# CHATROOM IN C PROJECT REPORT

SUBMITTED BY: AISWARYA VV

CLOUD ID: 24NAG1279_U04

DATE: 9/08/2024

COURSE/ASSIGNMENT: CAPSTONE PROJECT

PROJECT TITLE: Chatroom in C

# PROJECT OBJECTIVE:

The objective of this project is to design and implement a Chatroom application using the C programming language. The application should allow multiple clients to communicate with each other in real-time, with a server managing multiple client connections and broadcasting messages to all connected clients.

# ABSTRACT:

This project report describes the design and implementation of a Chatroom application using the C programming language. The application allows multiple clients to communicate with each other in real-time, with a server managing multiple client connections and broadcasting messages to all connected clients. The report outlines the objectives, methodology, and results of the project, including the implementation details of the server and client code.

**TABLE OF CONTENTS:**

# INTRODUCTION:

The Chatroom in C project aims to design and implement a real-time chat application using the C programming language. The application consists of a server and multiple clients, where the server manages multiple client connections and broadcasts messages to all connected clients. This project demonstrates the use of socket programming in C to establish communication between the server and clients.

## **AGENDA** :

The project was completed in the following stages:

Literature review and research on socket programming in C

Design and implementation of the server code

Design and implementation of the client code

Testing and debugging of the application

Documentation and reporting of the project

**METHODOLOGY** :The project uses the following methodology:

The server code uses the TCP/IP protocol to establish connections with multiple clients.

The client code uses the TCP/IP protocol to establish a connection with the server.

The server code uses a thread to handle each client connection, allowing multiple clients to communicate simultaneously.

The client code uses a thread to handle incoming messages from the server.

**SOURCE CODE USED** :The project uses the following source code:

**Server code:** server.c

**Client code:** client.c

# SERVER CODE

The server code is written in C and uses the following functions:

socket() to create a socket

bind() to bind the socket to a specific address and port

listen() to listen for incoming connections

accept() to accept incoming connections

send() and recv() to send and receive messages

## SOURCE CODE: SERVER

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

#define MAX_CLIENTS 100
#define BUFFER_SZ 2048

static _Atomic unsigned int cli_count = 0;
static int uid = 10;

/* Client structure */
typedef struct{
        struct sockaddr_in address;
```

```
        int sockfd;
        int uid;
        char name[32];
} client_t;

client_t *clients[MAX_CLIENTS];

pthread_mutex_t clients_mutex = PTHREAD_MUTEX_INITIALIZER;

