LAB L49+L50

22MIA1062
PRIYANSHU TEOTIA

EXPERIMENT 4

AIM: USING RASPBERRY (PI) DOING CLIENT AND SERVER CODE

PROCEDURE:

The **Raspberry Pi 4** is a powerful **single-board computer** that can be used in IoT applications. In a **Client-Server** IoT setup, Raspberry Pi 4 can act as either:

- A **Server** (collects sensor data and hosts a web service).
- A **Client** (requests and processes data from the server).

2. Components Required

Hardware Components

1. Two Raspberry Pi 4 Boards (or one Raspberry Pi and a computer)
2. Wi-Fi/Ethernet connection
3. Power supply for Raspberry Pi
4. Jumper Wires & Breadboard (for sensor connection)

Software Requirements

1. Raspberry Pi OS (Raspbian)
2. Flask (for the server)
3. Requests Library (for the client)

INPUT:

CODE:

SERVER CODE

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>


#define PORT 12345

#define BUFFER_SIZE 1024


int main() {

    int server_socket, client_socket;

    struct sockaddr_in server_addr, client_addr;

    socklen_t client_addr_len = sizeof(client_addr);

    char buffer[BUFFER_SIZE];


    // Create socket

    server_socket = socket(AF_INET, SOCK_STREAM, 0);

    if (server_socket == -1) {

        perror("Socket creation failed");

        exit(EXIT_FAILURE);

    }
```

```
// Configure server address
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY; // Listen on all interfaces
server_addr.sin_port = htons(PORT);

// Bind the socket to the address and port
if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
    perror("Bind failed");
    close(server_socket);
    exit(EXIT_FAILURE);
}

// Listen for incoming connections
if (listen(server_socket, 5) == -1) {
    perror("Listen failed");
    close(server_socket);
    exit(EXIT_FAILURE);
}
printf("Server listening on port %d...\n", PORT);

// Accept a connection from a client
client_socket = accept(server_socket, (struct sockaddr *)&client_addr, &client_addr_len);
if (client_socket == -1) {
    perror("Accept failed");

    close(server_socket);

    exit(EXIT_FAILURE);

}

printf("Client connected.\n");


// Receive data from the client

memset(buffer, 0, BUFFER_SIZE);

if (recv(client_socket, buffer, BUFFER_SIZE, 0) == -1) {

    perror("Receive failed");

    close(client_socket);

    close(server_socket);

    exit(EXIT_FAILURE);

}

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


// Send a response to the client

const char *response = "Hello from server!";

if (send(client_socket, response, strlen(response), 0) == -1) {

    perror("Send failed");

    close(client_socket);

    close(server_socket);

    exit(EXIT_FAILURE);

}
```

```
    printf("Response sent to client.\n");

    // Close sockets
    close(client_socket);
    close(server_socket);
    printf("Connection closed.\n");

