Assignment - 2
CSCI 21023- Data Communication and Networking
Faculty of Computing & Technology
University of Kelaniya

MULTI-CLIENT CHAT ROOM SERVER
USING SOCKET PROGRAMMING IN C

*******~Option – A~*******

Y.P. Viduruwan (CS/2020/006)
M.G.P. Jeewantha (CS/2020/027)

OVERVIEW

1. Introduction:

- This document presents a concise guide for implementing a multi-client chat room server in C using socket programming. The server allows multiple clients to connect, communicate, and share messages in real-time.

2. Implementation:

- The server is implemented using the C programming language and relies on the socket programming paradigm. It supports simultaneous connections from multiple clients.

3. Dependencies:

- Ensure that the following dependencies are met:
  - C Compiler (e.g., GCC)
  - Basic understanding of socket programming concepts
  - Linux Terminal or Command Prompt for execution.

4. Conclusion:

- This document provides a brief overview of a multi-client chat room server implemented in C using socket programming. Users can modify and expand upon this foundation to enhance features and functionality.

❖ Server Program

1. Server Program Overview:

- The server program facilitates a multi-client chat room environment, allowing users to connect, share messages, and interact in real-time. Implemented in C using socket programming, the server manages incoming connections, assigns unique names to clients, and handles message broadcasting.

2. Dependencies:

- The server program relies on standard C libraries and requires a C compiler (e.g., GCC) for compilation.

3. Features:

- **Dynamic Client Management:**
  - The server dynamically handles connections and disconnections of clients.

  Broadcast Messaging:
  - Messages sent by one client are broadcast to all other connected clients.

****

Set the file path:

```
vboxuser@Ubuntu:~$ cd /media/sf_ubuntu_directory/link/
vboxuser@Ubuntu:/media/sf_ubuntu_directory/link$
```

Compile the file:

```
vboxuser@Ubuntu:/media/sf_ubuntu_directory/link$ gcc server_2.c -o server_2
vboxuser@Ubuntu:/media/sf_ubuntu_directory/link$ make server_2
make: 'server_2' is up to date.
vboxuser@Ubuntu:/media/sf_ubuntu_directory/link$
```

Execute & run:

```
vboxuser@Ubuntu:/media/sf_ubuntu_directory/link$ ./server_2 9988
Enter the port number: 9988
=== WELCOME TO THE CHATROOM ===
```

➢ Server Program code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>
```

```c
#define LENGTH_NAME 31
#define LENGTH_MSG 101
#define LENGTH_SEND 201
#define MAX_CLIENTS 10

typedef struct ClientNode {
    int sockfd;
    struct ClientNode *next;
    char name[LENGTH_NAME];
} ClientNode;

ClientNode *root, *now;
int server_sockfd = 0;

// Utility function to send a message to all clients except the sender
void broadcast_message(char *s, int sockfd) {
    ClientNode *tmp = root;
    while (tmp != NULL) {
        if (tmp->sockfd != sockfd) { // Send to all clients except the sender
            if (send(tmp->sockfd, s, LENGTH_SEND, 0) == -1) {
                perror("send error");
                // Continue to the next client if sending fails
            }
        }
        tmp = tmp->next;
    }
}

// Function to handle Ctrl+C signal
void catch_ctrl_c_and_exit(int sig) {
    ClientNode *tmp;
    while (root != NULL) {
        close(root->sockfd);
        tmp = root;
        root = root->next;
        free(tmp);
    }
    printf("Bye\n");
    exit(EXIT_SUCCESS);
}
```

```
// Function to handle each client
void *handle_client(void *arg) {
    int sockfd = *((int *)arg);
    char name[LENGTH_NAME] = {};
    char recv_buffer[LENGTH_MSG] = {};
    char send_buffer[LENGTH_SEND] = {};

    // Naming
    if (recv(sockfd, name, LENGTH_NAME, 0) <= 0 || strlen(name) < 2 ||
strlen(name) >= LENGTH_NAME - 1) {
        printf("Didn't enter the name.\n");
        close(sockfd);
        return NULL;
    }

