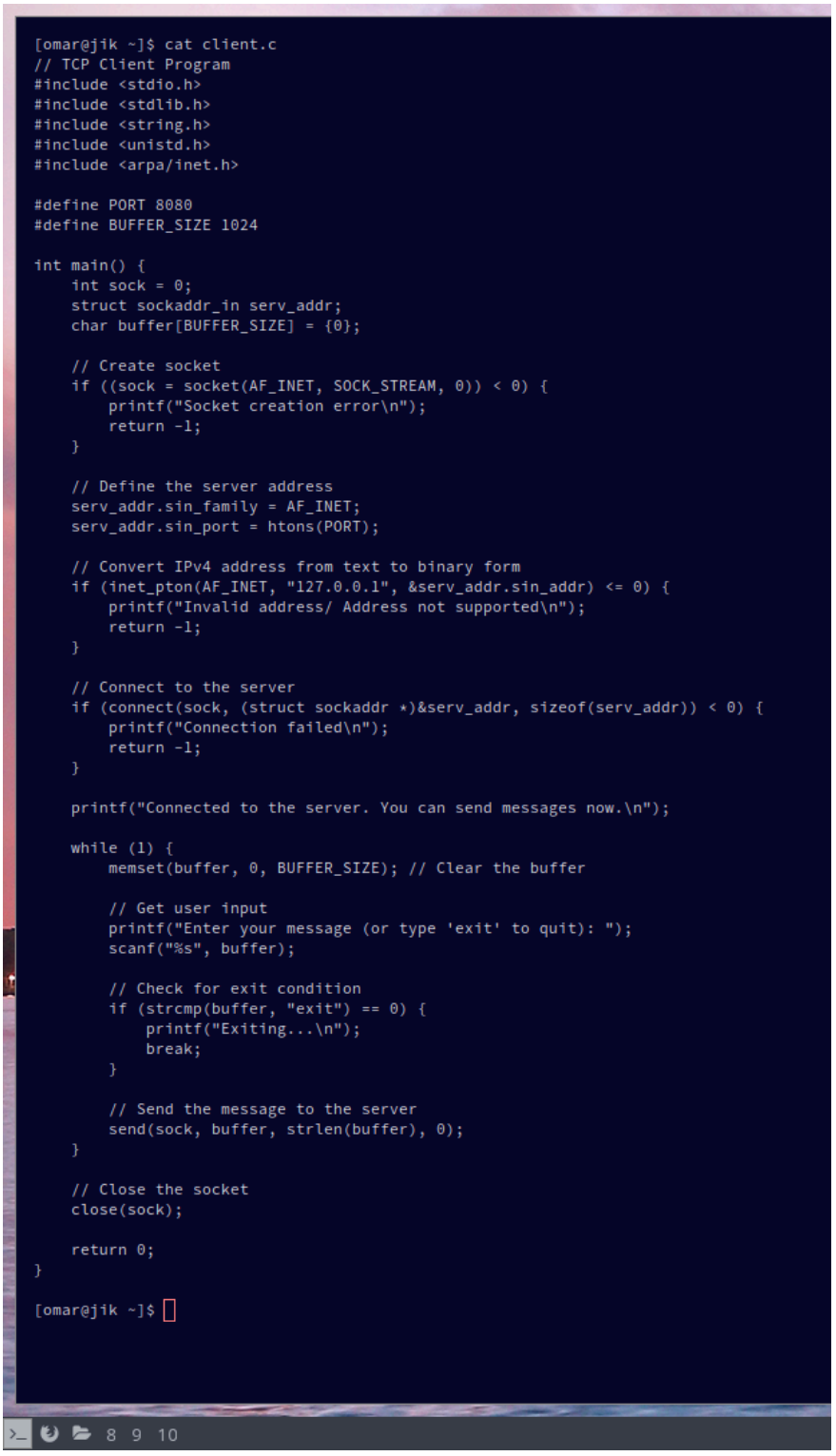

# LAB EXERCISE II : Socket Programming

**Name : Ivan J Madathil**

**Reg. No: 23BRS1127**


[omar@jik ~]$ gcc -o server server.c
[omar@jik ~]$ ./server
Server is listening on port 8080
Client connected. Waiting for messages...
Message from client: IvanJMadathil
Message from client: 23BRS1127
Message from client: hello
Message from client: testing
Message from client: 12
Client disconnected.
[omar@jik ~]$
[omar@jik ~]$ gcc -o client client.c
[omar@jik ~]$ ./
-bash: ./: Is a directory
[omar@jik ~]$ ./client
Connected to the server. You can send messages now.
Enter your message (or type 'exit' to quit): Ivan J Madathil
Enter your message (or type 'exit' to quit): Enter your message (or type 'exit' to quit): Enter your message (or type 'exit' to quit): 23BRS1127
Enter your message (or type 'exit' to quit):
hello
Enter your message (or type 'exit' to quit): testing
Enter your message (or type 'exit' to quit): 12
Enter your message (or type 'exit' to quit): exit
Exiting...
[omar@jik ~]$



```
[omar@jik ~]$ cat client.c
// TCP Client Program
#include <stdio.h>
#include <stdlib.h>
#include <string.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("Socket creation error\n");
        return -1;
    }

    // Define the server address
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 address from text to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        printf("Invalid address/ Address not supported\n");
        return -1;
    }

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

    printf("Connected to the server. You can send messages now.\n");

    while (1) {
        memset(buffer, 0, BUFFER_SIZE); // Clear the buffer

        // Get user input
        printf("Enter your message (or type 'exit' to quit): ");
        scanf("%s", buffer);

        // Check for exit condition
        if (strcmp(buffer, "exit") == 0) {
            printf("Exiting...\n");
            break;
        }

        // Send the message to the server
        send(sock, buffer, strlen(buffer), 0);
    }

    // Close the socket
    close(sock);

    return 0;
}

[omar@jik ~]$
```

```
[omar@jik ~]$ cat server.c
// TCP Server Program
#include <stdio.h>
#include <stdlib.h>
#include <string.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 opt = 1;
    int addrlen = sizeof(address);
    char buffer[BUFFER_SIZE] = {0};

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

    // Set socket options
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt))) {
        perror("Setsockopt failed");
        exit(EXIT_FAILURE);
    }

    // Define the server address
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

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

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

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

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

    printf("Client connected. Waiting for messages...\n");

    while (1) {
        memset(buffer, 0, BUFFER_SIZE); // Clear the buffer

        // Read input from the client
        int valread = read(new_socket, buffer, BUFFER_SIZE);
        if (valread <= 0) {
            printf("Client disconnected.\n");
            break;
        }

        // Display the message received from the client
        printf("Message from client: %s\n", buffer);
    }

    // Close the sockets
    close(new_socket);
    close(server_fd);

    return 0;
}
```