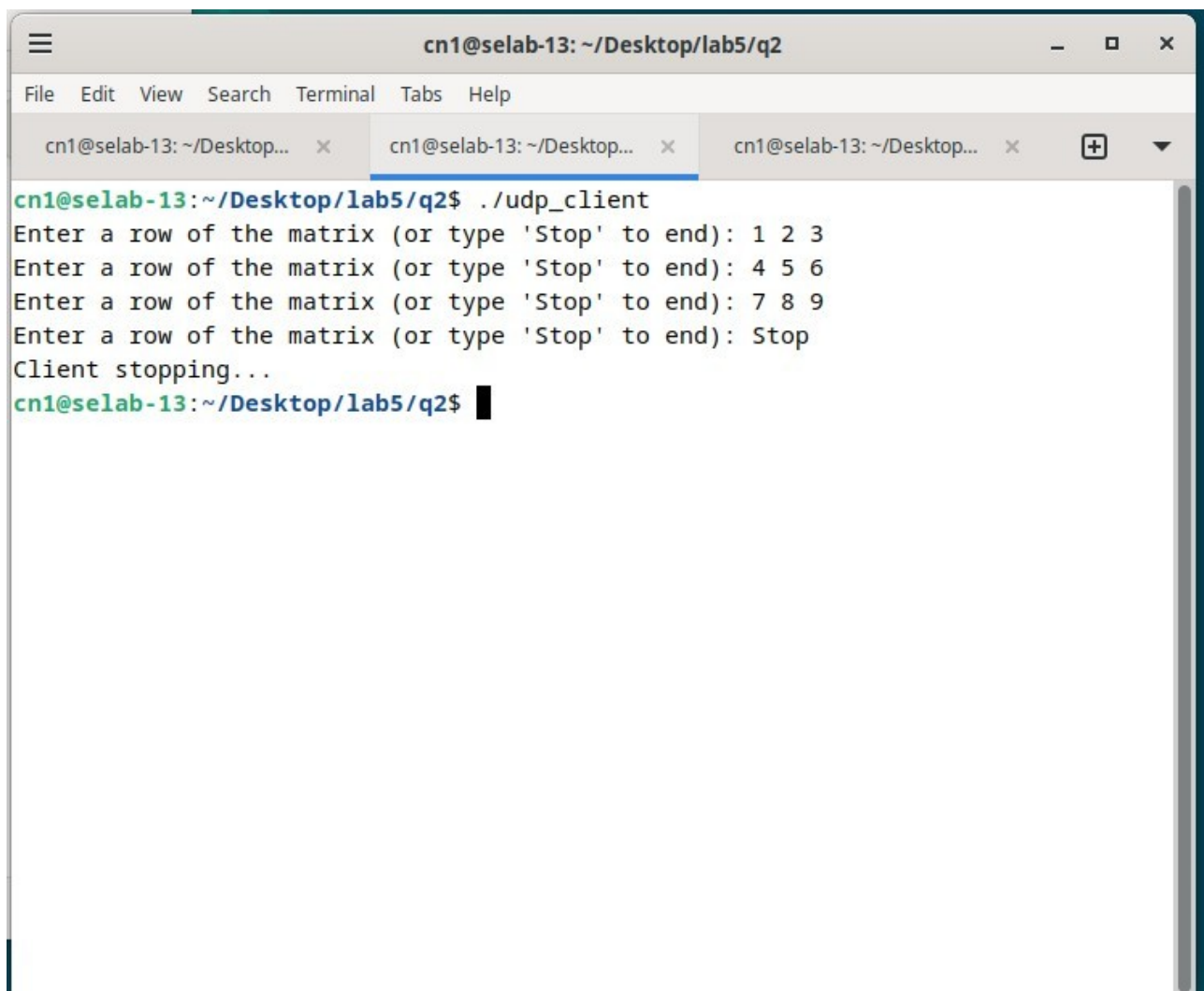
    sendto(sockfd, buffer, strlen(buffer), MSG_CONFIRM, (const struct
sockaddr *)&servaddr, sizeof(servaddr));