    // Adding client to the list
    ClientNode *new_node = (ClientNode *)malloc(sizeof(ClientNode));
    new_node->sockfd = sockfd;
    new_node->next = NULL;
    //strncpy(new_node->name, name, LENGTH_NAME);
    if (root == NULL) { // First client
        root = new_node;
       // now = new_node;
    } else { // Append to the list
        ClientNode *current =root;
        while (current->next!=NULL){
            current=current->next;
        }
        current->next=new_node;
    }

    // Announce new client
    sprintf(send_buffer, "%s has joined", name);
    broadcast_message(send_buffer, sockfd);

    // Receive messages
    while (1) {
        int receive = recv(sockfd, recv_buffer, LENGTH_MSG, 0);
        if (receive > 0) {
            if (strlen(recv_buffer) == 0) {
```

```
                continue;
            }
            printf("%s : is send massage to other
clients:  %s\n",name,recv_buffer);
            sprintf(send_buffer, "%s: %s", name, recv_buffer);
            broadcast_message(send_buffer, sockfd);
        } else if (receive == 0 || strcmp(recv_buffer, "exit") == 0) {
            sprintf(send_buffer, "%s has left", name);
            broadcast_message(send_buffer, sockfd);
            close(sockfd);
            return NULL;
        } else {
            perror("recv error");
            close(sockfd);
            return NULL;
        }
    }
}

int main() {
    signal(SIGINT, catch_ctrl_c_and_exit);

    // Get port number from the user
    int port;
    printf("Enter the port number: ");
    scanf("%d", &port);

    // Create socket
    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_sockfd == -1) {
        perror("Fail to create a socket.");
        exit(EXIT_FAILURE);
    }

    // Server address structure
    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(port); // Port

    // Bind
```

```
    if (bind(server_sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr))
< 0) {
        perror("Bind error");
        return -1;
    }

    // Listen
    if (listen(server_sockfd, 5) < 0) {
        perror("Listen error");
        return -1;
    }

    printf("=== WELCOME TO THE CHATROOM ===\n");
    int conected_clients=10;

    while (1) {
        if(conected_clients < MAX_CLIENTS){
            printf("Maximum number of clients reached.");
            continue;
        }
        struct sockaddr_in client_addr;
        socklen_t client_addr_len = sizeof(client_addr);
        int client_sockfd = accept(server_sockfd, (struct sockaddr
*)&client_addr, &client_addr_len);

        if(client_sockfd < 0){
            perror("Accept error");
            continue; // Continue to the next iteration.
        }

        // Client IP
        char client_ip[INET_ADDRSTRLEN] = {};
        inet_ntop(AF_INET, &client_addr.sin_addr, client_ip, INET_ADDRSTRLEN);

        printf("Client %s:%d connected.\n", client_ip,
ntohs(client_addr.sin_port));

        // Create a thread for each client
        pthread_t tid;
        if (pthread_create(&tid, NULL, handle_client, (void *)&client_sockfd) !=
0) {
```

```
            perror("Thread create error");
            close(client_sockfd);
            continue;
        }
        conected_clients++;

    }
    close(server_sockfd);

    return 0;
}
```

❖ Client Program :

1. Client Program Overview:

- The client program is designed to connect to the multi-client chat room server, enabling users to participate in real-time communication. Implemented in C using socket programming, the client allows users to input messages, sends them to the server, and receives messages from other connected clients.

2. Dependencies:

- The client program relies on standard C libraries and requires a C compiler (e.g., GCC) for compilation.

2. Signal Handling:

- The client program handles the CTRL+C signal for graceful termination, ensuring proper closure of the client-side connection.

****

Set the file path:

```
vboxuser@Ubuntu:~$ cd /media/sf_ubuntu_directory/link/
vboxuser@Ubuntu:/media/sf_ubuntu_directory/link$
```

Compile the file:

```
vboxuser@Ubuntu:/media/sf_ubuntu_directory/link$ gcc client_2.c -o client_2
vboxuser@Ubuntu:/media/sf_ubuntu_directory/link$
```

Execute & run:

```
vboxuser@Ubuntu:/media/sf_ubuntu_directory/link$ ./client_2 127.0.0.1 9988
Please enter your name :
```

Enter the name:

```
Please enter your name : nimal
Connected to Server: 127.0.0.1:9988
nimal :>
```

- **Client Program code:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

```c
#include <pthread.h>

#define LENGTH_NAME 30
#define LENGTH_SEND 201
#define LENGTH_MSG  501

volatile sig_atomic_t flag = 0;
int sockfd = 0;
char nickname[LENGTH_NAME] = {};

void str_overwrite_stdout() {
    printf("\033[0K");
    fflush(stdout);
}

void str_trim_lf(char* arr, int length) {
    int i;
    for (i = 0; i < length; i++) {
        if (arr[i] == '\n') {
            arr[i] = '\0';
            break;
        }
    }
}

void catch_ctrl_c_and_exit(int sig) {
    flag = 1;
}

void recv_msg_handler() {
    char receiveMessage[LENGTH_SEND] = {};
    while (1) {
        int receive = recv(sockfd, receiveMessage, LENGTH_SEND, 0);
        if (receive > 0) {
            printf("_______________________________\n");
            printf("\r :> %s\n", receiveMessage);
            printf("_______________________________\n");
            str_overwrite_stdout();
        } else if (receive == 0) {
            break;
        } else {
```

```
            // Handle -1 case if needed
        }
    }
}

void send_msg_handler() {
    char message[LENGTH_MSG] = {};
    while (1) {
        str_overwrite_stdout();
        while (fgets(message, LENGTH_MSG, stdin) != NULL) {
            str_trim_lf(message, LENGTH_MSG);
            if (strlen(message) == 0) {
                str_overwrite_stdout();
            } else {
                break;
            }
        }
        send(sockfd, message, LENGTH_MSG, 0);
        if (strcmp(message, "exit") == 0) {
            break;
        }
    }
    catch_ctrl_c_and_exit(2);
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <server_ip> <server_port>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    signal(SIGINT, catch_ctrl_c_and_exit);

    // Naming
    printf("Please enter your name : ");
    if (fgets(nickname, LENGTH_NAME, stdin) != NULL) {
        str_trim_lf(nickname, LENGTH_NAME);
    }
    if (strlen(nickname) < 2 || strlen(nickname) >= LENGTH_NAME-1) {
        printf("\nName must be more than one and less than thirty
characters.\n");
```

```
        exit(EXIT_FAILURE);
    }

    // Get server IP address and port number from command-line arguments
    char *server_ip = argv[1];
    int server_port = atoi(argv[2]);

    // Create socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("Fail to create a socket.");
        exit(EXIT_FAILURE);
    }

    // Socket information
    struct sockaddr_in server_info;
    int s_addrlen = sizeof(server_info);
    memset(&server_info, 0, s_addrlen);
    server_info.sin_family = PF_INET;
    server_info.sin_addr.s_addr = inet_addr(server_ip);
    server_info.sin_port = htons(server_port);

    // Connect to Server
    int err = connect(sockfd, (struct sockaddr *)&server_info, s_addrlen);
    if (err == -1) {
        printf("Connection to Server error!\n");
        exit(EXIT_FAILURE);
    }

    printf("Connected to Server: %s:%d\n", inet_ntoa(server_info.sin_addr),
ntohs(server_info.sin_port));
    printf("%s :>",nickname);
    send(sockfd, nickname, LENGTH_NAME, 0);

    pthread_t send_msg_thread;
    if (pthread_create(&send_msg_thread, NULL, (void *)send_msg_handler, NULL) !=
0) {
        printf("Fail to create pthread!\n");
        exit(EXIT_FAILURE);
    }
```

```
    pthread_t recv_msg_thread;
    if (pthread_create(&recv_msg_thread, NULL, (void *)recv_msg_handler, NULL) !=
0) {
        printf("Fail to create pthread!\n");
        exit(EXIT_FAILURE);
    }

    while (1) {
        if (flag) {
            printf("\nBye\n");
            break;
        }
    }

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

Workflow of the chat application

- *All* clients have joined concurrently to the server using the same **IP address (127.0.0.1)** and the same **port number (1212).**