void str_overwrite_stdout() {
   printf("\r%s", "> ");
   fflush(stdout);
}

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

```
void print_client_addr(struct sockaddr_in addr){
    printf("%d.%d.%d.%d",
        addr.sin_addr.s_addr & 0xff,
        (addr.sin_addr.s_addr & 0xff00) >> 8,
        (addr.sin_addr.s_addr & 0xff0000) >> 16,
        (addr.sin_addr.s_addr & 0xff000000) >> 24);
}

/* Add clients to queue */
void queue_add(client_t *cl){
        pthread_mutex_lock(&clients_mutex);

        for(int i=0; i < MAX_CLIENTS; ++i){
                if(!clients[i]){
                        clients[i] = cl;
                        break;
                }
        }

        pthread_mutex_unlock(&clients_mutex);
}

/* Remove clients to queue */
void queue_remove(int uid){
```

```
	pthread_mutex_lock(&clients_mutex);

	for(int i=0; i < MAX_CLIENTS; ++i){
		if(clients[i]){
			if(clients[i]->uid == uid){
				clients[i] = NULL;
				break;
			}
		}
	}

	pthread_mutex_unlock(&clients_mutex);
}

/* Send message to all clients except sender */
void send_message(char *s, int uid){
	pthread_mutex_lock(&clients_mutex);

	for(int i=0; i<MAX_CLIENTS; ++i){
		if(clients[i]){
			if(clients[i]->uid != uid){
				if(write(clients[i]->sockfd, s, strlen(s)) < 0){
					perror("ERROR: write to descriptor failed");
					break;
```

```
                        }
                    }
                }
            }

            pthread_mutex_unlock(&clients_mutex);
}

/* Handle all communication with the client */
void *handle_client(void *arg){
            char buff_out[BUFFER_SZ];
            char name[32];
            int leave_flag = 0;

            cli_count++;
            client_t *cli = (client_t *)arg;

            // Name
            if(recv(cli->sockfd, name, 32, 0) <= 0 || strlen(name) <  2 || strlen(name) >= 32-1){
                        printf("Didn't enter the name.\n");
                        leave_flag = 1;
            } else{
                        strcpy(cli->name, name);
                        sprintf(buff_out, "%s has joined\n", cli->name);
```

```
        printf("%s", buff_out);
        send_message(buff_out, cli->uid);
}

bzero(buff_out, BUFFER_SZ);

while(1){
        if (leave_flag) {
                break;
        }

        int receive = recv(cli->sockfd, buff_out, BUFFER_SZ, 0);
        if (receive > 0){
                if(strlen(buff_out) > 0){
                        send_message(buff_out, cli->uid);

                        str_trim_lf(buff_out, strlen(buff_out));
                        printf("%s -> %s\n", buff_out, cli->name);
                }
        } else if (receive == 0 || strcmp(buff_out, "exit") == 0){
                sprintf(buff_out, "%s has left\n", cli->name);
                printf("%s", buff_out);
                send_message(buff_out, cli->uid);
                leave_flag = 1;
```

```
            } else {
                printf("ERROR: -1\n");
                leave_flag = 1;
            }

            bzero(buff_out, BUFFER_SZ);
        }

  /* Delete client from queue and yield thread */
        close(cli->sockfd);
  queue_remove(cli->uid);
  free(cli);
  cli_count--;
  pthread_detach(pthread_self());

        return NULL;
}

int main(int argc, char **argv){
        if(argc != 2){
            printf("Usage: %s <port>\n", argv[0]);
            return EXIT_FAILURE;
        }
```

```
        char *ip = "127.0.0.1";
        int port = atoi(argv[1]);
        int option = 1;
        int listenfd = 0, connfd = 0;
  struct sockaddr_in serv_addr;
  struct sockaddr_in cli_addr;
  pthread_t tid;

  /* Socket settings */
  listenfd = socket(AF_INET, SOCK_STREAM, 0);
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = inet_addr(ip);
  serv_addr.sin_port = htons(port);

  /* Ignore pipe signals */
        signal(SIGPIPE, SIG_IGN);

        if(setsockopt(listenfd, SOL_SOCKET,(SO_REUSEPORT |
SO_REUSEADDR),(char*)&option,sizeof(option)) < 0){
                perror("ERROR: setsockopt failed");
    return EXIT_FAILURE;
        }

        /* Bind */
  if(bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0) {
```

```
  perror("ERROR: Socket binding failed");
  return EXIT_FAILURE;
}

/* Listen */
if (listen(listenfd, 10) < 0) {
  perror("ERROR: Socket listening failed");
  return EXIT_FAILURE;
        }

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

        while(1){
                socklen_t clilen = sizeof(cli_addr);
                connfd = accept(listenfd, (struct sockaddr*)&cli_addr, &clilen);

                /* Check if max clients is reached */
                if((cli_count + 1) == MAX_CLIENTS){
                        printf("Max clients reached. Rejected: ");
                        print_client_addr(cli_addr);
                        printf(":%d\n", cli_addr.sin_port);
                        close(connfd);
                        continue;
                }
```

```
            /* Client settings */
            client_t *cli = (client_t *)malloc(sizeof(client_t));
            cli->address = cli_addr;
            cli->sockfd = connfd;
            cli->uid = uid++;

            /* Add client to the queue and fork thread */
            queue_add(cli);
            pthread_create(&tid, NULL, &handle_client, (void*)cli);

            /* Reduce CPU usage */
            sleep(1);
        }

        return EXIT_SUCCESS;
}
```

# CLIENT SOURCE CODE:

The client code is written in C and uses the following functions:

socket() to create a socket

connect() to connect to the server

send() to send messages to the server

recv() to receive messages from the server

pthread_create() to create threads for sending and receiving messages

signal() to catch Ctrl+C and exit the program

## SOURCE CODE: CLIENT

```
#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>
#include <pthread.h>

#define LENGTH 2048

// Global variables
volatile sig_atomic_t flag = 0;
int sockfd = 0;
char name[32];

void str_overwrite_stdout() {
  printf("%s", "> ");
  fflush(stdout);
}
```