    // Stop the client if "Stop" is entered

    if (strcmp(buffer, "Stop") == 0) {

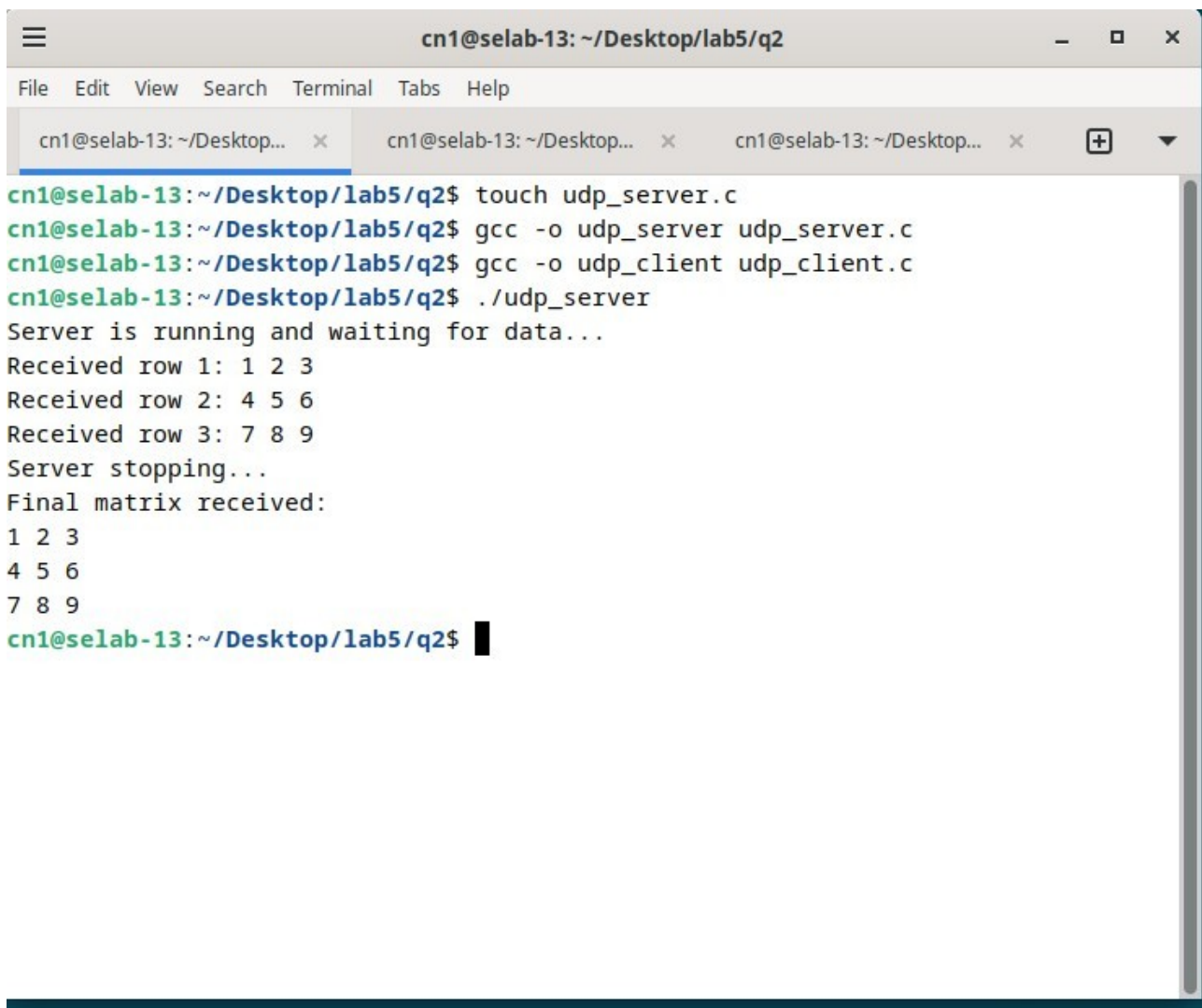
```

```
printf("Client stopping...\n");  
  
break;  
  
}  
  
}  
  
close(sockfd);  
  
return 0;  
  
}
```



The screenshot shows a terminal window titled "cn1@selab-13: ~/Desktop/lab5/q2". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", "Tabs", and "Help". There are three tabs open, all with the same title. The terminal content shows the execution of the program `./udp_client`. The program prompts the user to enter rows of a matrix. The user enters "1 2 3", "4 5 6", and "7 8 9". When prompted for the fourth row, the user enters "Stop". The program then prints "Client stopping..." and returns to the shell prompt.