    return 0;
}
```

CLIENT CODE

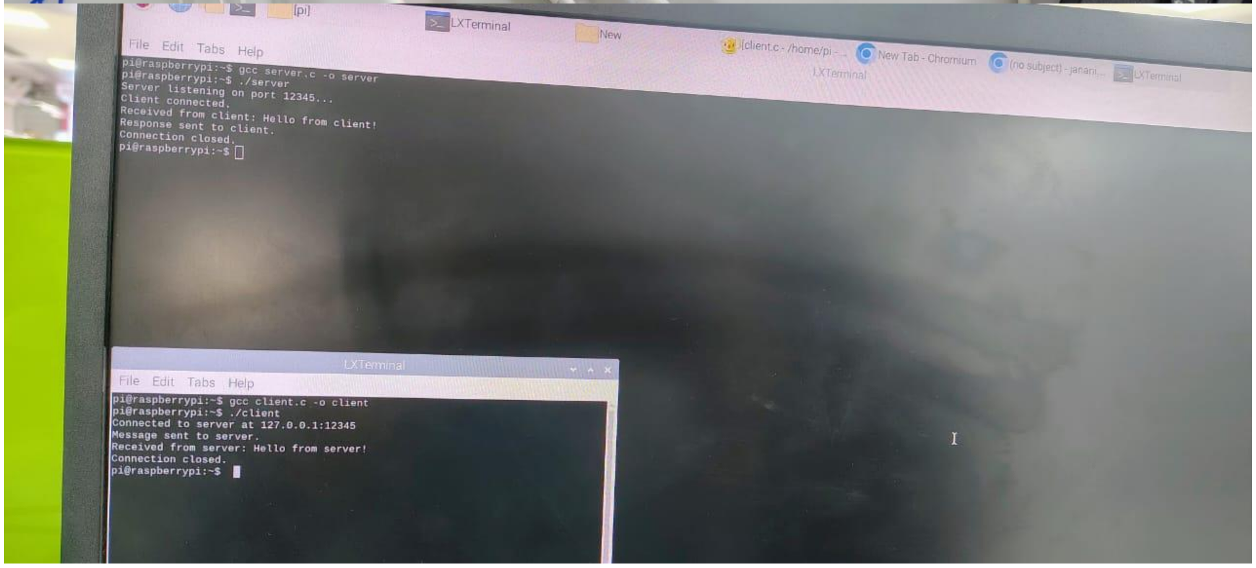
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define SERVER_IP "127.0.0.1" // Replace with server's IP address
#define PORT 12345
#define BUFFER_SIZE 1024

int main() {
    int client_socket;
    struct sockaddr_in server_addr;
```

```
char buffer[BUFFER_SIZE];


// Create socket
client_socket = socket(AF_INET, SOCK_STREAM, 0);
if (client_socket == -1) {
    perror("Socket creation failed");
    exit(EXIT_FAILURE);
}


// Configure server address
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(PORT);
if (inet_pton(AF_INET, SERVER_IP, &server_addr.sin_addr) <= 0) {
    perror("Invalid address/Address not supported");
    close(client_socket);
    exit(EXIT_FAILURE);
}


// Connect to the server
if (connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
    perror("Connection failed");
    close(client_socket);
    exit(EXIT_FAILURE);
}
```

```
printf("Connected to server at %s:%d\n", SERVER_IP, PORT);

// Send data to the server
const char *message = "Hello from client!";
if (send(client_socket, message, strlen(message), 0) == -1) {
    perror("Send failed");
    close(client_socket);
    exit(EXIT_FAILURE);
}
printf("Message sent to server.\n");

// Receive a response from the server
memset(buffer, 0, BUFFER_SIZE);
if (recv(client_socket, buffer, BUFFER_SIZE, 0) == -1) {
    perror("Receive failed");
    close(client_socket);
    exit(EXIT_FAILURE);
}
printf("Received from server: %s\n", buffer);

// Close the socket
close(client_socket);
printf("Connection closed.\n");
```

