

THE UNIVERSITY OF BUEA

P.O box 63,

Buea, South-West Region

Cameroon

Tel: (237) 674556329



REPUBLIC OF CAMEROON

PEACE-WORK-FATHERLAND

FACULTY OF ENGINEERING AND TECHNOLOGY

COMPUTER ENGINEERING

CEF 488: System and Network Programming

A Report on Socket Programming Lab

Presented by:

Tambong Kersten Melengfe (Fe21A440)

Course Supervisor:

Valery Nkemeni, PhD

Academic year: 2024

Introduction

Socket programming is essential for enabling communication between processes over a network. This lab focuses on implementing TCP and UDP servers and clients using the C programming language. Through practical exercises, we create programs that allow clients to send messages to servers, which then process and respond to these messages. We also explore inter-client communication and modify programs to handle different data types, such as integers. These exercises provide hands-on experience with socket programming, reinforcing key concepts in network programming.

Objectives

- To implement a TCP server and client in C.
- To implement a UDP server and client in C.
- To create programs to demonstrate inter-client communication using a server.
- To modify the programs to handle different data types (integer).

Problem Solving

Task1: TCP and Server

Using socket programming, implement a TCP server (tcpserver.c) and a TCP client (tcpclient.c) using C language.

Code

TcpServer.c

Defining the Port and Buffer Size of the Server

```
7
8 #define PORT 8080
9 #define BUFFER_SIZE 1024
10
```

Declaring Variable and Creating Socket file descriptor

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

// Creating socket file descriptor
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
    perror("socket failed");
    exit(EXIT_FAILURE);
}
```

Setting up the address structure and binding the socket to the address and port

```
// Setting up the address structure
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

// Binding the socket to the address and port
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
    perror("bind failed");
    close(server_fd);
    exit(EXIT_FAILURE);
}
```

Listening and Accepting Incoming Connections in the server

```
// Listening for incoming connections
if (listen(server_fd, 3) < 0) {
    perror("listen");
    close(server_fd);
    exit(EXIT_FAILURE);
}

// Accepting an incoming connection
if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen)) < 0) {
    perror("accept");
    close(server_fd);
    exit(EXIT_FAILURE);
}
```

Reading and Sending message to the client and ensuring the sockets are closed

```
48
49 // Reading data from the client
50 read(new_socket, buffer, BUFFER_SIZE);
51 printf("Message from client: %s\n", buffer);
52
53 // Sending a response to the client
54 send(new_socket, "Hello from server", strlen("Hello from server"), 0);
55
56 // Closing the connection
57 close(new_socket);
58 close(server_fd);
59 return 0;
60
61
```

TcpClient.c

converting Ip address to binary form

```
tcpclient.c > main()
0 int main() {
1     struct sockaddr_in serv_addr;
2     char buffer[BUFFER_SIZE] = {0};
3
4     // Creating socket file descriptor
5     if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
6         printf("\n Socket creation error \n");
7         return -1;
8     }
9
10    // Setting up the server address structure
11    serv_addr.sin_family = AF_INET;
12    serv_addr.sin_port = htons(PORT);
13
14    // Converting IP address to binary form and setting it
15    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
16        printf("\nInvalid address/ Address not supported \n");
17        return -1;
18    }
19
20}
```

Connecting to the Server, and sending a message to the server and reading response

```
// Connecting to the server
if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
    printf("\nConnection Failed \n");
    return -1;
}

// Sending a message to the server
send(sock, "Hello from client", strlen("Hello from client"), 0);

// Reading the response from the server
read(sock, buffer, BUFFER_SIZE);
printf("Message from server: %s\n", buffer);

// Closing the socket
close(sock);
return 0;
```

Output

TcpServer.c

```
slade@slade-VMware-Virtual-Platform:~/Lab5$ ls
slade@slade-VMware-Virtual-Platform:~/Lab5$ ls
client1.c client2.c server.c tcpclient tcpclient.c tcpserver tcpserver.c udpclient udpclient.c udpserver udpserver.c
slade@slade-VMware-Virtual-Platform:~/Lab5$ ./tcpserver
Message from client: Hello from client
slade@slade-VMware-Virtual-Platform:~/Lab5$
```

TcpClient.c

