# Computer Networks Lab-3

# Socket Programming

**Akhil P S – 24MCS1018**

**MTech CSE**

1. How to write a C program to connect a server and client using sockets, supporting both single and multiple client connections?

## Server.c

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <pthread.h>

#define PORT 8080

#define BUFFER_SIZE 1024

// Function to handle client connections

void *handle_client(void *client_socket) {

int sock = *(int *)client_socket;

char buffer[BUFFER_SIZE];

int bytes_read;

// Communicate with the client

while ((bytes_read = read(sock, buffer, sizeof(buffer) - 1)) > 0) {

buffer[bytes_read] = '\0'; // Null-terminate the string

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

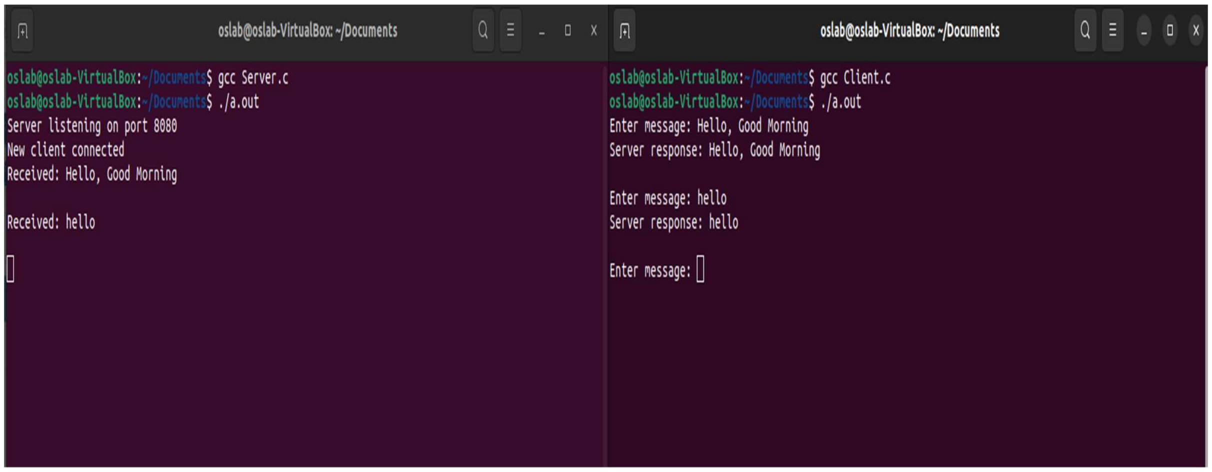
```
send(sock, buffer, bytes_read, 0); // Echo back the received message
}
// Close the socket and exit the thread
close(sock);
printf("Client disconnected\n");
free(client_socket);
return NULL;
}
int main() {
int server_fd, new_socket;
struct sockaddr_in address;
int addrlen = sizeof(address);
// Create socket file descriptor
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
perror("Socket failed");
exit(EXIT_FAILURE);
}
// Bind the socket to the specified 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);
}
// Start listening for incoming connections
if (listen(server_fd, 3) < 0) {
```

```
perror("Listen failed");

close(server_fd);

exit(EXIT_FAILURE);

}

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

// Accept incoming connections in a loop

while (1) {

int *client_socket = malloc(sizeof(int));

if ((*client_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen)) <
0) {

perror("Accept failed");

free(client_socket);

continue;

}

printf("New client connected\n");

// Create a new thread for the client

pthread_t thread_id;

if (pthread_create(&thread_id, NULL, handle_client, (void *)client_socket) != 0) {

perror("Thread creation failed");

free(client_socket);

} else {

pthread_detach(thread_id); // Detach the thread to free resources on exit

}

}

// Close the server socket (this line will never be reached in this example)

close(server_fd);

return 0;

}
```