```
cn1@selab-13:~/Desktop/lab5/q2$ ./udp_client  
Enter a row of the matrix (or type 'Stop' to end): 1 2 3  
Enter a row of the matrix (or type 'Stop' to end): 4 5 6  
Enter a row of the matrix (or type 'Stop' to end): 7 8 9  
Enter a row of the matrix (or type 'Stop' to end): Stop  
Client stopping...  
cn1@selab-13:~/Desktop/lab5/q2$
```



```
cn1@selab-13: ~/Desktop/lab5/q2
File Edit View Search Terminal Tabs Help
cn1@selab-13: ~/Desktop... x cn1@selab-13: ~/Desktop... x cn1@selab-13: ~/Desktop... x
cn1@selab-13:~/Desktop/lab5/q2$ touch udp_server.c
cn1@selab-13:~/Desktop/lab5/q2$ gcc -o udp_server udp_server.c
cn1@selab-13:~/Desktop/lab5/q2$ gcc -o udp_client udp_client.c
cn1@selab-13:~/Desktop/lab5/q2$ ./udp_server
Server is running and waiting for data...
Received row 1: 1 2 3
Received row 2: 4 5 6
Received row 3: 7 8 9
Server stopping...
Final matrix received:
1 2 3
4 5 6
7 8 9
cn1@selab-13:~/Desktop/lab5/q2$
```

Conclusion

The project successfully implements matrix communication using UDP, where the client sends rows iteratively, and the server reconstructs the matrix. This demonstrates efficient data transfer using connectionless UDP protocol for real-time applications.

Q3.To illustrate encryption and decryption of messages using TCP. The client accepts messages to be encrypted through standard input device. The client will encrypt the string by adding 4 (random value) to ASCII value of each alphabet. The encrypted message is sent to the server. The server then decrypts the message and displays both encrypted and decrypted forms of the string. Program terminates after one session.

Objective

This project illustrates message encryption and decryption using TCP. The client encrypts a message by adding a fixed value to the ASCII value of each character and sends it to the server. The server decrypts the message and displays both the encrypted and decrypted versions.

Overview

The client-side program encrypts user input by adding 4 to the ASCII value of each character and sends the encrypted message to the server via TCP. The server receives the encrypted message, decrypts it by subtracting 4 from the ASCII value, and displays both the encrypted and decrypted messages.

Server Code (TCP Server)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define BUFFER_SIZE 1024

void decrypt(char *message) {
    for (int i = 0; i < strlen(message); i++) {
        if ((message[i] >= 'a' && message[i] <= 'z') || (message[i] >= 'A' && message[i] <= 'Z')) {
            message[i] = message[i] - 4;
        }
    }
}
```

```

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[BUFFER_SIZE] = {0};

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        printf("Socket failed\n");
        exit(1);
    }

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        printf("Bind failed\n");
        exit(1);
    }

    if (listen(server_fd, 3) < 0) {
        printf("Listen failed\n");
        exit(1);
    }

    if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
(socklen_t*)&addrlen)) < 0) {
        printf("Accept failed\n");
        exit(1);
    }

    read(new_socket, buffer, BUFFER_SIZE);
    printf("Encrypted message received: %s\n", buffer);
}

```

```
    decrypt(buffer);
    send(new_socket, buffer, strlen(buffer), 0);
    printf("Decrypted message sent back to client: %s\n", buffer);

    close(new_socket);
    close(server_fd);
    return 0;
}
```