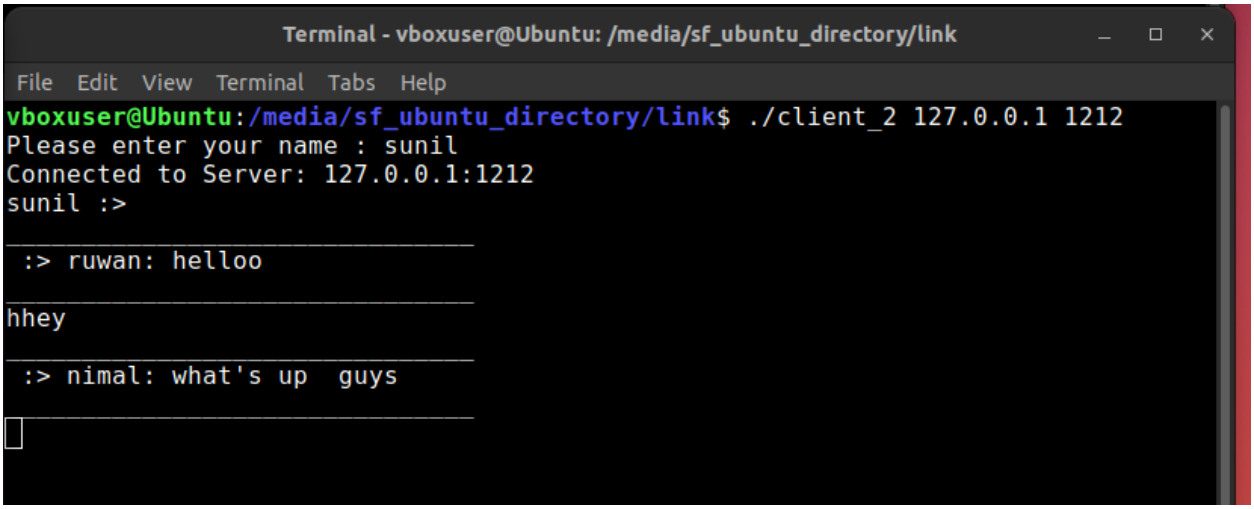
Server:

```
Terminal - vboxuser@Ubuntu: /media/sf_ubuntu_directory/link
File  Edit  View  Terminal  Tabs  Help
vboxuser@Ubuntu:~$ cd /media/sf_ubuntu_directory/link/
vboxuser@Ubuntu:/media/sf_ubuntu_directory/link$ ./server_2 1212
Enter the port number: 1212
=== WELCOME TO THE CHATROOM ===
Client 127.0.0.1:36238 connected.
Client 127.0.0.1:55516 connected.
Client 127.0.0.1:49752 connected.
ruwan : is send massage to other clients:  helloo
sunil : is send massage to other clients:  hhey
nimal : is send massage to other clients:  what's up  guys
```

