****

**OPERATING SYSTEM PROJECT REPORT**

**GROUP CHAT SYSTEM USING CLIENT-SERVER COMMUNICATION**

**Instructor: Miss Razia Qamar**

**Group Members**

**Marium Haris (15084)**

**M.Hammad Shaikh (14983)**

**Class ID: 114727**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **SNO** | **CONTENTS** | **PAGE NO** |
| 1 | Acknowledgments | 2 |
| 2 | Introduction | 3 |
| 3 | System requirements | 3 |
| 4 | System design | 3 |
| 5 | Implementation | 4 |
| 6 | Key Features | 4 |
| 7 | Project Components | 4 |
| 8 | Conclusion | 15 |

**ACKNOWLEDGMENTS**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

We would like to express our deepest gratitude to our project supervisor, Miss Razia Qamar, for her invaluable guidance, support, and encouragement throughout the development of this project. Her insights and expertise have been instrumental in the successful completion of this **group chat system using client-server communication.** We appreciate her dedication and the time she invested in reviewing our work and providing constructive feedback.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**INTRODUCTION**

The Client-Server Communication for Group Chat project is designed to facilitate a group chat application where multiple clients can send messages to a server, which then displays these messages with user identifiers. This project is implemented using the C programming language on an Ubuntu operating system, with the environment set up using Oracle VirtualBox. The project consists of a server and five clients.

**SOFTWARE REQUIREMENTS**

**Ubuntu OS:** The operating system used for development and deployment.

**Oracle VirtualBox:** Virtualization software for running Ubuntu instances.

**GCC:** GNU Compiler Collection for compiling C programs.

**POSIX Sockets:** For implementing network communication.

**SYSTEM DESIGN**

**Architecture**

The system follows a client-server architecture, where the server manages communication between multiple clients. Each client sends messages to the server, which then relays these messages to all other connected clients.

**Components**

**Server:** Manages client connections and relays messages.

**Clients:** Connect to the server and send/receive messages.

**Communication Protocol**

The communication between clients and the server is handled using TCP/IP sockets. The server listens on a specific port for incoming client connections. Each message from a client is prefixed with a user identifier (e.g., "User A") before being broadcast to all clients.

**IMPLEMENTATION**

**Technologies Used**

**C Language:** For implementing the client and server programs.

**POSIX Sockets:** For network communication.

**Ubuntu OS:** Development and deployment environment.

**Oracle VirtualBox:** To host Ubuntu instances.

**KEY FEATURES**

**Multi-Client Support:** The server can handle multiple client connections simultaneously.

**Message Broadcasting:** Messages sent by one client are broadcast to all other clients.

**User Identification:** Messages are prefixed with a user identifier for clarity.

**PROJECT COMPONENTS**

**SERVER**

The server component is responsible for receiving messages from all connected clients and displaying them in a structured format.

**Implementation Details:**

- The server creates FIFO (First In First Out) named pipes for communication with each client.

- It listens for incoming messages from clients using the select system call to handle multiple file descriptors simultaneously.

- Upon receiving a message from a client, it reads from the corresponding FIFO and displays the message along with the sender's identifier.