**Client.c**

```
#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;

}

serv_addr.sin_family = AF_INET;

serv_addr.sin_port = htons(PORT);

// Convert IPv4 and IPv6 addresses 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;
}
// Communicate with the server
while (1) {
printf("Enter message: ");
fgets(buffer, BUFFER_SIZE, stdin);
// Send message to server
send(sock, buffer, strlen(buffer), 0);
// Read response from server
int bytes_read = read(sock, buffer, sizeof(buffer) - 1);
if (bytes_read > 0) {
buffer[bytes_read] = '\0'; // Null-terminate the string
printf("Server response: %s\n", buffer);
}
// Exit if the user types "exit"
if (strncmp(buffer, "exit", 4) == 0) {
break;
}
}
// Close the socket
close(sock);
return 0;
}
```

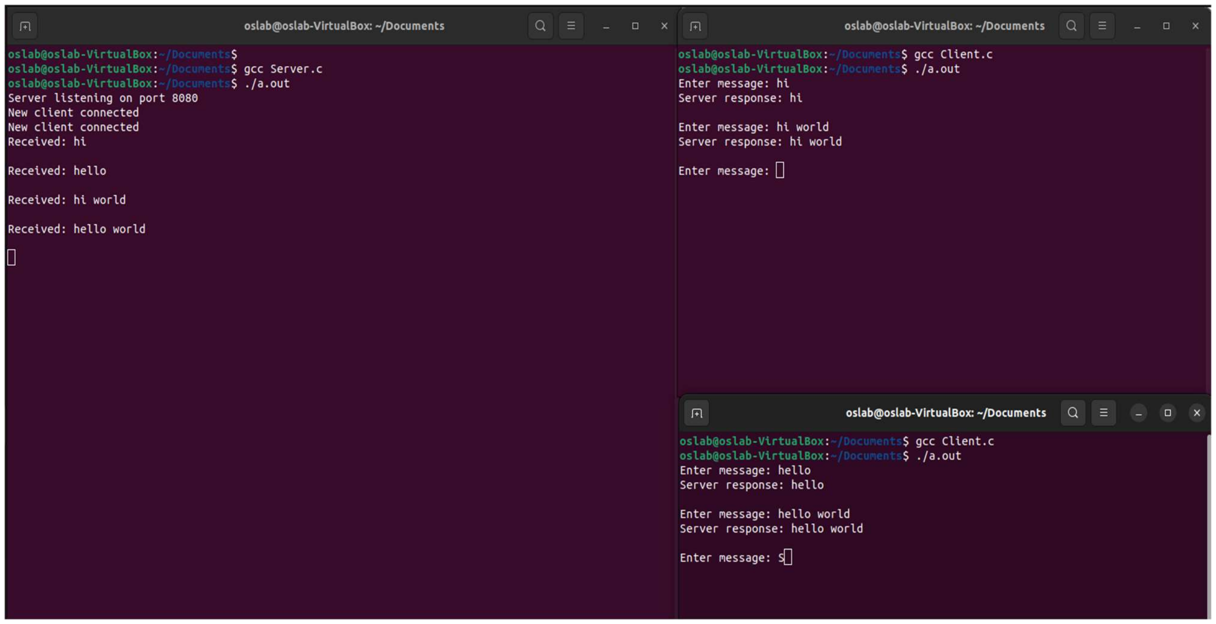
## Output(Single Server, Single Client):



```
oslab@oslab-VirtualBox:~/Documents$ gcc Server.c
oslab@oslab-VirtualBox:~/Documents$ ./a.out
Server listening on port 8080
New client connected
Received: Hello, Good Morning

Received: hello

```

```
oslab@oslab-VirtualBox:~/Documents$ gcc Client.c
oslab@oslab-VirtualBox:~/Documents$ ./a.out
Enter message: Hello, Good Morning
Server response: Hello, Good Morning

Enter message: hello
Server response: hello

Enter message:
```

## Output(Single Server, Multiple Client):



```
oslab@oslab-VirtualBox:~/Documents$
oslab@oslab-VirtualBox:~/Documents$ gcc Server.c
oslab@oslab-VirtualBox:~/Documents$ ./a.out
Server listening on port 8080
New client connected
New client connected
Received: hi

Received: hello

Received: hi world

Received: hello world

