Project II

EEL 4781 – Computer Networks

Section No:	01
Instructor:	Dr. Ming Yu
Title:	Two-Way Socket Communication for Client-Server Chat Program
Name:	Ruth Massock
FSUID:	rgm23b
Date Delivered:	11/14/2024

Contents

1. Overview

The purpose of this project was to implement a two-way communication system between a client and server using socket programming in any language put I used C. This application simulates a simple chat between two users, where the client sends a message to the server, the server reads the message and replies, and both parties display the messages on their respective terminals. Communication terminates when the client sends an “Over” message.

2. Design and Implementation

The project consists of two separate programs:

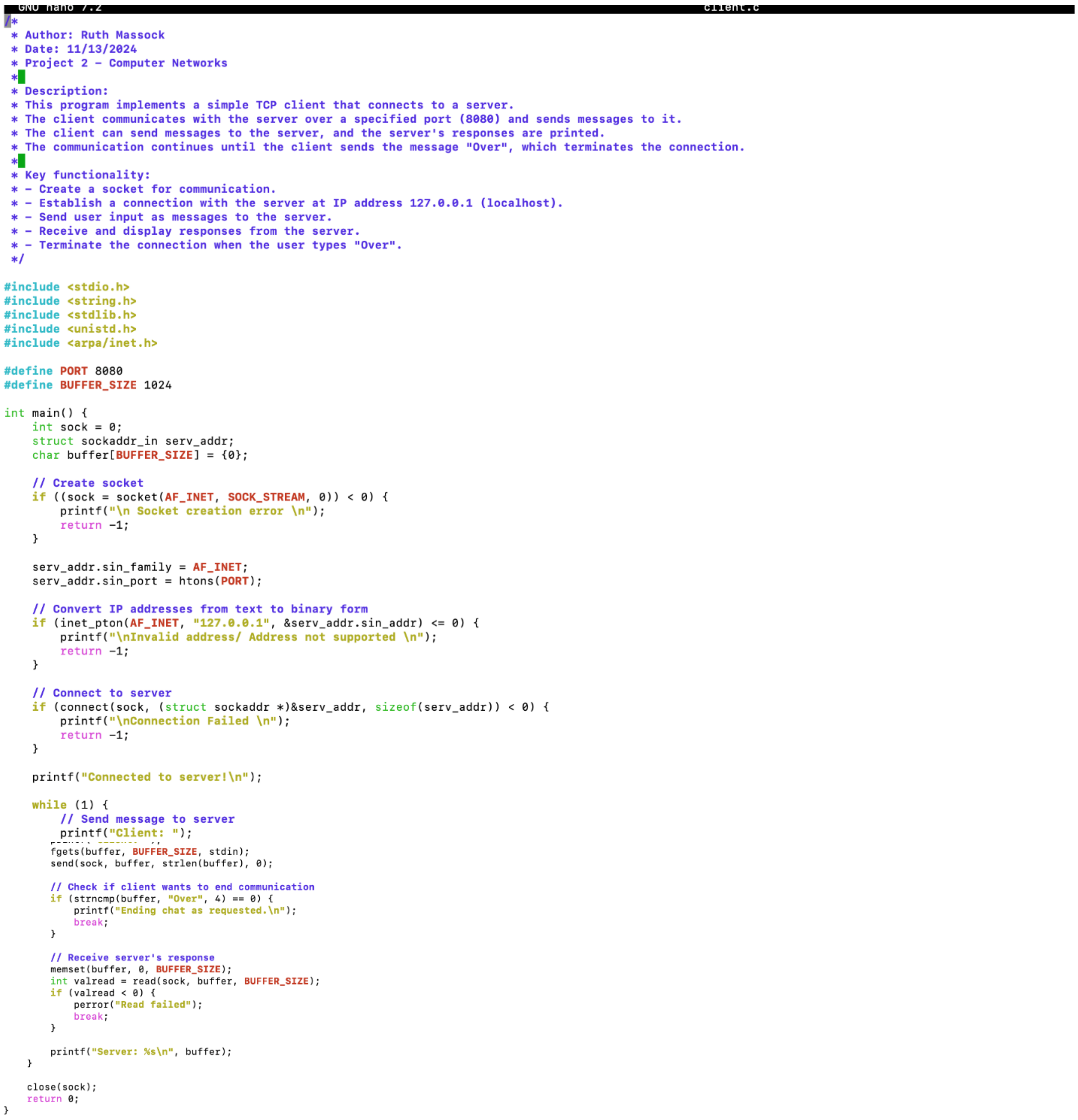
1. **Server Program (server.c)**: This program initializes a socket, binds it to a specific port, listens for incoming client connections, and facilitates the reading and replying of messages.
2. **Client Program (client.c)**: This program creates a socket and connects to the server, allowing it to send messages and receive responses from the server.