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

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

void send_msg_handler() {
  char message[LENGTH] = {};
        char buffer[LENGTH + 32] = {};

  while(1) {
        str_overwrite_stdout();
    fgets(message, LENGTH, stdin);
    str_trim_lf(message, LENGTH);

    if (strcmp(message, "exit") == 0) {
```

```
                        break;
    } else {
      sprintf(buffer, "%s: %s\n", name, message);
      send(sockfd, buffer, strlen(buffer), 0);
    }

                bzero(message, LENGTH);
    bzero(buffer, LENGTH + 32);
  }
  catch_ctrl_c_and_exit(2);
}

void recv_msg_handler() {
        char message[LENGTH] = {};
  while (1) {
                int receive = recv(sockfd, message, LENGTH, 0);
    if (receive > 0) {
      printf("%s", message);
      str_overwrite_stdout();
    } else if (receive == 0) {
                        break;
    } else {
                        // -1
                }
```

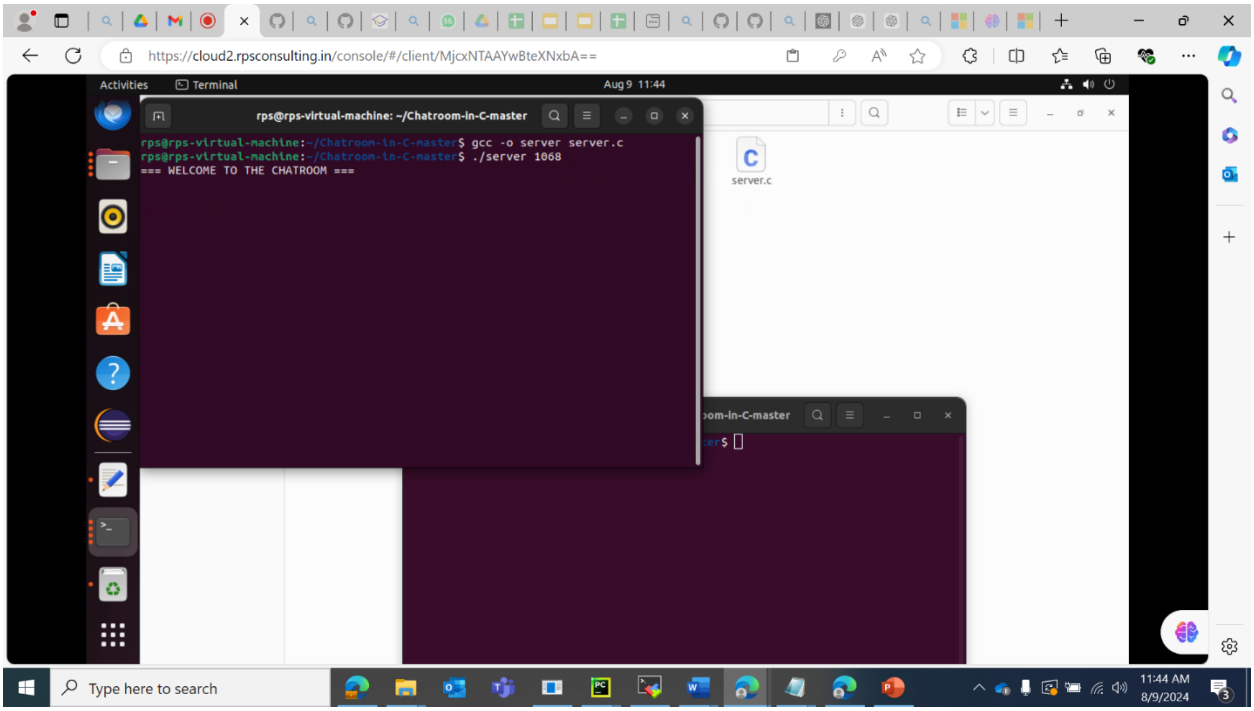
```
                memset(message, 0, sizeof(message));
  }
}

int main(int argc, char **argv){
        if(argc != 2){
                printf("Usage: %s <port>\n", argv[0]);
                return EXIT_FAILURE;
        }