**CODE:**

**#include <stdio.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <fcntl.h>**

**#include <sys/stat.h>**

**#include <sys/types.h>**

**#include <sys/select.h>**

**#define FIFO\_FILE\_1 "MYFIFO1"**

**#define FIFO\_FILE\_2 "MYFIFO2"**

**#define FIFO\_FILE\_3 "MYFIFO3"**

**#define FIFO\_FILE\_4 "MYFIFO4"**

**#define FIFO\_FILE\_5 "MYFIFO5"**

**int main() {**

**int fd1, fd2, fd3, fd4, fd5;**

**char readbuf1[80], readbuf2[80], readbuf3[80], readbuf4[80], readbuf5[80];**

**fd\_set read\_fds;**

**int max\_fd;**

**mkfifo(FIFO\_FILE\_1, 0666);**

**mkfifo(FIFO\_FILE\_2, 0666);**

**mkfifo(FIFO\_FILE\_3, 0666);**

**mkfifo(FIFO\_FILE\_4, 0666);**

**mkfifo(FIFO\_FILE\_5, 0666);**

**while (1) {**

**FD\_ZERO(&read\_fds);**

**max\_fd = -1;**

**fd1 = open(FIFO\_FILE\_1, O\_RDONLY | O\_NONBLOCK);**

**if (fd1 != -1) {**

**FD\_SET(fd1, &read\_fds);**

**max\_fd = (fd1 > max\_fd) ? fd1 : max\_fd;**

**}**

**fd2 = open(FIFO\_FILE\_2, O\_RDONLY | O\_NONBLOCK);**

**if (fd2 != -1) {**

**FD\_SET(fd2, &read\_fds);**