Client Code (TCP Client)

```
#include <stdio.h>#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER_SIZE 1024

void encrypt(char *message) {

    for (int i = 0; i < strlen(message); i++) {
```

```
        if ((message[i] >= 'a' && message[i] <= 'z') || (message[i] >= 'A' &&
message[i] <= 'Z')) {

            message[i] = message[i] + 4;

        }

    }

}
```

```
int main() {

    int sock = 0;

    struct sockaddr_in serv_addr;

    char message[BUFFER_SIZE] = {0};

    char buffer[BUFFER_SIZE] = {0};

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {

        printf("Socket creation error\n");

        return -1;

    }

    serv_addr.sin_family = AF_INET;

    serv_addr.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
```

```
    printf("Invalid address\n");

    return -1;

}

if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
{

    printf("Connection failed\n");

    return -1;

}

printf("Enter message: ");

fgets(message, BUFFER_SIZE, stdin);

message[strcspn(message, "\n")] = 0;

encrypt(message);

send(sock, message, strlen(message), 0);

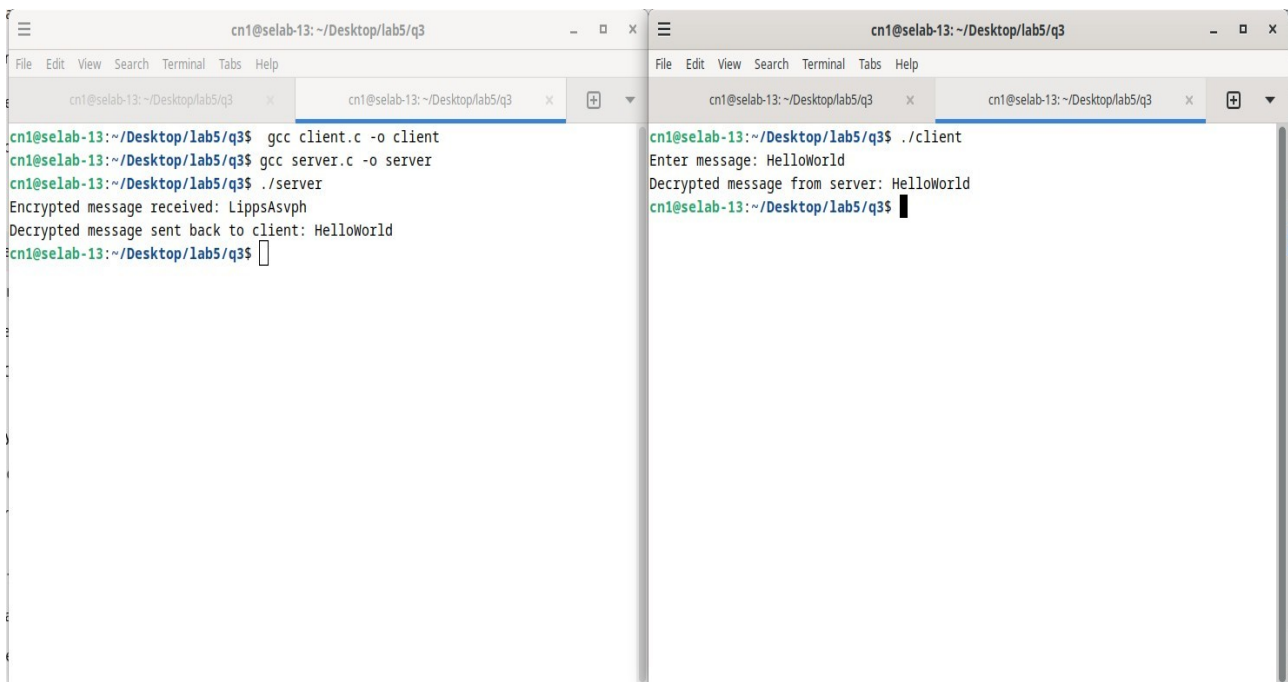
read(sock, buffer, BUFFER_SIZE);

printf("Decrypted message from server: %s\n", buffer);

close(sock);

return 0;

}
```

```
cn1@selab-13: ~/Desktop/lab5/q3
File Edit View Search Terminal Tabs Help
cn1@selab-13: ~/Desktop/lab5/q3 x
cn1@selab-13: ~/Desktop/lab5/q3 x
+
cn1@selab-13:~/Desktop/lab5/q3$ gcc client.c -o client
cn1@selab-13:~/Desktop/lab5/q3$ gcc server.c -o server
cn1@selab-13:~/Desktop/lab5/q3$ ./server
Encrypted message received: LippsAsvph
Decrypted message sent back to client: HelloWorld
cn1@selab-13:~/Desktop/lab5/q3$

cn1@selab-13: ~/Desktop/lab5/q3
File Edit View Search Terminal Tabs Help
cn1@selab-13: ~/Desktop/lab5/q3 x
cn1@selab-13: ~/Desktop/lab5/q3 x
+
cn1@selab-13:~/Desktop/lab5/q3$ ./client
Enter message: HelloWorld
Decrypted message from server: HelloWorld
cn1@selab-13:~/Desktop/lab5/q3$
```