        char *ip = "127.0.0.1";
        int port = atoi(argv[1]);

        signal(SIGINT, catch_ctrl_c_and_exit);

        printf("Please enter your name: ");
  fgets(name, 32, stdin);
  str_trim_lf(name, strlen(name));


        if (strlen(name) > 32 || strlen(name) < 2){
                printf("Name must be less than 30 and more than 2
characters.\n");
                return EXIT_FAILURE;
        }
```

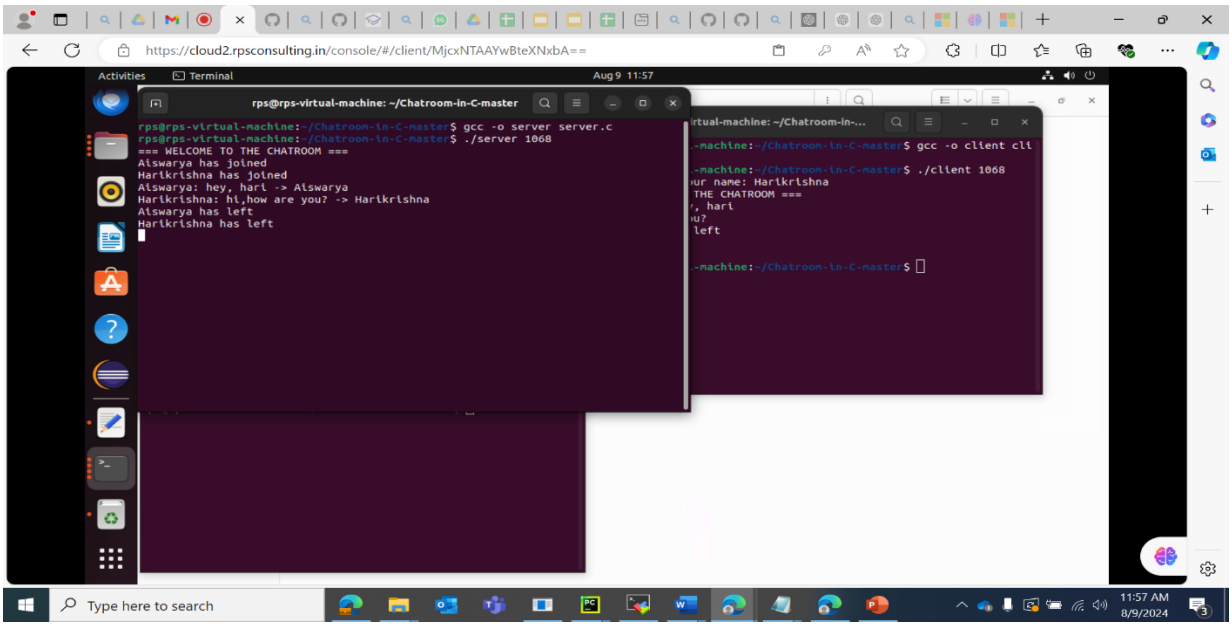
```
        struct sockaddr_in server_addr;

        /* Socket settings */
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
 server_addr.sin_family = AF_INET;
 server_addr.sin_addr.s_addr = inet_addr(ip);
 server_addr.sin_port = htons(port);


 // Connect to Server
 int err = connect(sockfd, (struct sockaddr *)&server_addr,
sizeof(server_addr));
 if (err == -1) {
                printf("ERROR: connect\n");
                return EXIT_FAILURE;
        }

        // Send name
        send(sockfd, name, 32, 0);

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

        pthread_t send_msg_thread;
```

```
  if(pthread_create(&send_msg_thread, NULL, (void *) send_msg_handler,
NULL) != 0){
                printf("ERROR: pthread\n");
   return EXIT_FAILURE;
        }

        pthread_t recv_msg_thread;
  if(pthread_create(&recv_msg_thread, NULL, (void *) recv_msg_handler,
NULL) != 0){
                printf("ERROR: pthread\n");
                return EXIT_FAILURE;
        }