```

```
oslab@oslab-VirtualBox:~/Documents$ gcc Client.c
oslab@oslab-VirtualBox:~/Documents$ ./a.out
Enter message: hi
Server response: hi

Enter message: hi world
Server response: hi world

Enter message:
```

```
oslab@oslab-VirtualBox:~/Documents$ gcc Client.c
oslab@oslab-VirtualBox:~/Documents$ ./a.out
Enter message: hello
Server response: hello

Enter message: hello world
Server response: hello world

Enter message: S
```

2. How to write a C program to connect a server and client using sockets, where the client sends text and the server responds with the text converted to all caps?

## Server.c

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

```
#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <pthread.h>

#include <arpa/inet.h>

#include <ctype.h>


#define PORT 8080

#define BUFFER_SIZE 1024


// Function to convert a message to uppercase

void to_uppercase(char *msg) {

    for (int i = 0; msg[i]; i++) {

        msg[i] = toupper(msg[i]);

    }

}


// Thread function to handle communication with each client

void *handle_client(void *arg) {

    int client_socket = *(int *)arg;

    char buffer[BUFFER_SIZE];

    int bytes_read;


    // Receive messages from the client

    while ((bytes_read = read(client_socket, buffer, sizeof(buffer) - 1)) > 0) {

        buffer[bytes_read] = '\0';  // Null-terminate the received message


        // Convert the message to uppercase
        to_uppercase(buffer);

        // Send the uppercase message back to the client
        send(client_socket, buffer, strlen(buffer), 0);
    }

    // Close the client socket when done
    if (bytes_read == 0) {
        printf("Client disconnected\n");
    } else {
        perror("Read failed");
    }
    close(client_socket);
    return NULL;
}

int main() {
    int server_fd, client_socket;
    struct sockaddr_in address;
    socklen_t addr_len = sizeof(address);
    pthread_t thread_id;

    // Create the server socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket creation failed");
        return -1;
    }
```

```
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;  // Listen on any available network interface
address.sin_port = htons(PORT);

// Bind the socket to the specified port
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
    perror("Bind failed");
    return -1;
}

// Listen for incoming connections
if (listen(server_fd, 3) < 0) {
    perror("Listen failed");
    return -1;
}

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

// Accept incoming client connections and spawn a thread for each client
while (1) {
    if ((client_socket = accept(server_fd, (struct sockaddr *)&address, &addr_len)) < 0) {
        perror("Accept failed");
        continue;
    }

    printf("New client connected\n");

    // Create a new thread to handle the client
```

```
        if (pthread_create(&thread_id, NULL, handle_client, (void *)&client_socket) != 0) {
            perror("Thread creation failed");
            close(client_socket);
        } else {
            pthread_detach(thread_id);  // Detach the thread so it cleans up automatically
        }
    }

    // Close the server socket (this will never be reached in this infinite loop)
    close(server_fd);
    return 0;
}
```