**max\_fd = (fd2 > max\_fd) ? fd2 : max\_fd;**

**}**

**fd3 = open(FIFO\_FILE\_3, O\_RDONLY | O\_NONBLOCK);**

**if (fd3 != -1) {**

**FD\_SET(fd3, &read\_fds);**

**max\_fd = (fd3 > max\_fd) ? fd3 : max\_fd;**

**}**

**fd4 = open(FIFO\_FILE\_4, O\_RDONLY | O\_NONBLOCK);**

**if (fd4 != -1) {**

**FD\_SET(fd4, &read\_fds);**

**max\_fd = (fd4 > max\_fd) ? fd4 : max\_fd;**

**}**

**fd5 = open(FIFO\_FILE\_5, O\_RDONLY | O\_NONBLOCK);**

**if (fd5 != -1) {**

**FD\_SET(fd5, &read\_fds);**

**max\_fd = (fd5 > max\_fd) ? fd5 : max\_fd;**

**}**

**if (max\_fd == -1) {**

**sleep(1);**

**continue;**

**}**

**if (select(max\_fd + 1, &read\_fds, NULL, NULL, NULL) == -1) {**

**perror("select");**

**return 1;**

**}**

**if (FD\_ISSET(fd1, &read\_fds)) {**

**read(fd1, readbuf1, sizeof(readbuf1));**

**printf("User A says: %s\n", readbuf1);**

**close(fd1);**

**}**

**if (FD\_ISSET(fd2, &read\_fds)) {**

**read(fd2, readbuf2, sizeof(readbuf2));**

**printf("User B says: %s\n", readbuf2);**

**close(fd2);**

**}**

**if (FD\_ISSET(fd3, &read\_fds)) {**

**read(fd3, readbuf3, sizeof(readbuf3));**

**printf("User C says: %s\n", readbuf3);**

**close(fd3);**

**}**

**if (FD\_ISSET(fd4, &read\_fds)) {**

**read(fd4, readbuf4, sizeof(readbuf4));**

**printf("User D says: %s\n", readbuf4);**

**close(fd4);**

**}**

**if (FD\_ISSET(fd5, &read\_fds)) {**

**read(fd5, readbuf5, sizeof(readbuf5));**

**printf("User E says: %s\n", readbuf5);**

**close(fd5);**

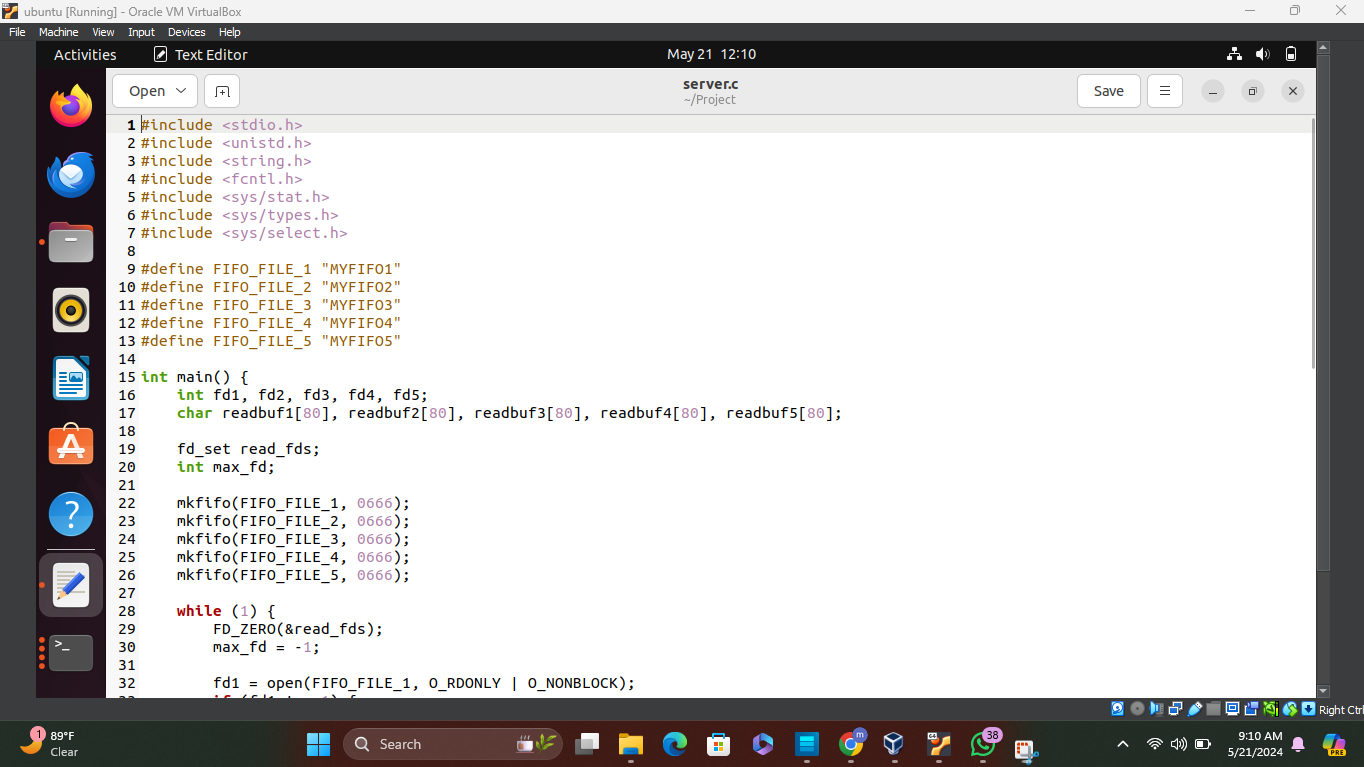
**}**

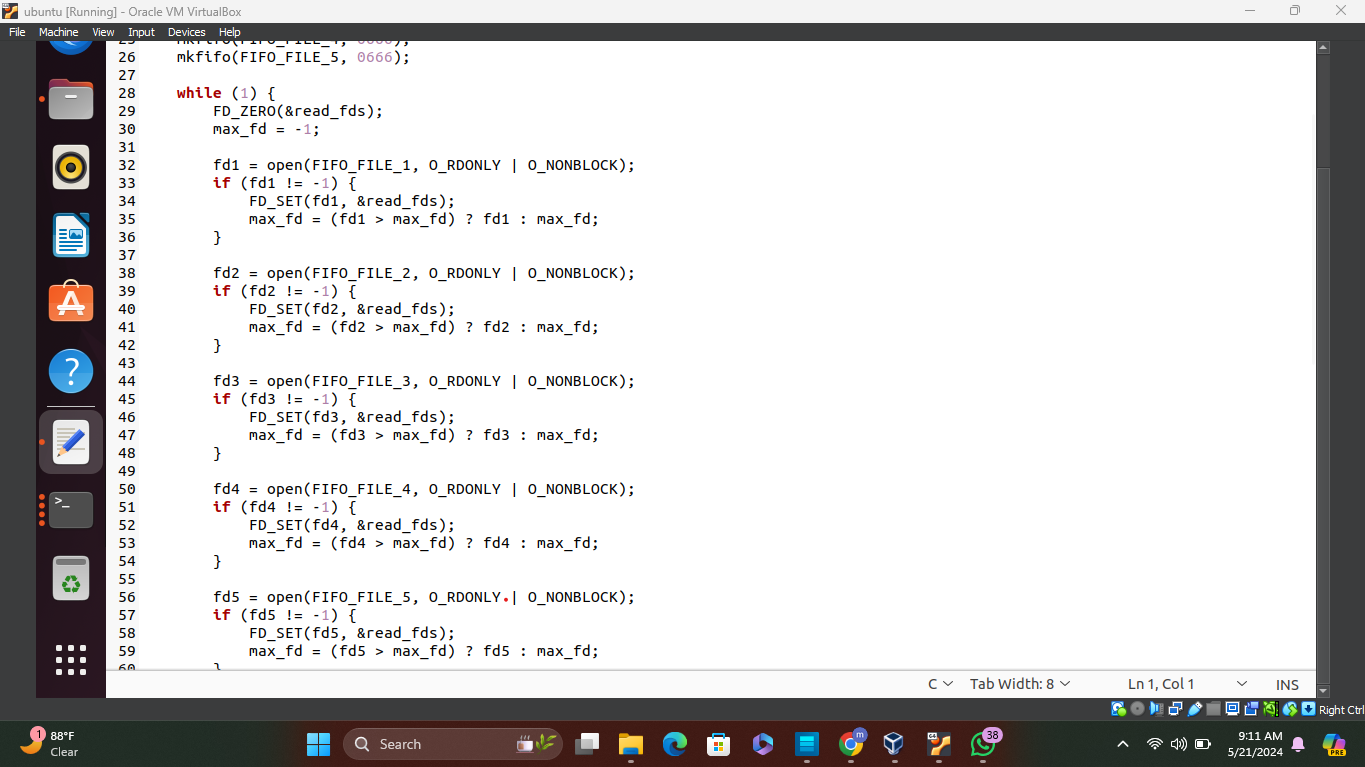
**}**

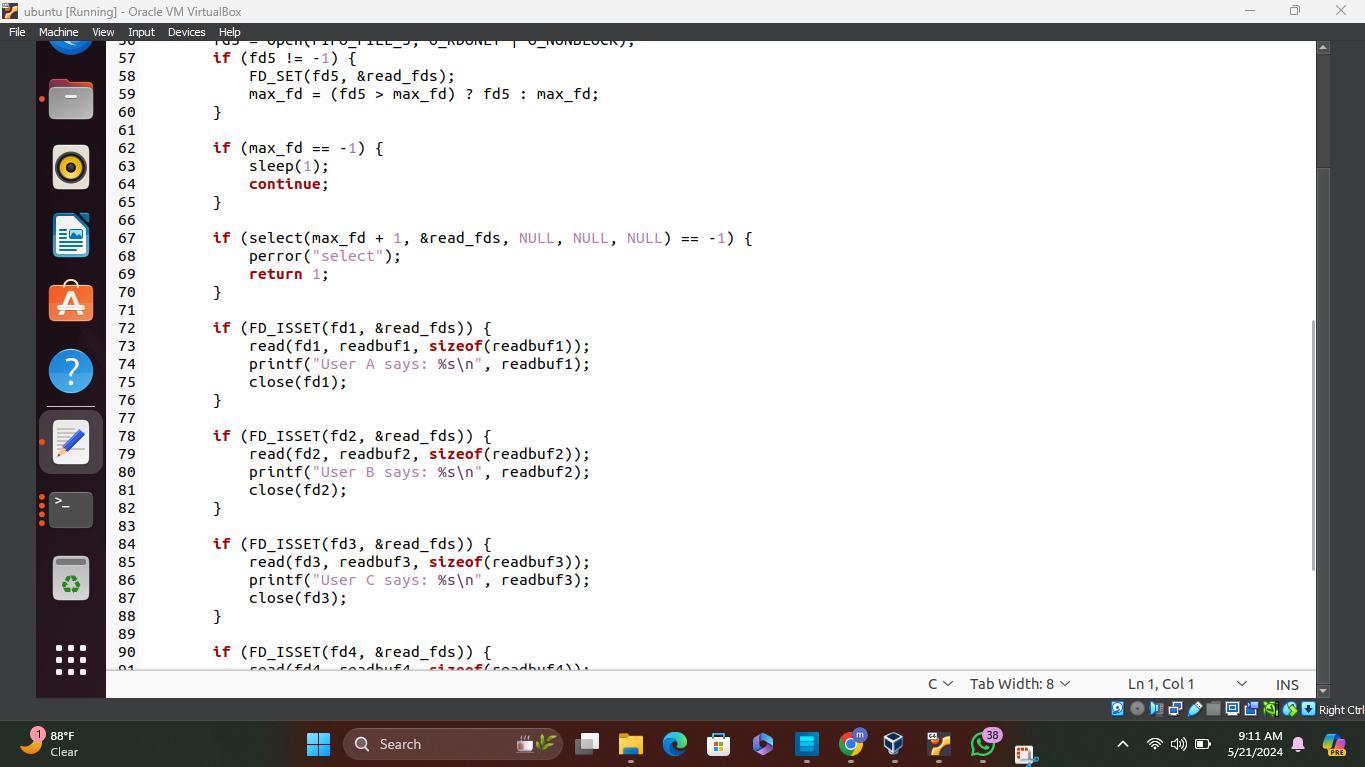
**return 0;**

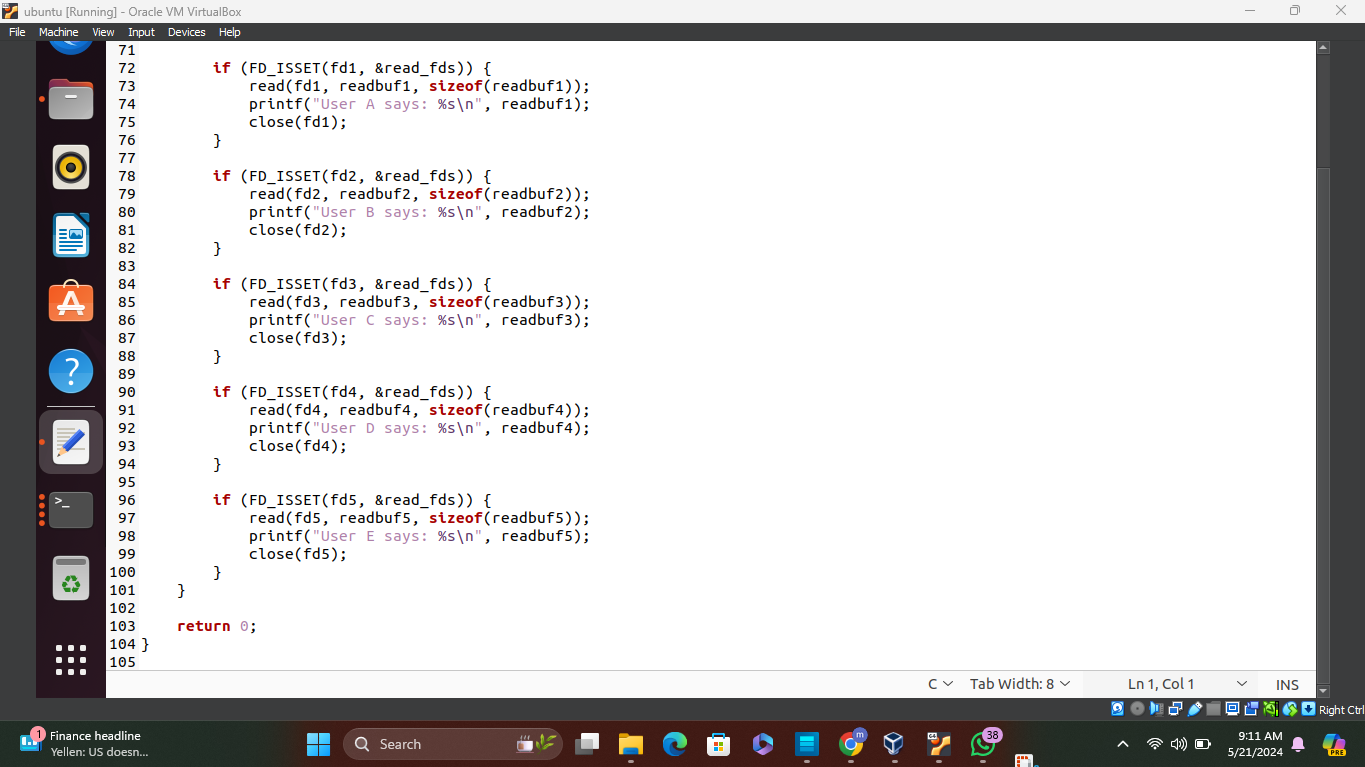
**}**

**SCREENSHOTS**

****

****

****

****

**CLIENTS**

Clients are responsible for sending messages to the server. Each client connects to the server via a named pipe and sends messages entered by the user.

**Implementation Details:**

- Each client creates a FIFO named pipe with a unique identifier.

- It prompts the user to enter a message, reads the input, and sends it to the server through the FIFO.