Key Functionalities:

- **Server Program**:
    - Creates a socket and binds it to PORT 8080.
    - Listens for client connections and accepts them.
    - Reads messages sent by the client and displays them.
    - Responds with a predefined message: "Message received".
    - Ends communication if the client sends the message "Over".
- **Client Program**:
    - Connects to the server on PORT 8080.
    - Takes user input, sends it to the server, and waits for the server’s response.
    - Displays the server’s reply after each message.
    - Ends communication by sending "Over".

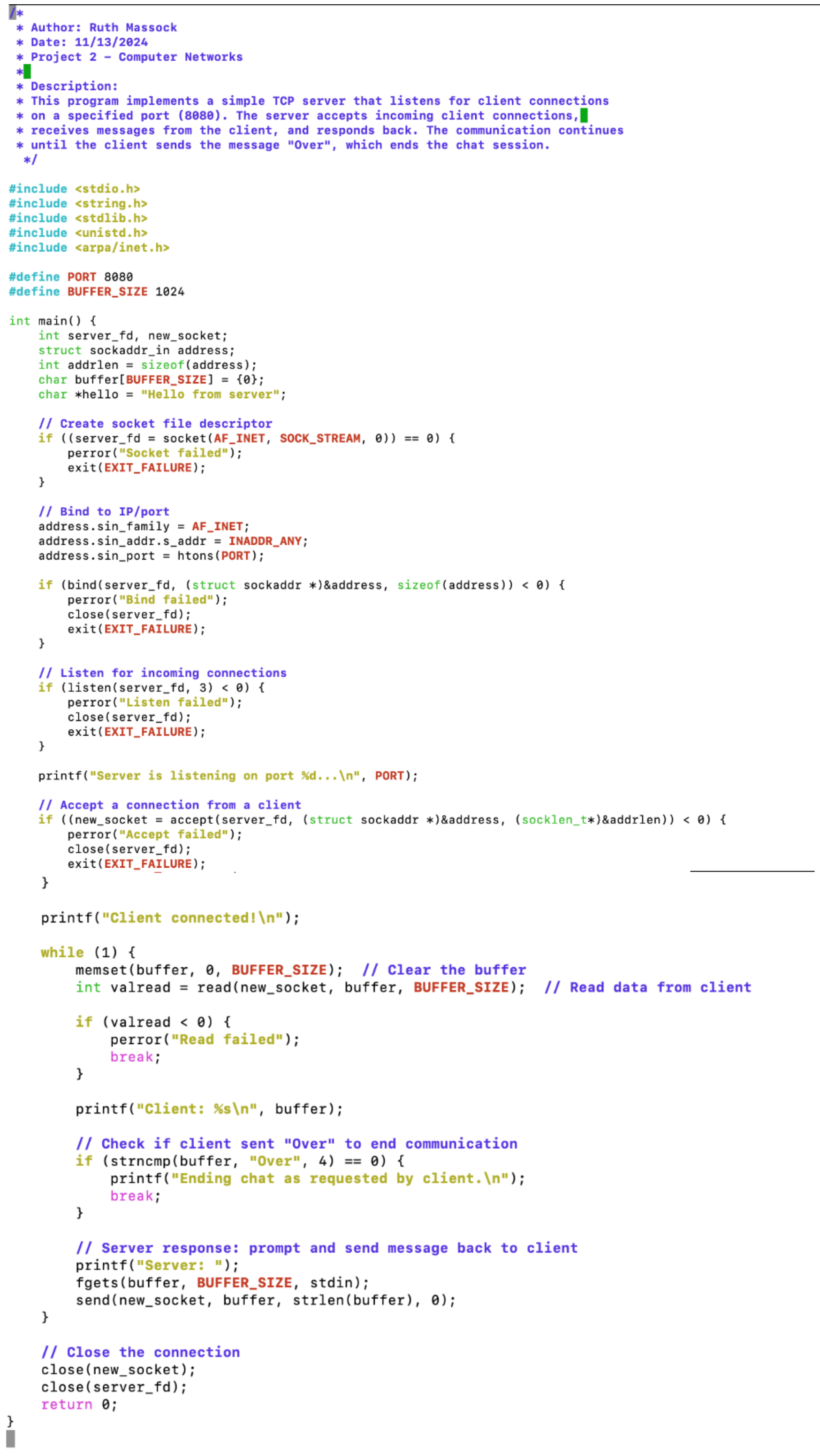
3. Code

client.c



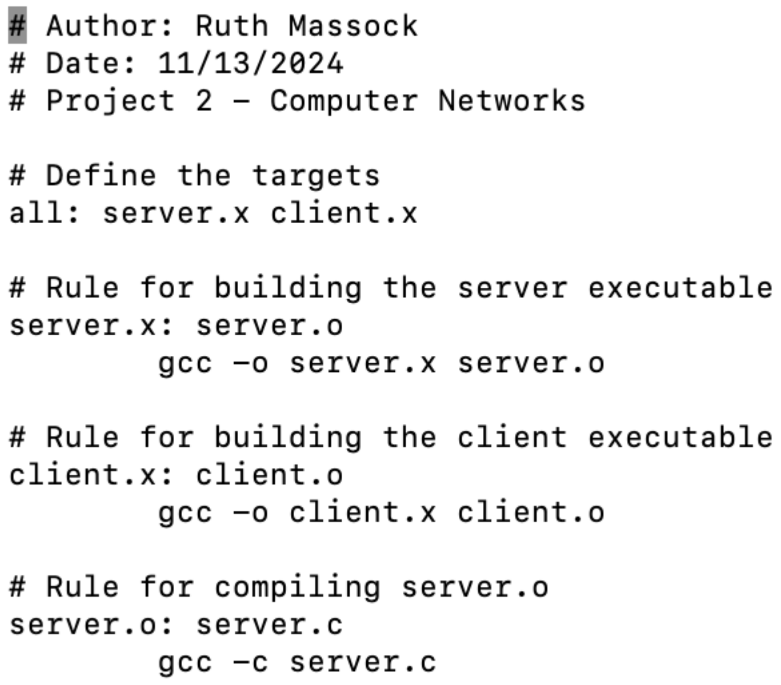
```
GNU nano 7.2                                                          client.c
/*
 * Author: Ruth Massock
 * Date: 11/13/2024
 * Project 2 - Computer Networks
 *
 * Description:
 * This program implements a simple TCP client that connects to a server.
 * The client communicates with the server over a specified port (8080) and sends messages to it.
 * The client can send messages to the server, and the server's responses are printed.
 * The communication continues until the client sends the message "Over", which terminates the connection.
 *
 * Key functionality:
 * - Create a socket for communication.
 * - Establish a connection with the server at IP address 127.0.0.1 (localhost).
 * - Send user input as messages to the server.
 * - Receive and display responses from the server.
 * - Terminate the connection when the user types "Over".
 */

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

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[BUFFER_SIZE] = {0};

    // Create socket
    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);

    // Convert IP addresses from text to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }

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

    printf("Connected to server!\n");

    while (1) {
        // Send message to server
        printf("Client: ");
        fgets(buffer, BUFFER_SIZE, stdin);
        send(sock, buffer, strlen(buffer), 0);

        // Check if client wants to end communication
        if (strncmp(buffer, "Over", 4) == 0) {
            printf("Ending chat as requested.\n");
            break;
        }

        // Receive server's response
        memset(buffer, 0, BUFFER_SIZE);
        int valread = read(sock, buffer, BUFFER_SIZE);
        if (valread < 0) {
            perror("Read failed");
            break;
        }

        printf("Server: %s\n", buffer);
    }

    close(sock);
    return 0;
}
```