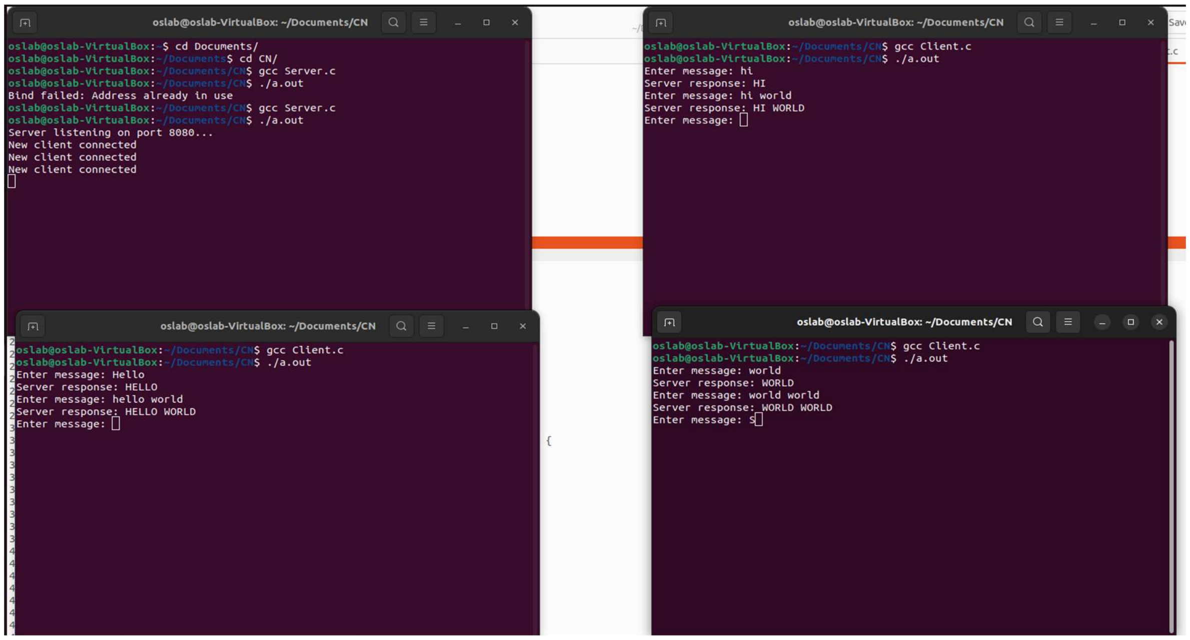
**Client.c**

```
#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;
    struct sockaddr_in server_addr;
    char buffer[BUFFER_SIZE];
```

```
// Create socket
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("Socket creation failed");
    return -1;
}

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

// Convert IPv4 address from text to binary form
if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0) {
    perror("Invalid address");
    return -1;
}

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

// Communicate with the server
while (1) {
    printf("Enter message: ");
    fgets(buffer, BUFFER_SIZE, stdin);

    // Remove newline character from the input message
    buffer[strcspn(buffer, "\n")] = '\0';
```

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

        // Receive the transformed message from the server
        int bytes_read = read(sock, buffer, sizeof(buffer) - 1);
        if (bytes_read > 0) {
            buffer[bytes_read] = '\0';
            printf("Server response: %s\n", buffer);
        }

        if (strncmp(buffer, "exit", 4) == 0) {
            break;
        }
    }

    // Close the socket
    close(sock);
    return 0;
}
```

**Output(Single Server, Multiple Clients):**



```
oslab@oslab-VirtualBox:~$ cd Documents/
oslab@oslab-VirtualBox:~/Documents$ cd CN/
oslab@oslab-VirtualBox:~/Documents/CN$ gcc Server.c
oslab@oslab-VirtualBox:~/Documents/CN$ ./a.out
Bind failed: Address already in use
oslab@oslab-VirtualBox:~/Documents/CN$ gcc Server.c
oslab@oslab-VirtualBox:~/Documents/CN$ ./a.out
Server listening on port 8080...
New client connected
New client connected
New client connected
```

```
oslab@oslab-VirtualBox:~/Documents/CN$ gcc Client.c
oslab@oslab-VirtualBox:~/Documents/CN$ ./a.out
Enter message: hi
Server response: HI
Enter message: hi world
Server response: HI WORLD
Enter message: 
```

```
oslab@oslab-VirtualBox:~/Documents/CN$ gcc Client.c
oslab@oslab-VirtualBox:~/Documents/CN$ ./a.out
Enter message: Hello
Server response: HELLO
Enter message: hello world
Server response: HELLO WORLD
Enter message: 
```

```
oslab@oslab-VirtualBox:~/Documents/CN$ gcc Client.c
oslab@oslab-VirtualBox:~/Documents/CN$ ./a.out
Enter message: world
Server response: WORLD
Enter message: world world
Server response: WORLD WORLD
Enter message: S
```