**CODE:**

**#include <stdio.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <fcntl.h>**

**#include <sys/stat.h>**

**#include <sys/types.h>**

**#define MAX\_BUF 80**

**int main() {**

**int fd;**

**char fifo\_name[20];**

**char readbuf[MAX\_BUF];**

**sprintf(fifo\_name, "MYFIFO%d", (CLIENTID));**

**mkfifo(fifo\_name, 0666);**

**printf("You can type your message:\n");**

**while(1) {**

**fd = open(fifo\_name, O\_WRONLY);**

**fgets(readbuf, MAX\_BUF, stdin);**

**write(fd, readbuf, strlen(readbuf)+1);**

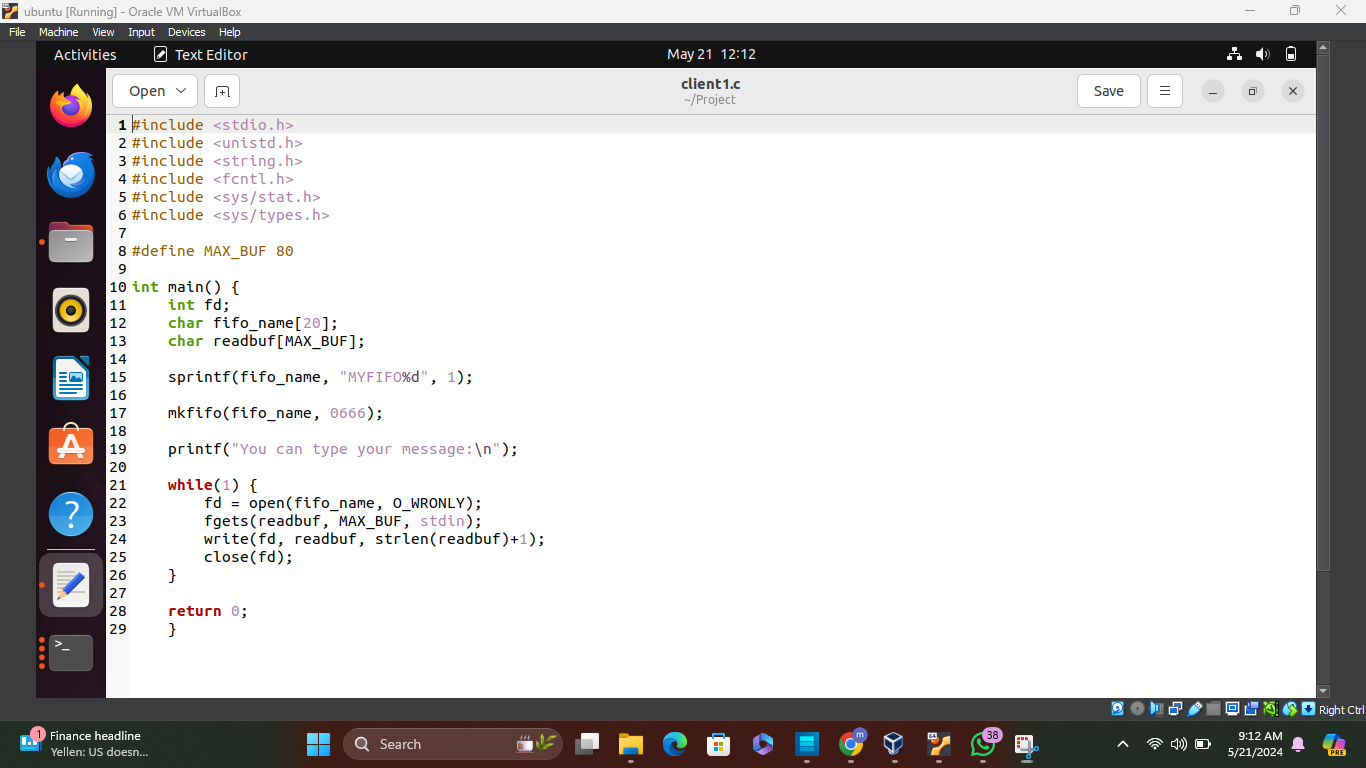
**close(fd);**

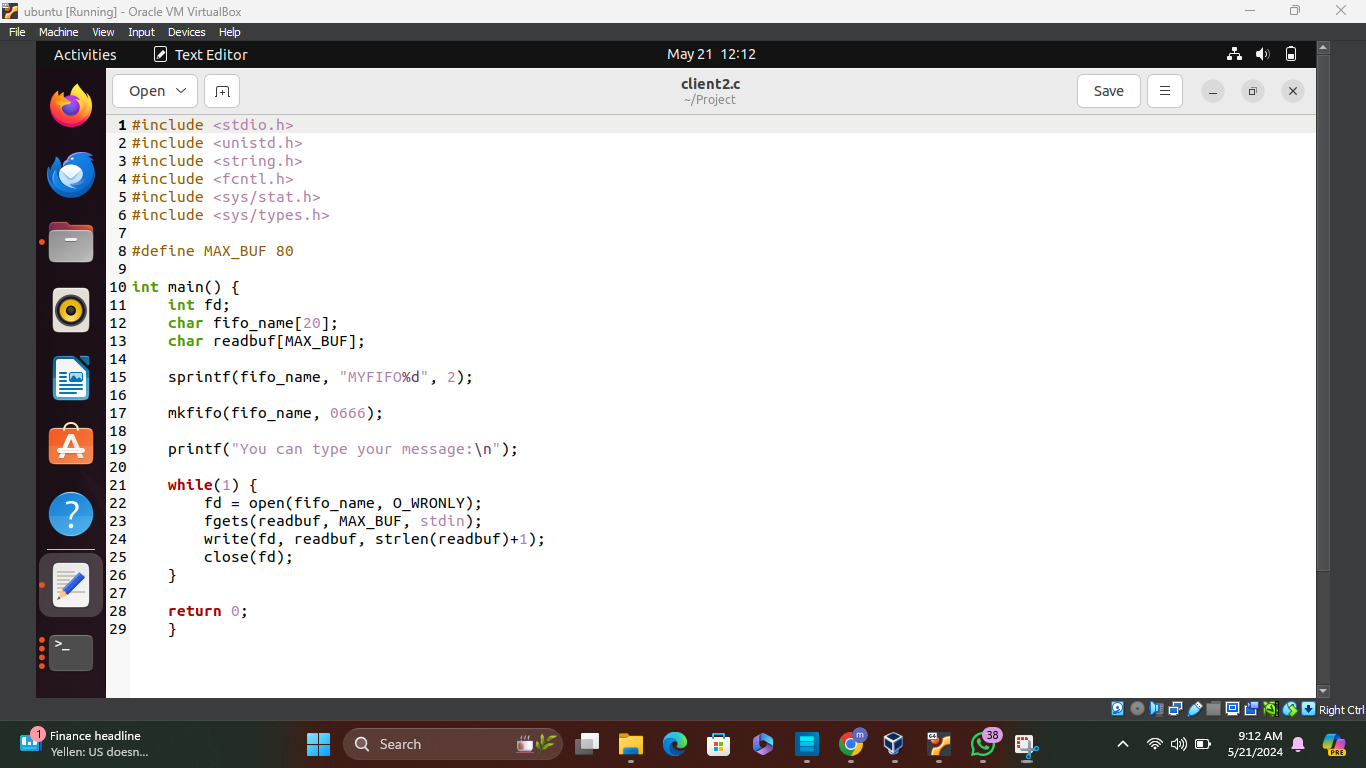
**}**

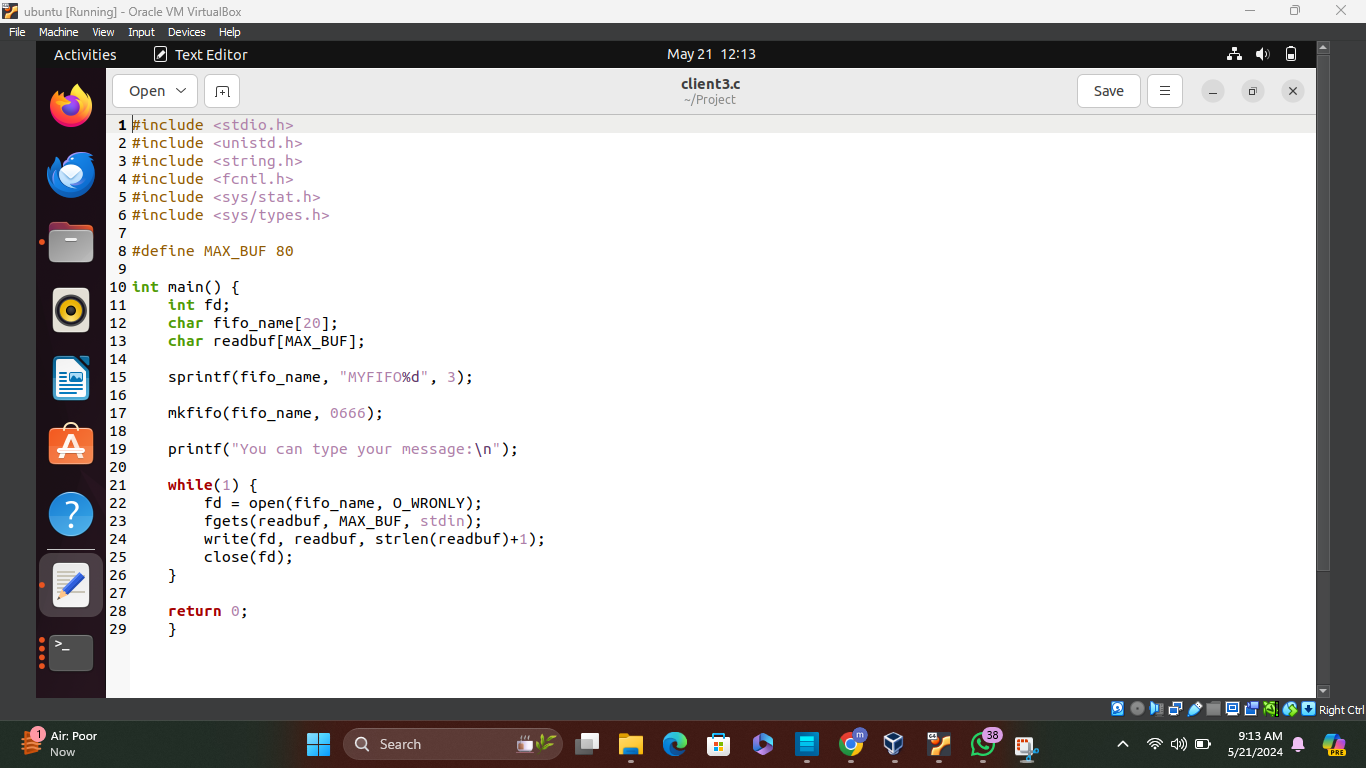