server.c



```c
/*
 * Author: Ruth Massock
 * Date: 11/13/2024
 * Project 2 - Computer Networks
 *
 * Description:
 * This program implements a simple TCP server that listens for client connections
 * on a specified port (8080). The server accepts incoming client connections,
 * receives messages from the client, and responds back. The communication continues
 * until the client sends the message "Over", which ends the chat session.
  */

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

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    char buffer[BUFFER_SIZE] = {0};
    char *hello = "Hello from server";

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

    // Bind to IP/port
    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) {
        perror("Bind failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

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

    printf("Server is listening on port %d...\n", PORT);

    // Accept a connection from a client
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen)) < 0) {
        perror("Accept failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    printf("Client connected!\n");

    while (1) {
        memset(buffer, 0, BUFFER_SIZE);  // Clear the buffer
        int valread = read(new_socket, buffer, BUFFER_SIZE);  // Read data from client

        if (valread < 0) {
            perror("Read failed");
            break;
        }

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

        // Check if client sent "Over" to end communication
        if (strncmp(buffer, "Over", 4) == 0) {
            printf("Ending chat as requested by client.\n");
            break;
        }

        // Server response: prompt and send message back to client
        printf("Server: ");
        fgets(buffer, BUFFER_SIZE, stdin);
        send(new_socket, buffer, strlen(buffer), 0);
    }

    // Close the connection
    close(new_socket);
    close(server_fd);
    return 0;
}
```

makefile



```
# Author: Ruth Massock
# Date: 11/13/2024
# Project 2 - Computer Networks

# Define the targets
all: server.x client.x

# Rule for building the server executable
server.x: server.o
	gcc -o server.x server.o

# Rule for building the client executable
client.x: client.o
	gcc -o client.x client.o

# Rule for compiling server.o
server.o: server.c
	gcc -c server.c
```

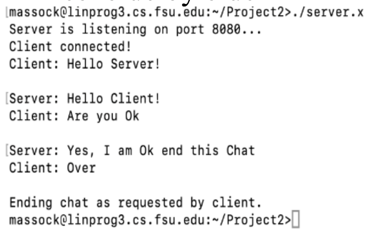
4. Testing and Results

For testing, we started the server program first, followed by the client program. Messages were sent from the client to the server, and we observed the following:

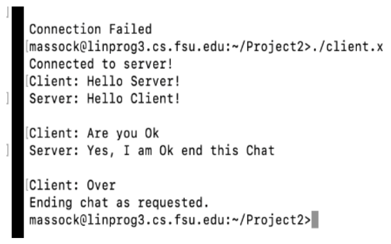
- **Message Transmission**: Messages typed in the client program appeared in the server program’s terminal, verifying that data was being transmitted correctly.
- **Server Replies**: The server responded with "Message received," which was displayed in the client terminal after each message.
- **Termination**: When the client sent "Over," both programs terminated gracefully, confirming proper end-of-session handling.