```
slade@slade-VMware-Virtual-Platform:~/Lab5$ ls
client1.c client2.c server.c tcpclient tcpclient.c tcpserver tcpserver.c udpclient udpclient.c udpserver udpserver.c
slade@slade-VMware-Virtual-Platform:~/Lab5$ ./tcpclient
Message from server: Hello from server
slade@slade-VMware-Virtual-Platform:~/Lab5$
```

Results

- The client successfully sends a message to the server.
- The server receives the message and responds back to the client.
- The client prints the server's response.

Task2: UDP Server and Client

Using socket programming, implement a UDP server (udpserver.c) and a UDP client (udpclient.c) using C language

Code

UdpServer.c

Binding the socket to the address, receiving and sending response to client and closing socket

```
30 // Binding the socket to the address and port
31 if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
32     perror("bind failed");
33     close(sockfd);
34     exit(EXIT_FAILURE);
35 }
36
37 int len, n;
38 len = sizeof(cliaddr);
39
40 // Receiving data from the client
41 n = recvfrom(sockfd, (char *)buffer, BUFFER_SIZE, MSG_WAITALL, (struct sockaddr *)&cliaddr, &len);
42 buffer[n] = '\0';
43 printf("Client: %s\n", buffer);
44
45 // Sending a response to the client
46 sendto(sockfd, "Hello from server", strlen("Hello from server"), MSG_CONFIRM, (const struct sockaddr *)&cliaddr, len);
47
48 // Closing the socket
49 close(sockfd);
50 return 0;
51 }
```

UdpClient.c

Setting up the udp server address and structure, sending message to server and receiving response

```
24 // Setting up the server address structure
25 servaddr.sin_family = AF_INET;
26 servaddr.sin_port = htons(PORT);
27 servaddr.sin_addr.s_addr = INADDR_ANY;
28
29 int n, len;
30
31 // Sending a message to the server
32 sendto(sockfd, "Hello from client", strlen("Hello from client"), MSG_CONFIRM, (const struct sockaddr *)&servaddr, sizeof(servaddr))
33
34 // Receiving the response from the server
35 n = recvfrom(sockfd, (char *)buffer, BUFFER_SIZE, MSG_WAITALL, (struct sockaddr *)&servaddr, &len);
36 buffer[n] = '\0';
37 printf("Server: %s\n", buffer);
38
39 // Closing the socket
40 close(sockfd);
41 return 0;
42 }
43
```

Output

UdpServer.c

```
slade@slade-Virtual-Platform:~/Lab$ ./udpserver
Client: Hello from client
slade@slade-Virtual-Platform:~/Lab$
```

UDPCClient.c

```
slade@slade-Virtual-Platform:~/Lab$ ls
client1.c client2.c server.c tcpclient tcpclient.c tcpserver tcpserver.c udpclient udpclient.c udpserver udpserver.c
slade@slade-Virtual-Platform:~/Lab$ ./udpclient
Server: Hello from server
slade@slade-Virtual-Platform:~/Lab$
```

Results

- The client sends a message to the server using UDP.
- The server receives the message and responds back to the client.
- The client prints the server's response.

Task 3: Inter-Client Communication

Create three programs, two of which are clients to a single server. Client1 will send a character to the server process. The server will decrement the letter to the next letter in the alphabet and send the result to client2. Client2 prints the letter it receives and then all the processes terminate.

Code

Server.c

Establishing Server connection between clients and sending data between the client1 and 2

```

35 // Listening for incoming connections
36 if (listen(server_fd, 3) < 0) {
37     perror("listen");
38     close(server_fd);
39     exit(EXIT_FAILURE);
40 }
41
42 // Accepting connection from Client1
43 if ((client1_fd = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen)) < 0) {
44     perror("accept");
45     close(server_fd);
46     exit(EXIT_FAILURE);
47 }
48
49 // Reading data from Client1
50 read(client1_fd, buffer, BUFFER_SIZE);
51 printf("Received from Client1: %s\n", buffer);
52 buffer[0]--; // Decrement the character
53 printf("Sending to Client2: %s\n", buffer);
54
55 // Accepting connection from Client2
56 if ((client2_fd = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen)) < 0) {
57     perror("accept");
58     close(server_fd);
59     exit(EXIT_FAILURE);
60 }
61
62 // Sending the modified character to Client2
63 send(client2_fd, buffer, strlen(buffer), 0);
64
65 // Closing the connections
66 close(client1_fd);
67 close(client2_fd);

```

Client1.c