**return 0;**

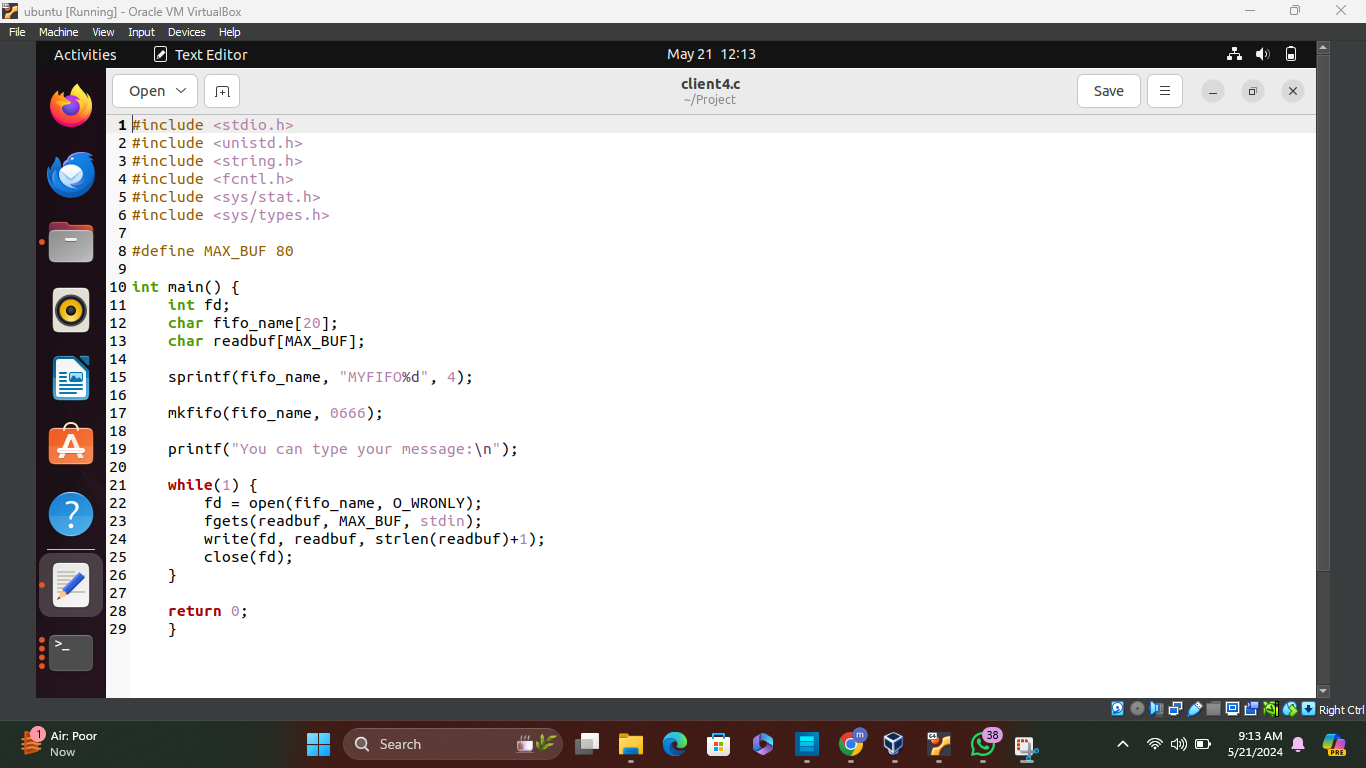
**}**

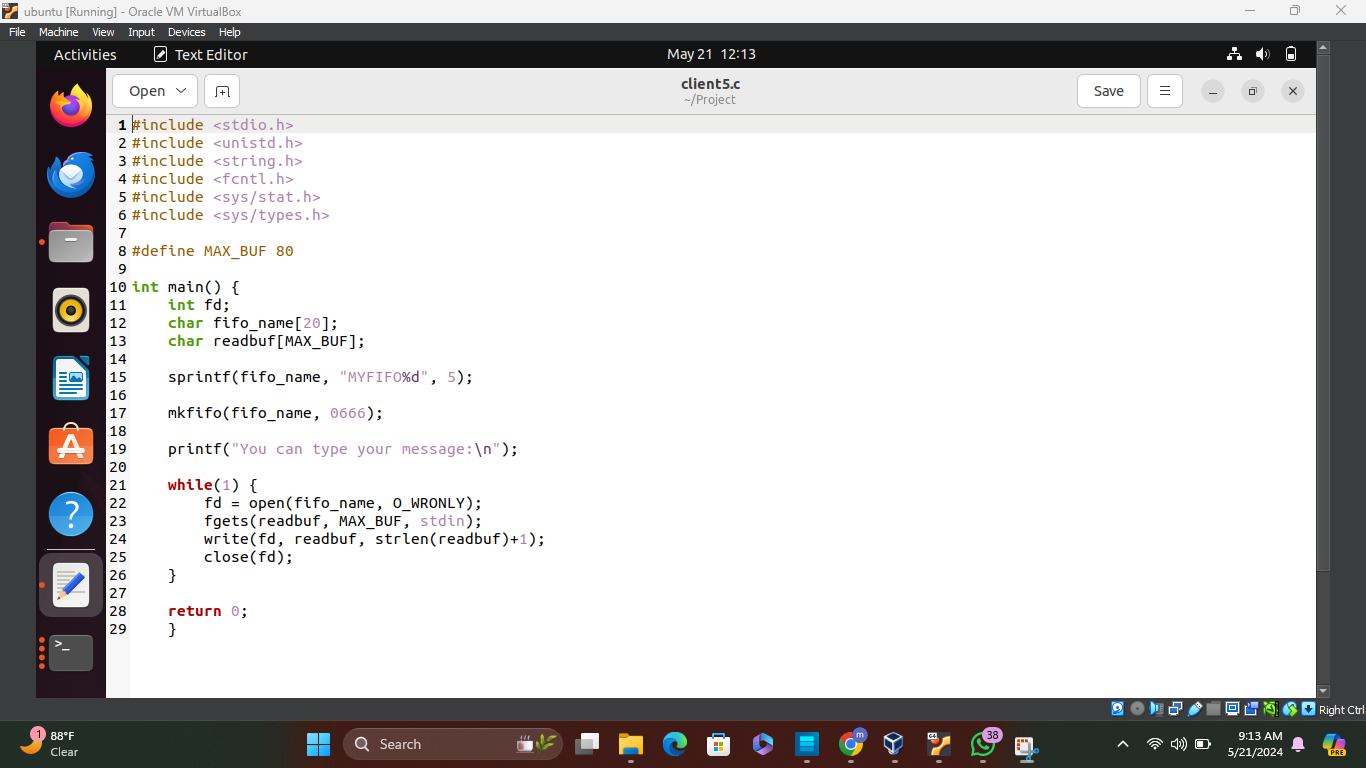
**SCREENSHOTS**

****

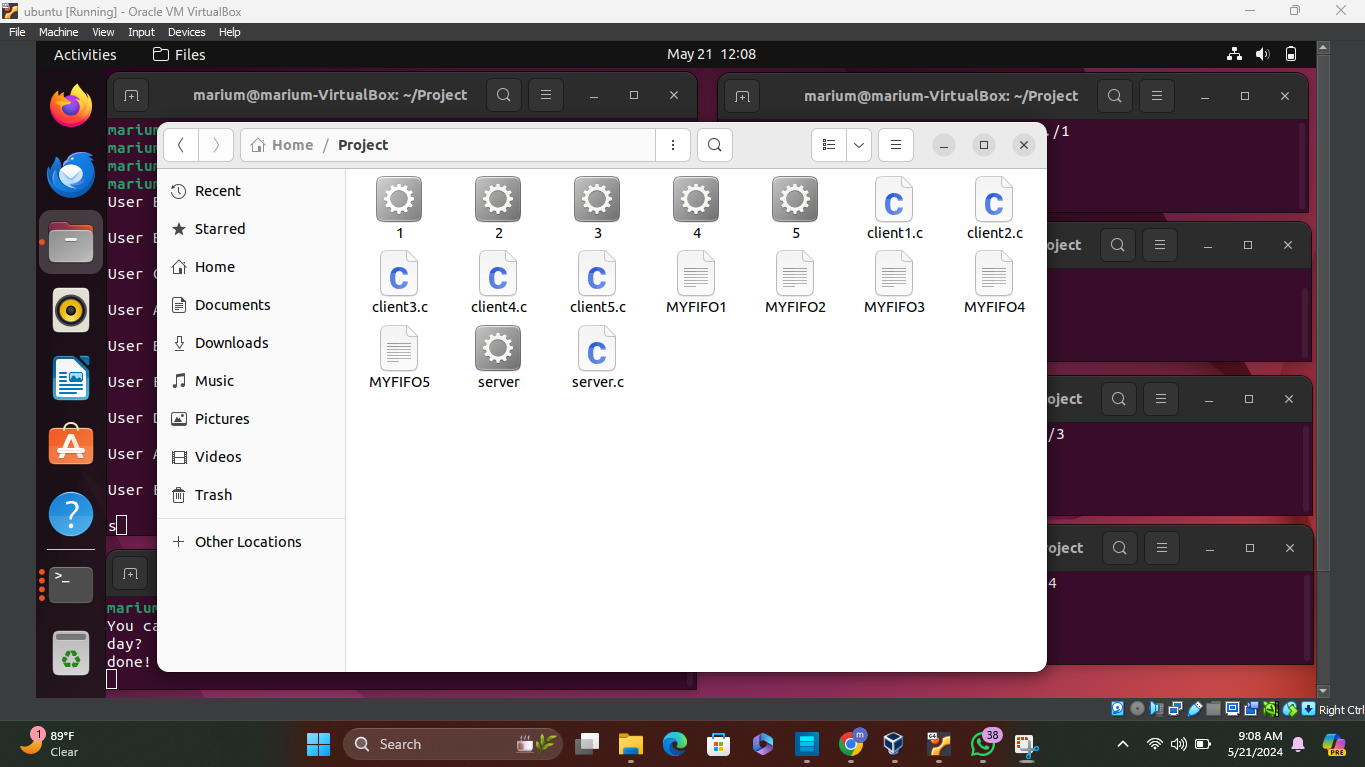
****

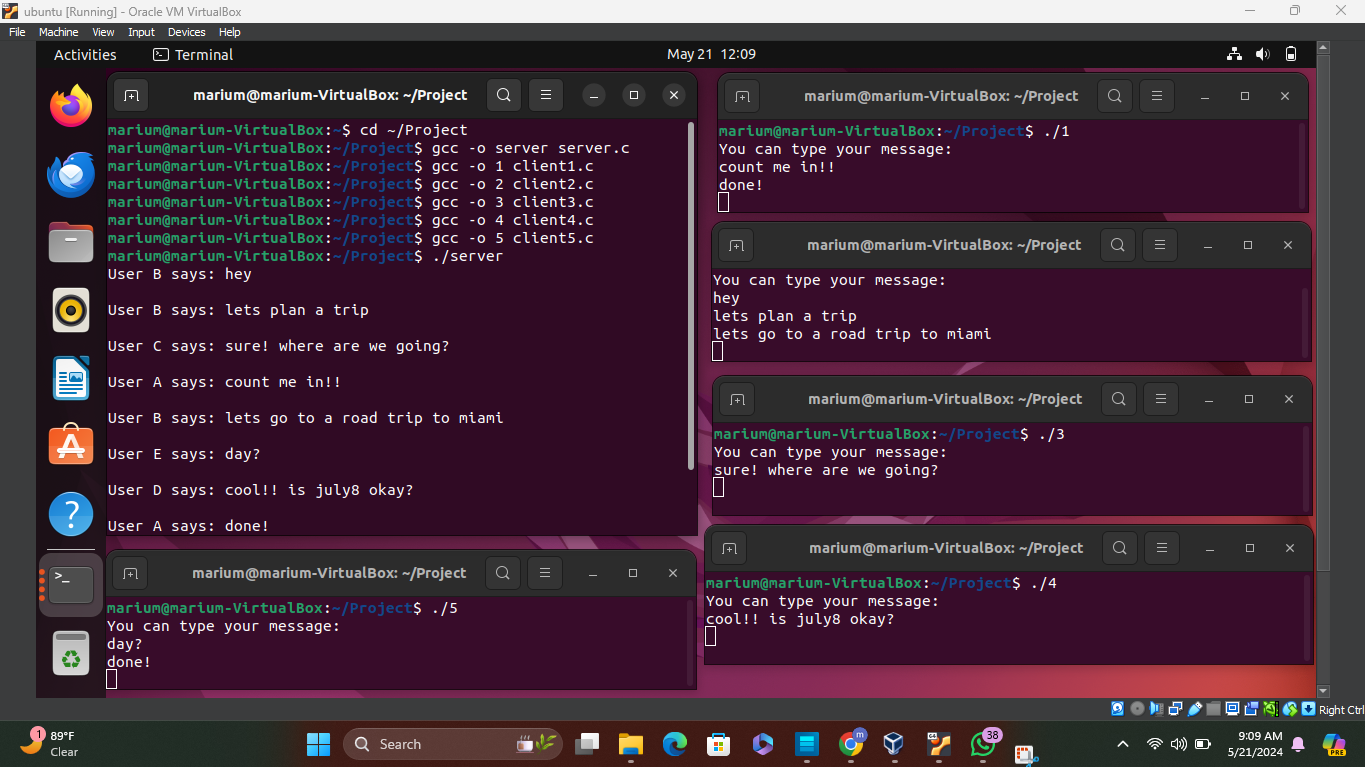
****

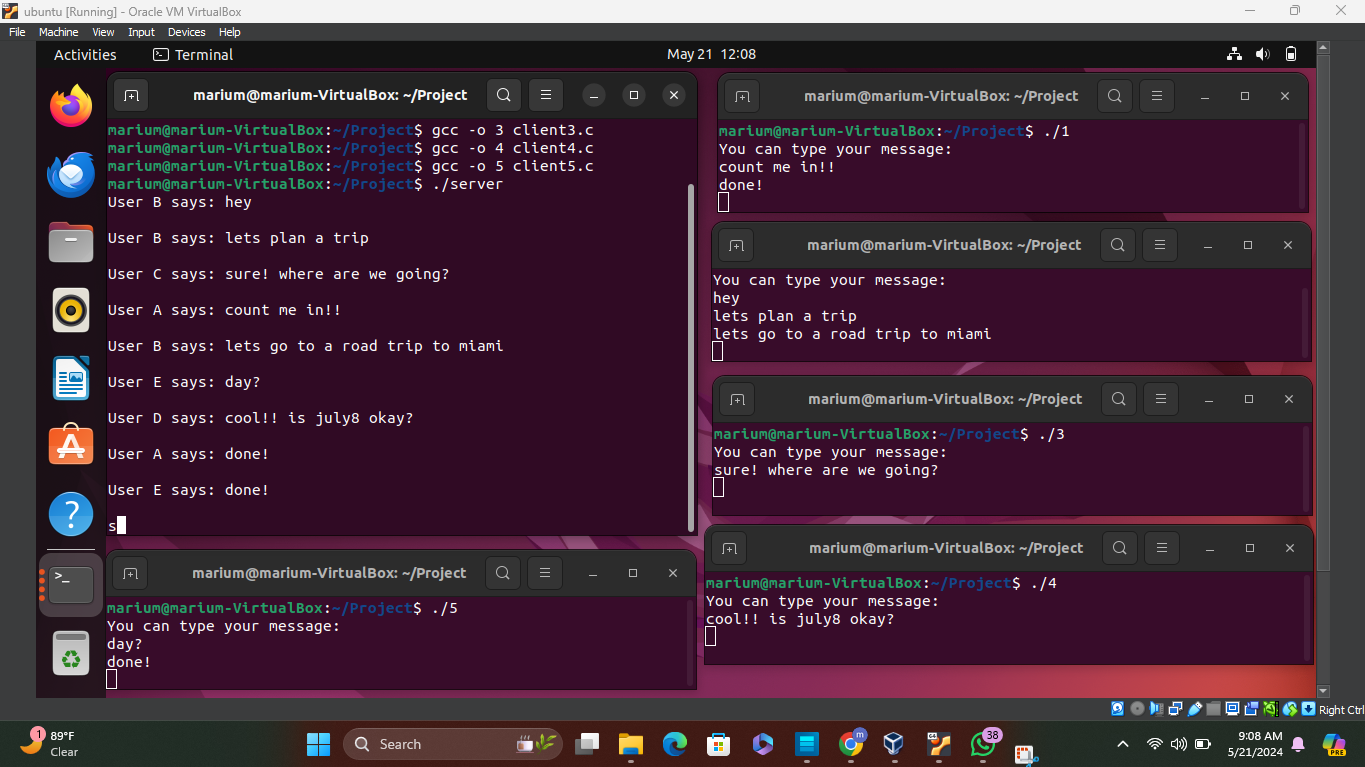
****

****

**WORKING:**

****

****

****

**CONCLUSION**

The Group Chat System demonstrates the implementation of client-server communication using named pipes in C. It allows multiple clients to exchange messages with a central server, facilitating real-time group communication. This project enhances understanding of inter-process communication and network programming concepts.