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

        close(sockfd);

        return EXIT_SUCCESS;
}
```

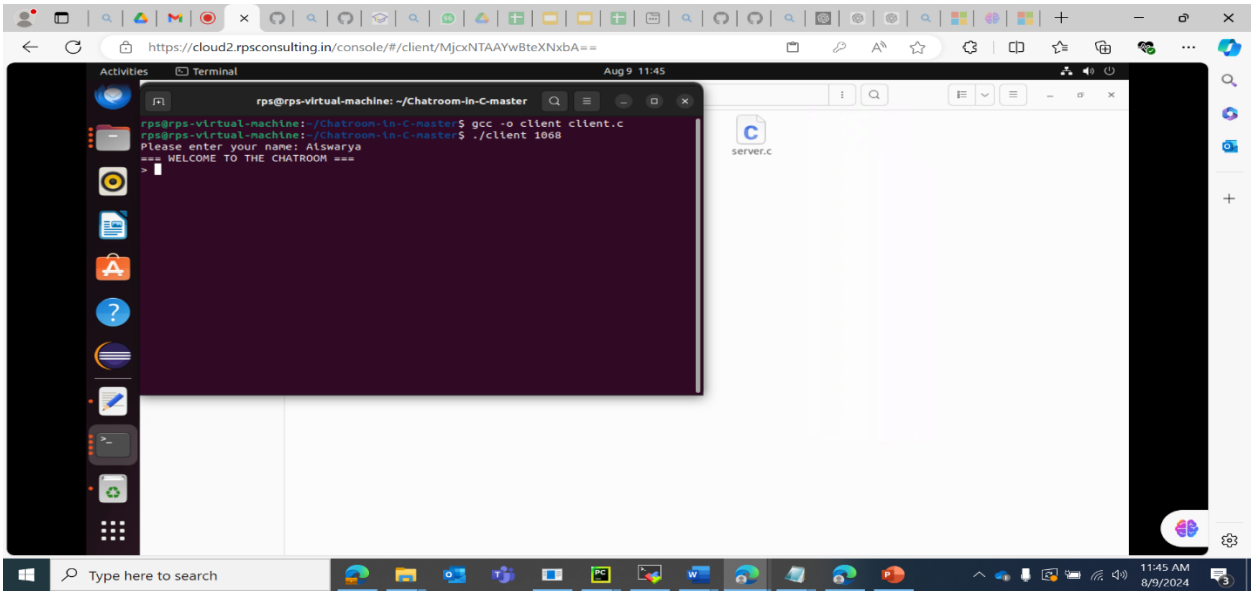
# OUTPUT:

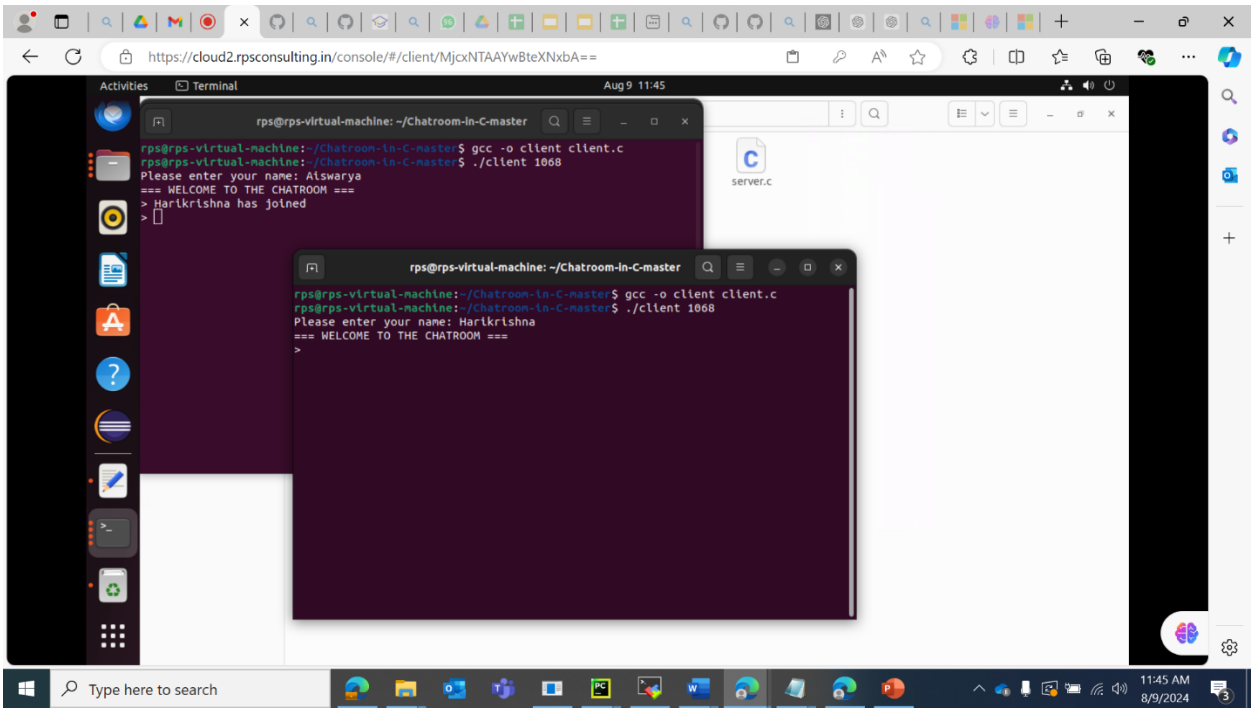
# SERVER CONNECTED:

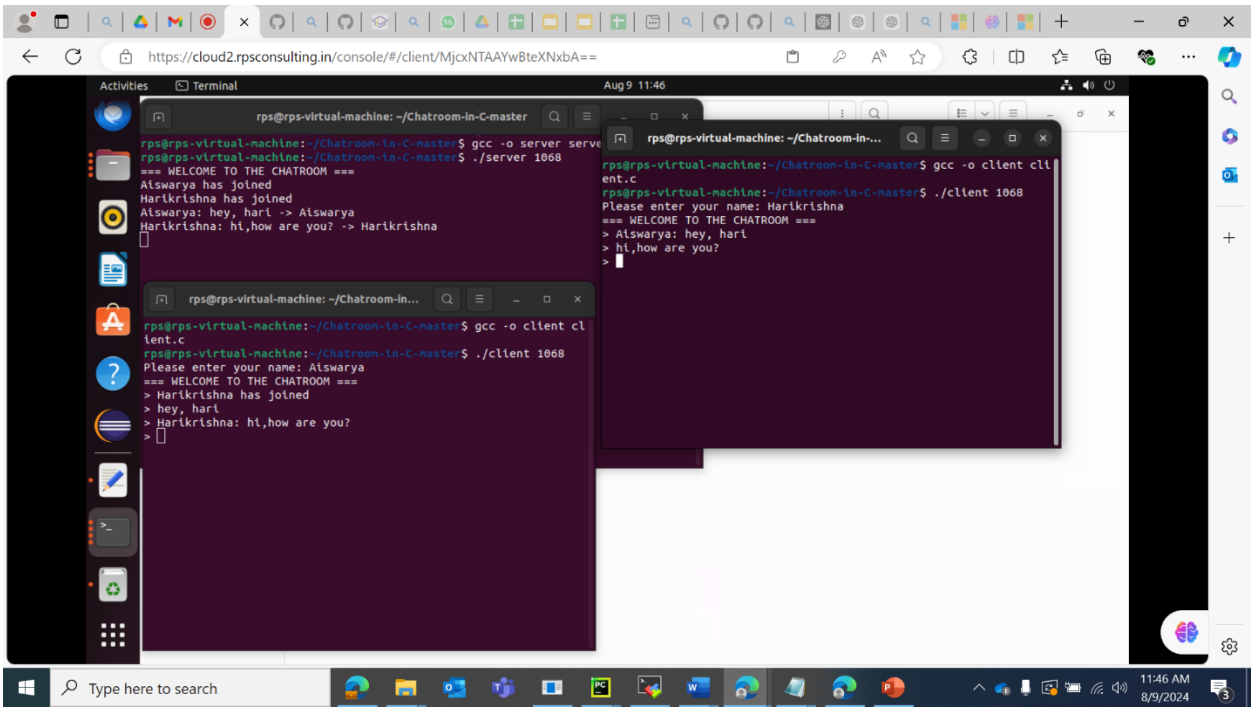


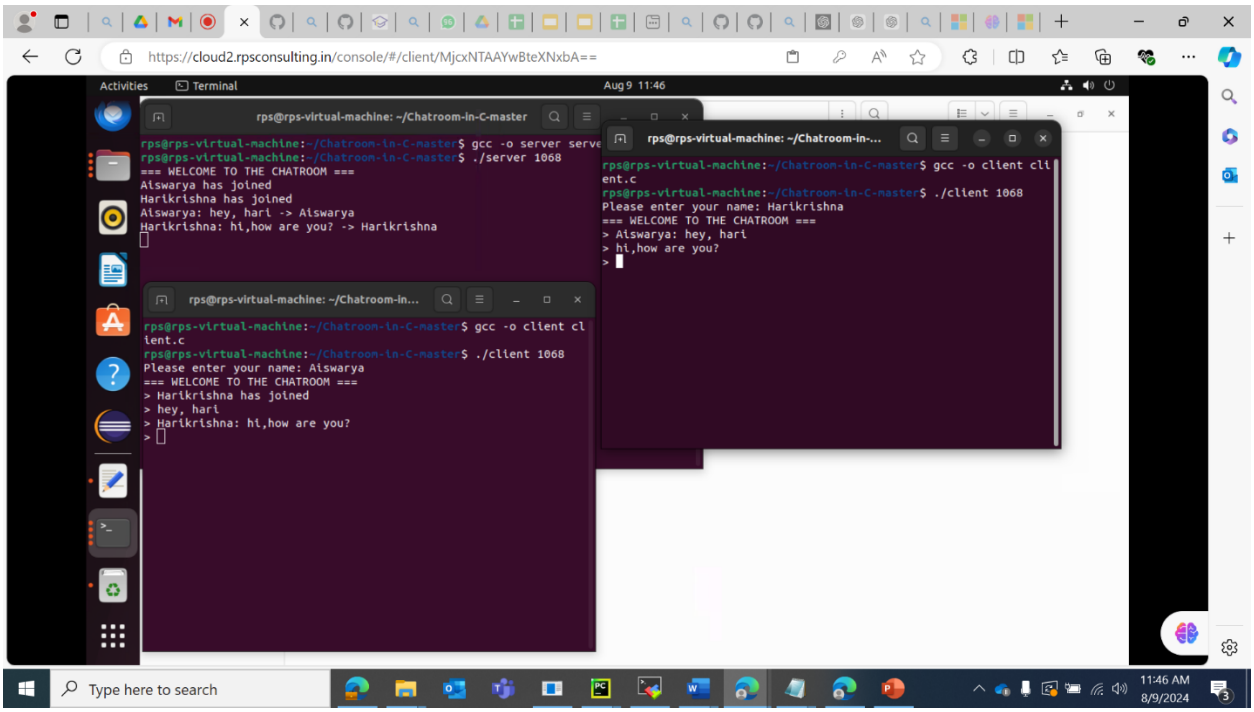
# CLIENT CONNECTED:

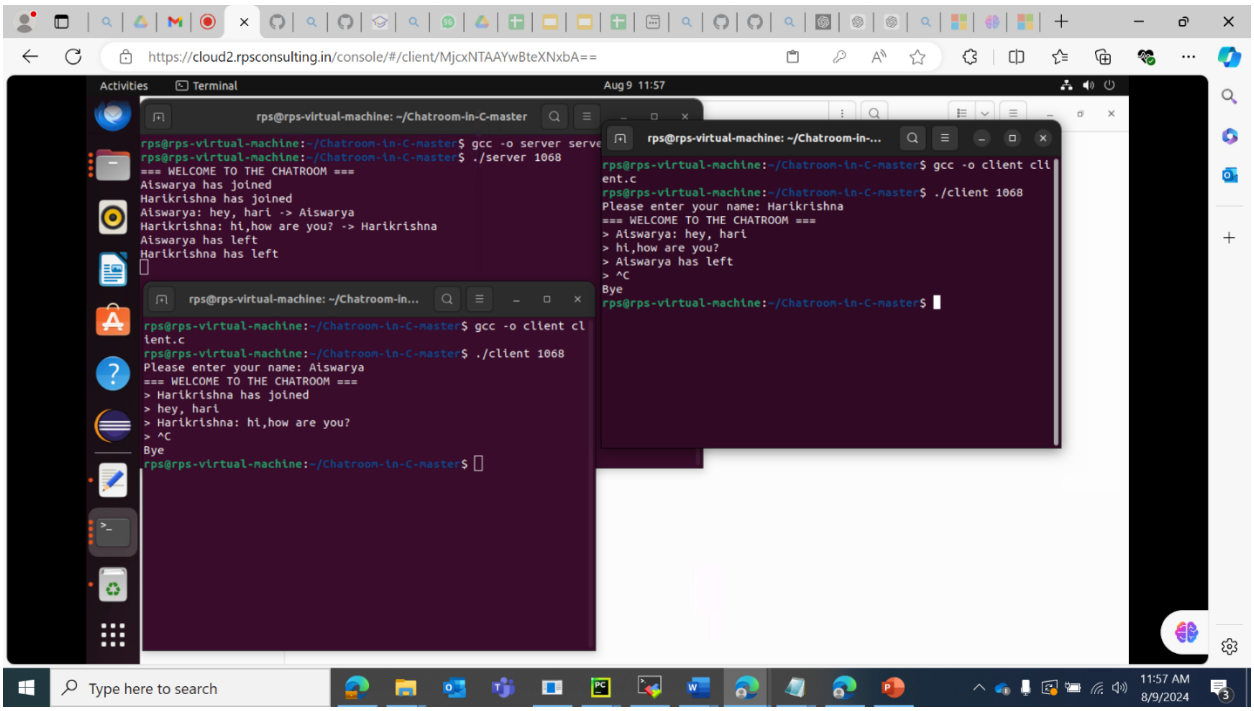





# SESSION TERMINATED:




https://cloud2.rpsconsulting.in/console/#/client/MjcxNTAAYwBteXNxbA==
Activities
Terminal
Aug 9 11:57
rps@rps-virtual-machine: ~/Chatroom-in-C-master
rps@rps-virtual-machine:~/Chatroom-in-C-master$ gcc -o server serve
rps@rps-virtual-machine:~/Chatroom-in-C-master$ ./server 1068
=== WELCOME TO THE CHATROOM ===