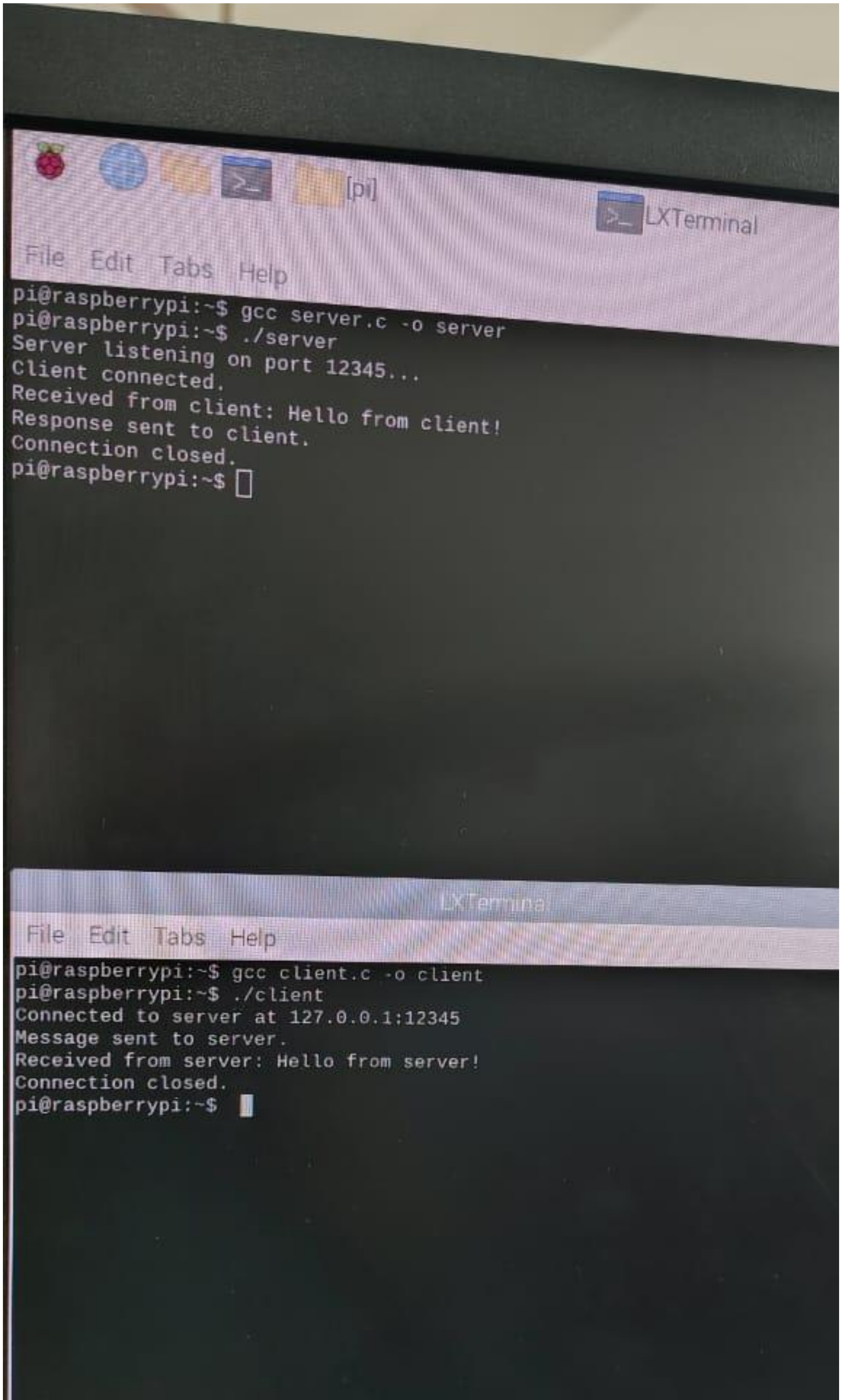
```
    return 0;
}
```

OUTPUT:



```
pi@raspberrypi:~$ gcc server.c -o server
pi@raspberrypi:~$ ./server
Server listening on port 12345...
Client connected.
Received from client: Hello from client!
Response sent to client.
Connection closed.
pi@raspberrypi:~$
```

```
pi@raspberrypi:~$ gcc client.c -o client
pi@raspberrypi:~$ ./client
Connected to server at 127.0.0.1:12345
Message sent to server.
Received from server: Hello from server!
Connection closed.
pi@raspberrypi:~$
```



```
pi@raspberrypi:~$ gcc server.c -o server
pi@raspberrypi:~$ ./server
Server listening on port 12345...
Client connected.
Received from client: Hello from client!
Response sent to client.
Connection closed.
pi@raspberrypi:~$
```

```
pi@raspberrypi:~$ gcc client.c -o client
pi@raspberrypi:~$ ./client
Connected to server at 127.0.0.1:12345
Message sent to server.
Received from server: Hello from server!
Connection closed.
pi@raspberrypi:~$
```

RESULT:

Hence , client & server code is executed in the terminal successfully.