Sample Test Case

All test side by side

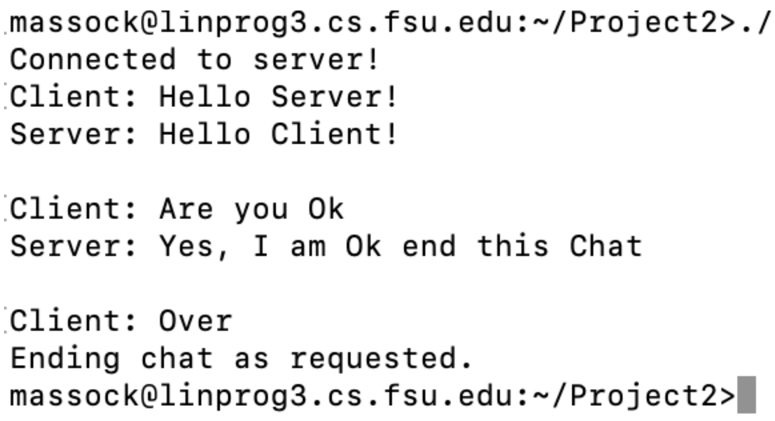


```
massock@linprog3.cs.fsu.edu:~/Project2>./server.x
Server is listening on port 8080...
Client connected!
Client: Hello Server!

Server: Hello Client!
Client: Are you Ok

Server: Yes, I am Ok end this Chat
Client: Over

Ending chat as requested by client.
massock@linprog3.cs.fsu.edu:~/Project2>
```



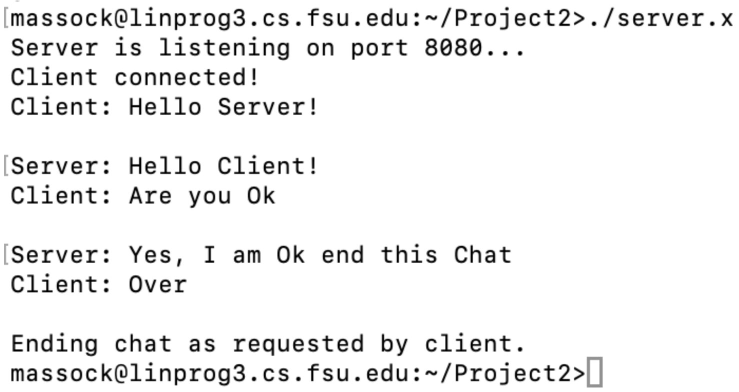
```
Connection Failed
massock@linprog3.cs.fsu.edu:~/Project2>./client.x
Connected to server!
Client: Hello Server!
Server: Hello Client!

Client: Are you Ok
Server: Yes, I am Ok end this Chat

Client: Over
Ending chat as requested.
massock@linprog3.cs.fsu.edu:~/Project2>
```

Client output



```
massock@linprog3.cs.fsu.edu:~/Project2>./client.x
Connected to server!
Client: Hello Server!
Server: Hello Client!

Client: Are you Ok
Server: Yes, I am Ok end this Chat

Client: Over
Ending chat as requested.
massock@linprog3.cs.fsu.edu:~/Project2>
```

Server output



```
massock@linprog3.cs.fsu.edu:~/Project2>./server.x
Server is listening on port 8080...
Client connected!
Client: Hello Server!

Server: Hello Client!
Client: Are you Ok

Server: Yes, I am Ok end this Chat
Client: Over

Ending chat as requested by client.
massock@linprog3.cs.fsu.edu:~/Project2>
```

5. Conclusion

The two-way communication socket application was successfully implemented in C, simulating a simple chat system between a client and server. The program demonstrated bidirectional data transfer, message display on both terminals, and clean session termination upon receiving the "Over" message from the client. This project reinforced key concepts in socket programming, including creating, binding, listening, connecting, reading, and writing sockets. This project provides a foundation for more complex networked applications, including real-time chat systems and interactive client-server applications.