```

10 int main() {
11     int sock = 0;
12     struct sockaddr_in serv_addr;
13     char buffer[BUFFER_SIZE] = "A"; // Sending character A
14
15     // Creating socket file descriptor
16     if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
17         printf("\n Socket creation error \n");
18         return -1;
19     }
20
21     // Setting up the server address structure
22     serv_addr.sin_family = AF_INET;
23     serv_addr.sin_port = htons(PORT);
24
25     // Converting IP address to binary form and setting it
26     if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
27         printf("\nInvalid address/ Address not supported \n");
28         return -1;
29     }
30
31     // Connecting to the server
32     if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
33         printf("\nConnection Failed \n");
34         return -1;
35     }
36
37     // Sending a message to the server
38     send(sock, buffer, strlen(buffer), 0);
39
40     // Closing the socket
41     close(sock);
42     return 0;
43 }

```

Client2.c

```
C tcpserver.c C tcpclient.c C udpserver.c C server.c C client1.c C client2.c x C udpclient.c
C client2.c main()
7 #define PORT 8080
8 #define BUFFER_SIZE 1024
9
10 int main() {
11     int sock = 0;
12     struct sockaddr_in serv_addr;
13     char buffer[BUFFER_SIZE] = {0};
14
15     // Creating socket file descriptor
16     if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
17         printf("\n Socket creation error \n");
18         return -1;
19     }
20
21     // Setting up the server address structure
22     serv_addr.sin_family = AF_INET;
23     serv_addr.sin_port = htons(PORT);
24
25     // Converting IP address to binary form and setting it
26     if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
27         printf("\nInvalid address/ Address not supported \n");
28         return -1;
29     }
30
31     // Connecting to the server
32     if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
33         printf("\nConnection Failed \n");
34         return -1;
35     }
36
37     // Reading the message from the server
38     read(sock, buffer, BUFFER_SIZE);
39     printf("Received from server: %s\n", buffer);
40
41     // Closing the socket
42     close(sock);
43     return 0;
44 }
```

Output Server

```
slade@slade-VMware-Virtual-Platform: ~/Lab5
slade@slade-VMware-Virtual-Platform: ~/Lab5
slade@slade-VMware-Virtual-Platform: ~/Lab5
slade@slade-VMware-Virtual-Platform: ~/Lab5$ ls
Desktop Documents Downloads Lab5 Music Pictures Public snap Templates Videos
slade@slade-VMware-Virtual-Platform: ~/Lab5$ cd Lab5/
slade@slade-VMware-Virtual-Platform: ~/Lab5$ ls
client1.c client2.c server.c tcpclient tcpclient.c tcpserver tcpserver.c udpclient udpclient.c udpserver udpserver.c
slade@slade-VMware-Virtual-Platform: ~/Lab5$ ./tcpserver
Message from client: Hello from client
slade@slade-VMware-Virtual-Platform: ~/Lab5$ ./udpserver
Client: Hello from client
slade@slade-VMware-Virtual-Platform: ~/Lab5$ ls
client1 client1.c client2 client2.c server server.c tcpclient tcpclient.c tcpserver tcpserver.c udpclient udpclient.c udpserver udpserver.c
slade@slade-VMware-Virtual-Platform: ~/Lab5$ ./server
Received from Client1: A
Sending to Client2: @
slade@slade-VMware-Virtual-Platform: ~/Lab5$
```

Client1

```
Jul 5 02:15
slade@slade-VMware-Virtual-Platform: ~/Lab5
slade@slade-VMware-Virtual-Platform: ~/Lab5
slade@slade-VMware-Virtual-Platform: ~/Lab5
slade@slade-VMware-Virtual-Platform: ~/Lab5$ ls
client1.c client2.c server.c tcpclient tcpclient.c tcpserver tcpserver.c udpclient udpclient.c udpserver udpserver.c
slade@slade-VMware-Virtual-Platform: ~/Lab5$ ./udpclient
Server: Hello from server
slade@slade-VMware-Virtual-Platform: ~/Lab5$ ls
client1 client1.c client2 client2.c server server.c tcpclient tcpclient.c tcpserver tcpserver.c udpclient udpclient.c udpserver udpserver.c
slade@slade-VMware-Virtual-Platform: ~/Lab5$ ./client1
slade@slade-VMware-Virtual-Platform: ~/Lab5$
```

Client2