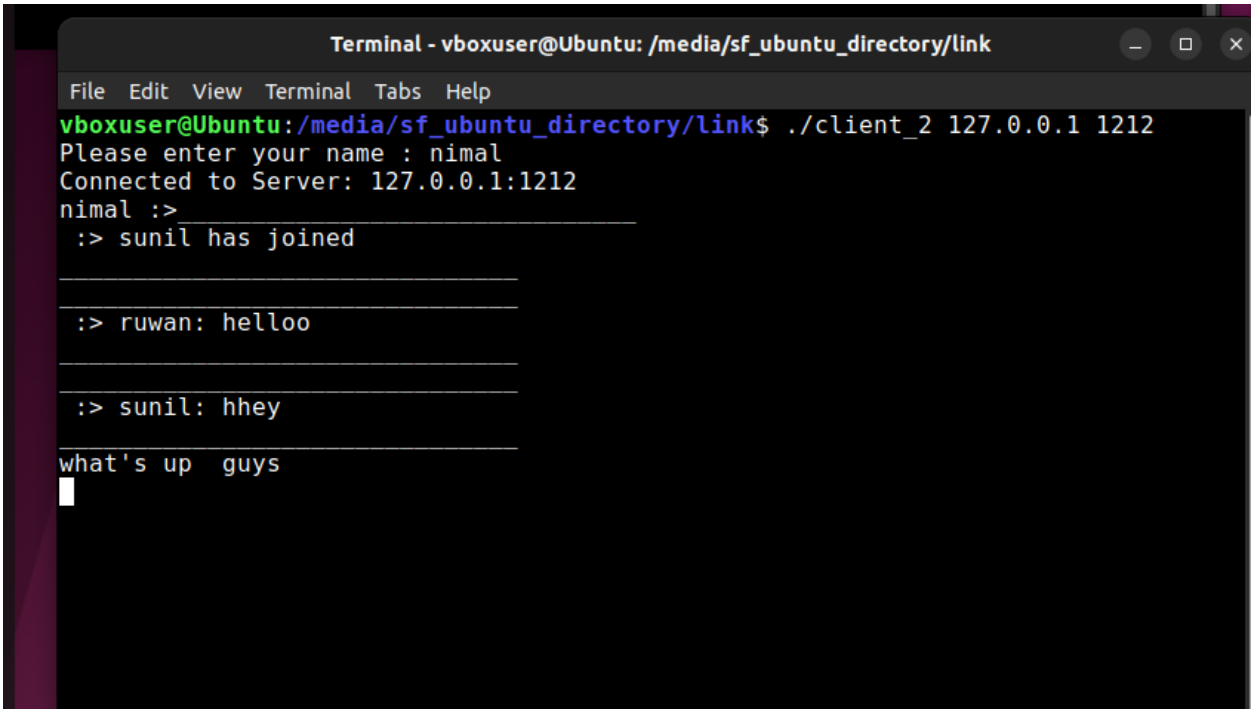
Client 1:

```
Terminal - vboxuser@Ubuntu: /media/sf_ubuntu_directory/link
File  Edit  View  Terminal  Tabs  Help
vboxuser@Ubuntu:/media/sf_ubuntu_directory/link$ ./client_2 127.0.0.1 1212
Please enter your name : ruwan
Connected to Server: 127.0.0.1:1212
ruwan :>_________________________________
 :> nimal has joined
_________________________________
_________________________________
 :> sunil has joined
_________________________________
helloo
_________________________________
 :> sunil: hhey
_________________________________
_________________________________
 :> nimal: what's up  guys
_________________________________
```

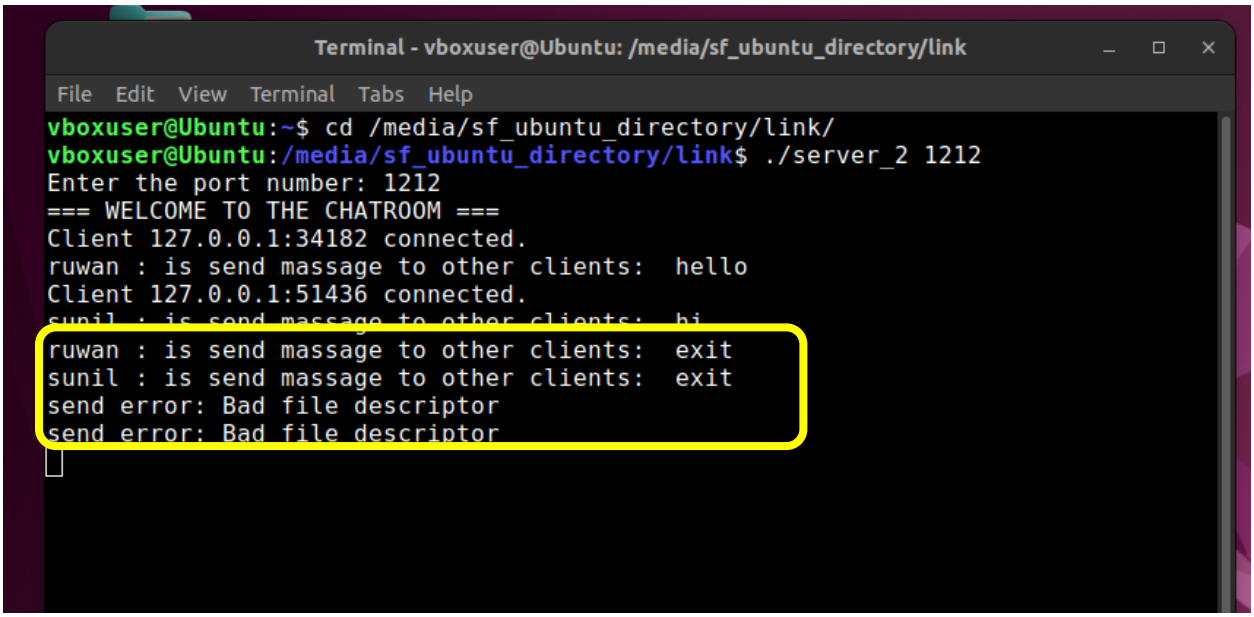
Client 2:

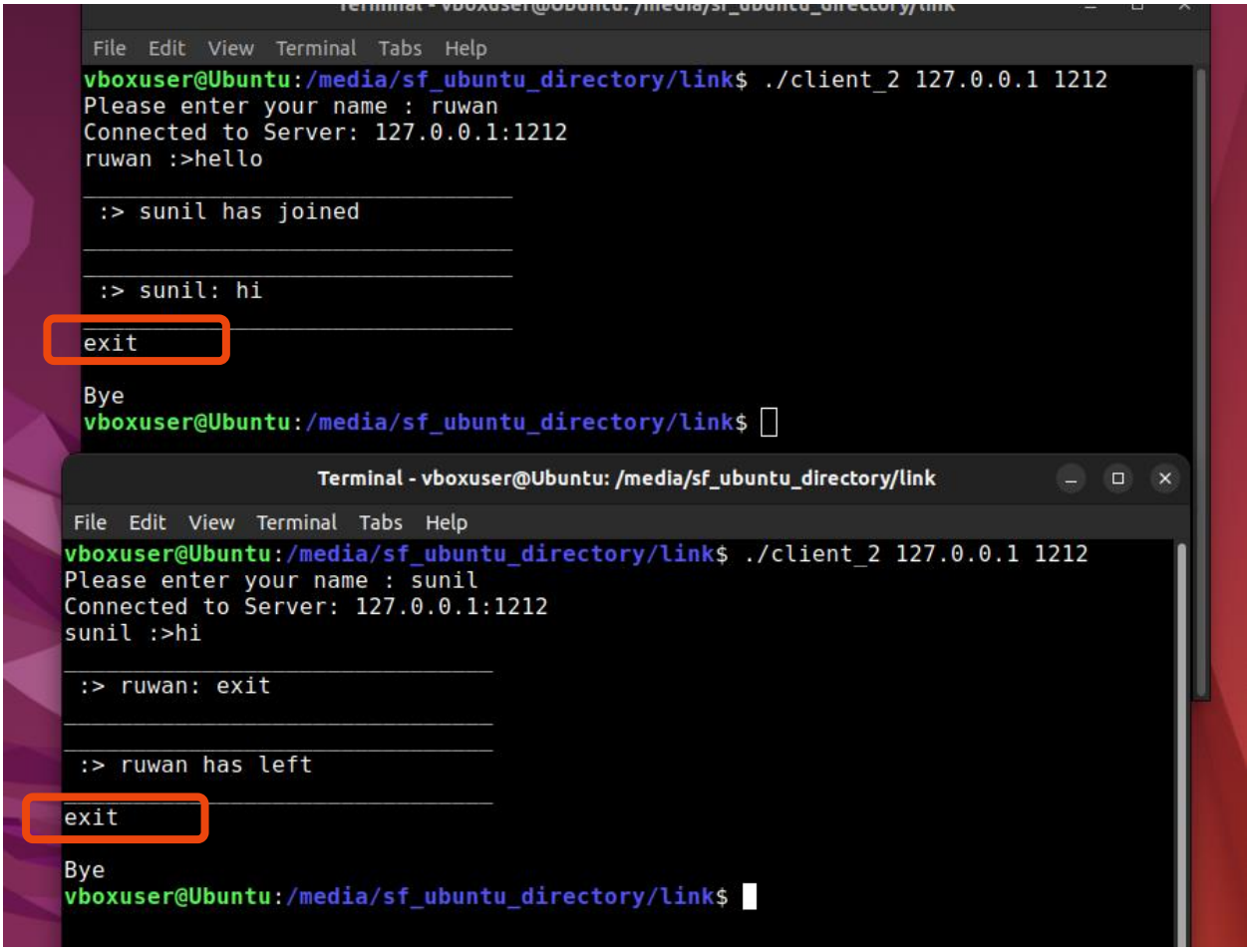


Client 3:



- **Clients exiting the chat.**
- **Using the "exit" command.**




File Edit View Terminal Tabs Help
vboxuser@Ubuntu:/media/sf_ubuntu_directory/link$ ./client_2 127.0.0.1 1212
Please enter your name : ruwan
Connected to Server: 127.0.0.1:1212
ruwan :>hello
:> sunil has joined
:> sunil: hi
exit
Bye
vboxuser@Ubuntu:/media/sf_ubuntu_directory/link$
Terminal - vboxuser@Ubuntu: /media/sf_ubuntu_directory/link
File Edit View Terminal Tabs Help
vboxuser@Ubuntu:/media/sf_ubuntu_directory/link$ ./client_2 127.0.0.1 1212
Please enter your name : sunil