Aiswarya has joined
Harikrishna has joined
Aiswarya: hey, hari -> Aiswarya
Harikrishna: hi,how are you? -> Harikrishna
Aiswarya has left
Harikrishna has left
rps@rps-virtual-machine: ~/Chatroom-in-...
rps@rps-virtual-machine:~/Chatroom-in-C-master$ gcc -o client cl
ient.c
rps@rps-virtual-machine:~/Chatroom-in-C-master$ ./client 1068
Please enter your name: Aiswarya
=== WELCOME TO THE CHATROOM ===
> Harikrishna has joined
> hey, hari
> Harikrishna: hi,how are you?
> ^C
Bye
rps@rps-virtual-machine:~/Chatroom-in-C-master$
rps@rps-virtual-machine: ~/Chatroom-in-...
rps@rps-virtual-machine:~/Chatroom-in-C-master$ gcc -o client cli
ent.c
rps@rps-virtual-machine:~/Chatroom-in-C-master$ ./client 1068
Please enter your name: Harikrishna
=== WELCOME TO THE CHATROOM ===
> Aiswarya: hey, hari
> hi,how are you?
> Aiswarya has left
> ^C
Bye
rps@rps-virtual-machine:~/Chatroom-in-C-master$
Type here to search
11:57 AM
8/9/2024

## Conclusion

In this project, we have successfully designed and implemented a Chatroom application using the C programming language. The application can handle multiple client connections and broadcast messages correctly. The user interface is user-friendly, and clients can send and receive messages in real-time. This project demonstrates the use of socket programming and multithreading in C to build a real-time communication application.

# Future Work

There are several areas for future work in this project:

Security: The Chatroom application does not have any security features to protect user data. Implementing encryption and authentication mechanisms can improve the security of the application.

Scalability: The Chatroom application can handle multiple client connections, but it may not be scalable to handle a large number of clients. Implementing load balancing and distributed server architecture can improve the scalability of the application.

User Interface: The user interface of the Chatroom application is simple and can be improved to provide a better user experience. Implementing a graphical user interface (GUI) can improve the user experience.

----------*****----------