```
Jul 5 02:16
slade@slade-VMware-Virtual-Platform: ~/Lab5
slade@slade-VMware-Virtual-Platform: ~/Lab5
slade@slade-VMware-Virtual-Platform: ~/Lab5
slade@slade-VMware-Virtual-Platform: ~/Lab5$ ls
client1 client1.c client2 client2.c server server.c tcpclient tcpclient.c tcpserver tcpserver.c udpclient udpclient.c udpserver udpserver.c
slade@slade-VMware-Virtual-Platform: ~/Lab5$ ./client2
Received from server: @
slade@slade-VMware-Virtual-Platform: ~/Lab5$
```


Results

Client1 sends a character to the server.

The server decrements the character and sends it to Client2.

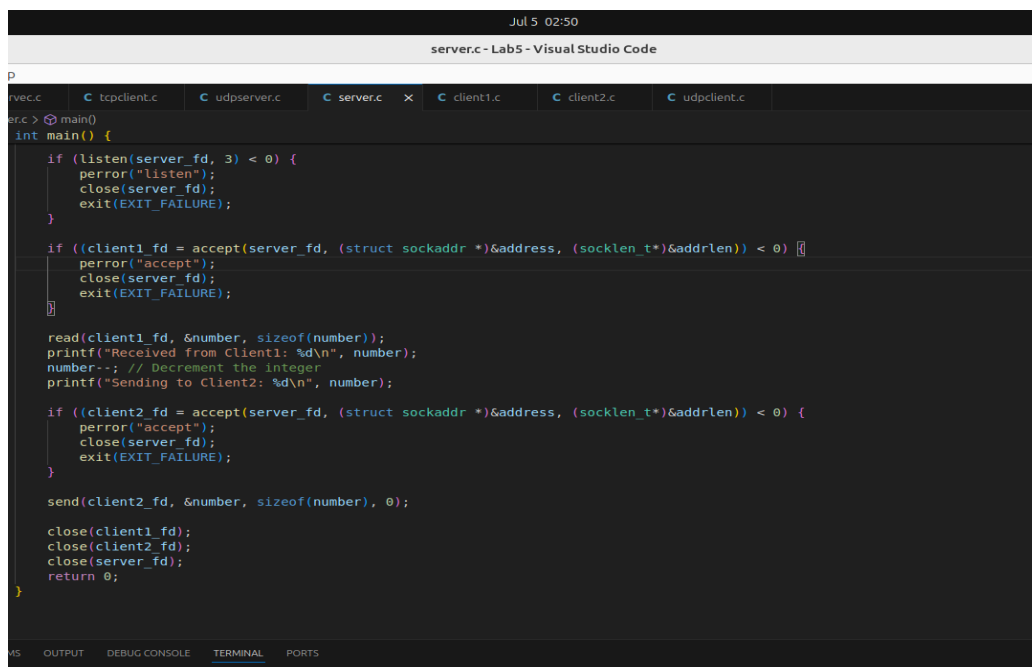
Client2 prints the received character.

Task 4: Modify your program in task 3 such that the data type of the message should be integer and the server should decrement the integer before transmitting it to client2.

Code

Modifications made

Server.c



```
server.c - Lab5 - Visual Studio Code

int main() {
    if (listen(server_fd, 3) < 0) {
        perror("listen");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    if ((client1_fd = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen)) < 0) {
        perror("accept");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    read(client1_fd, &number, sizeof(number));
    printf("Received from Client1: %d\n", number);
    number--; // Decrement the integer
    printf("Sending to Client2: %d\n", number);

    if ((client2_fd = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen)) < 0) {
        perror("accept");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    send(client2_fd, &number, sizeof(number), 0);

    close(client1_fd);
    close(client2_fd);
    close(server_fd);
    return 0;
}
```

Client1

```
 2  #include <stdlib.h>
 3  #include <unistd.h>
 4  #include <arpa/inet.h>
 5
 6  #define PORT 8080
 7
 8  int main() {
 9      int sock = 0;
10      struct sockaddr_in serv_addr;
11      int number = 10; // Sending integer 10
12
13      if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
14          printf("\n Socket creation error \n");
15          return -1;
16      }
17
18      serv_addr.sin_family = AF_INET;
19      serv_addr.sin_port = htons(PORT);
20
21      if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
22          printf("\nInvalid address/ Address not supported \n");
23          return -1;
24      }
25
26      if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
27          printf("\nConnection Failed \n");
28          return -1;
29      }
30
31      send(sock, &number, sizeof(number), 0);
32
33      close(sock);
34      return 0;
35 }
```

Client 2