Connected to Server: 127.0.0.1:1212
sunil :>hi
:> ruwan: exit
:> ruwan has left
exit
Bye
vboxuser@Ubuntu:/media/sf_ubuntu_directory/link$

~Hosted Chat Room Server on the Web~

❖ *Quick Guide:*

Deploy Server Code:

- We uploaded the chat room server code to the hosting platform. Before hosting we ensured the server code was compatible with the hosting environment.

Obtain Public IP Address:

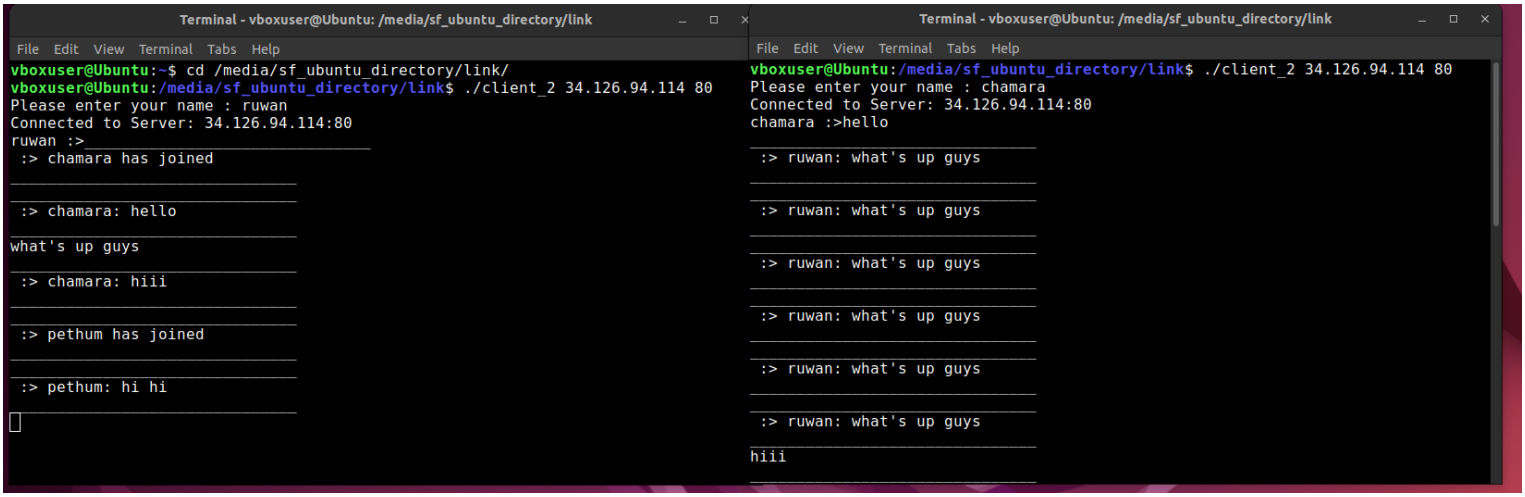
- After deploying your server code, obtain the public IP address assigned to the hosted server. This IP address will serve as the unique identifier for clients to connect to the chat room.