Conclusion

This project successfully demonstrates TCP-based communication where a client encrypts a message and sends it to a server, which decrypts and displays both versions. This illustrates basic encryption/decryption and data transmission over a network.

Q4. Write a client program to send a manually crafted HTTP request packet to a Web Server and display all fields received in HTTP Response at client Side.

Objective

This project demonstrates how to manually craft an HTTP request packet in a client program, send it to a web server, and display all fields received in the HTTP response.

Overview

The client-side program manually creates a basic HTTP request (GET) and sends it to a web server via TCP. The response from the server is received, and all fields in the HTTP response (such as headers, status code, and body) are displayed on the client side.

Client Code (HTTP Client)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 80

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char request[] = "GET / HTTP/1.1\r\nHost:
www.example.com\r\nConnection: close\r\n\r\n";
    char buffer[3000] = {0};

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        exit(EXIT_FAILURE);
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    inet_pton(AF_INET, "93.184.216.34", &serv_addr.sin_addr); // IP address
    for example.com

    connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr));

    send(sock, request, strlen(request), 0);

    read(sock, buffer, 3000);

    printf("HTTP Response:\n%s\n", buffer);

    close(sock);
}
```

```
    return 0;
}
```

Output

```
cn1@selab-13:~/Desktop/lab5/q4$ gcc http_client.c -o http_client
```

```
cn1@selab-13:~/Desktop/lab5/q4$ ./http_client
```

```
HTTP/1.1 200 OK
```

```
Accept-Ranges: bytes
```

```
Age: 496800
```

```
Cache-Control: max-age=604800
```

```
Content-Type: text/html; charset=UTF-8
```

```
Date: Mon, 16 Sep 2024 03:59:23 GMT
```

```
Etag: "3147526947"
```

```
Expires: Mon, 23 Sep 2024 03:59:23 GMT
```

```
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
```

```
Server: ECAcc (Iac/55CF)
```

```
Vary: Accept-Encoding
```

```
X-Cache: HIT
```

```
Content-Length: 1256
```

```
Connection: close
```

```
<!doctype html>
```

```
<html>
```

```
<head>
```

```
    <title>Example Domain</title>
```

```
    <meta charset="utf-8" />
```

```
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1" />
```

```
    <style type="text/css">
```

```
    body {
```

```
    background-color: #f0f0f2;
```

```
    margin: 0;
```

```
    padding: 0;
```

```
    font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;
```

```
    }
```

```
    div {
```

```
    width: 600px;
```

```

margin: 5em auto;
padding: 2em;
background-color: #fdfdff;
border-radius: 0.5em;
box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
}
a:link, a:visited {
color: #38488f;
text-decoration: none;
}
@media (max-width: 700px) {
div {
margin: 0 auto;
width: auto;
}
}
</style>
</head>

<body>
<div>
  <h1>Example Domain</h1>
  <p>This domain is for use in illustrative examples in documents. You
may use this
  domain in literature without prior coordination or asking for
permission.</p>
  <p><a href="https://www.iana.org/domains/example">More
information...</a></p>
</div>
</body>
</html>

```

Conclusion

This project successfully demonstrates how to manually craft an HTTP request and send it to a web server using TCP. The program displays all fields received in the HTTP response, such as headers and body content, giving insight into basic web communication via raw socket programming.