```
 8  int main() {
 9      int sock = 0;
10      struct sockaddr_in serv_addr;
11      int number;
12
13      if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
14          printf("\n Socket creation error \n");
15          return -1;
16      }
17
18      serv_addr.sin_family = AF_INET;
19      serv_addr.sin_port = htons(PORT);
20
21      if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
22          printf("\nInvalid address/ Address not supported \n");
23          return -1;
24      }
25
26      if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
27          printf("\nConnection Failed \n");
28          return -1;
29      }
30
31      read(sock, &number, sizeof(number));
32      printf("Received from server: %d\n", number);
33
34      close(sock);
35      return 0;
36 }
37
```

Output

Server

```
slade@slade-VMware-Virtual-Platform:~/Lab5$ ./server
Received from Client1: 10
Sending to Client2: 9
slade@slade-VMware-Virtual-Platform:~/Lab5$ ./server
```

Client2

```
Received from server: 10
slade@slade-VMware-Virtual-Platform:~/Lab5$ ./client2
Received from server: 9
slade@slade-VMware-Virtual-Platform:~/Lab5$
```

Task5Modify for Float Value Transmission

Code

Server.c

I created a power function

```
// Function to calculate square root using Babylonian method
float sqrt_approx(float num) {
    float x = num;
    float y = 1.0;
    float epsilon = 0.00001; // Accuracy level

    while (x - y > epsilon) {
        x = (x + y) / 2;
        y = num / x;
    }
    return x;
}

// Function to calculate power of 1.5
float power1_5(float num) {
    return num * num * sqrt_approx(num);
}

int main() {
```

Reading from client one and increase by a power 1.5

```
84 // Reading data from Client1
85 read(client1_fd, &number, sizeof(number));
86 printf("Received from client1: %f\n", number);
87 number = power1_5(number); // Increase the float value by a power of 1.5
88 printf("Modified value: %f\n", number);
89
90 }
```

Client1.c

Sending float number

```
int sock = 0;
struct sockaddr_in serv_addr;
float number = 2.0; // Sending float value 2.0

if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    printf("\n 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("\nInvalid address/ Address not supported \n");
    return -1;
}
```

Client2.c

```
#define PORT 8080

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    float number;

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n 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("\nInvalid address/ Address not supported \n");
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        printf("\nConnection Failed \n");
        return -1;
    }

    read(sock, &number, sizeof(number));
    printf("Received from server: %f\n", number);
    close(sock);
    return 0;
}
```

Output

Server output

```
Received from server: 9
slade@slade-VMware-Virtual-Platform:~/Lab$ ls
client1 client1.c client2 client2.c server.c tcpclient tcpclient.c tcpserver tcpserver.c udpclient udpclient.c udpserver udpserver.c
slade@slade-VMware-Virtual-Platform:~/Lab$ gcc server.c -o server
/usr/bin/ld: /tmp/ccQXY1IY.o: in function 'main':
server.c:(.text+0x18c): undefined reference to 'pow'
collect2: error: ld returned 1 exit status
slade@slade-VMware-Virtual-Platform:~/Lab$ gcc server.c -o server
/usr/bin/ld: /tmp/ccpnP097.o: in function 'power1_5.0':
server.c:(.text+0x33): undefined reference to 'sqrt'
collect2: error: ld returned 1 exit status
slade@slade-VMware-Virtual-Platform:~/Lab$ gcc server.c -o server
slade@slade-VMware-Virtual-Platform:~/Lab$ ./server
Received from Client1: 2.000000
Modified value: 5.656863
slade@slade-VMware-Virtual-Platform:~/Lab$
```

Client2

```
slade@slade-VMware-Virtual-Platform:~/Lab$ gcc client2.c -o client
slade@slade-VMware-Virtual-Platform:~/Lab$ ./client
Received from server: 5.656863
slade@slade-VMware-Virtual-Platform:~/Lab$
```

Results:

- Client1 sends a float value to the server.
- The server increases the float value by a power of 1.5 using the custom function and prints both the received and the modified value.
- The server sends the modified float value to Client2.
- Client2 prints the received float value.

Conclusion

The series of tasks in this lab exercise provided a comprehensive hands-on experience with socket programming in C. Through the implementation of TCP and UDP servers and clients, as well as the creation of a multi-client communication setup, several key concepts and skills were developed:

Socket Programming Basics:

Understanding how to create, bind, and listen on sockets.

Establishing connections between clients and servers using TCP and UDP protocols.

Data Transmission:

Learning to send and receive various data types (strings, integers, and floats) between clients and servers.

Handling data transmission errors and ensuring reliable communication.

Inter-Process Communication:

Implementing a server that mediates communication between two clients.

Modifying transmitted data on the server before forwarding it to another client.