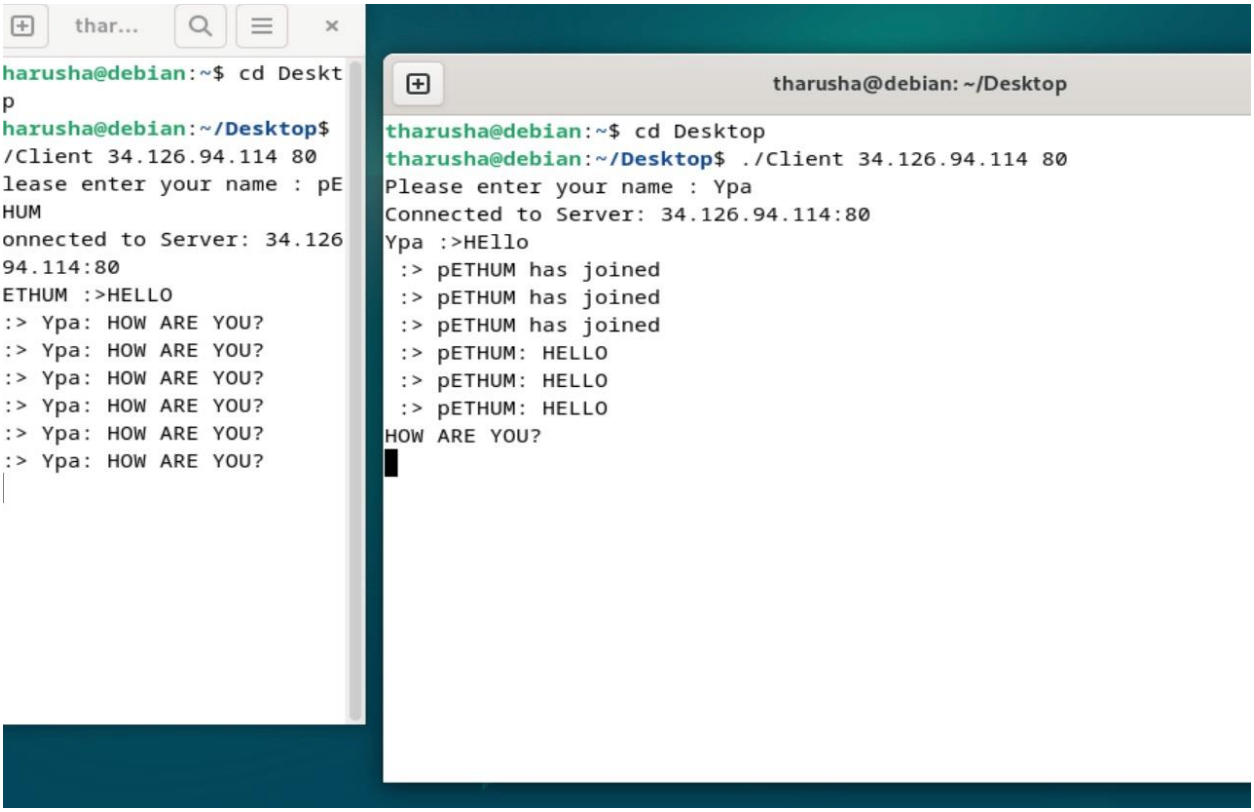
❖ *Documentation:*

- Hosting details to connect to the chat room server.
- But sometimes the Server does not work in the given IP and port some issues or the network traffic (sometimes sever would be shut down for some server issues)
- Server IP :> **34.126.94.114**
- Port: :> **80**

❖ Clients connected to the hosted server:

Assignment - 2
CSCI 21023- Data Communication and Networking
Faculty of Computing & Technology
University of Kelaniya





~ END ~