Multi-Threaded Chat Server and Client

Introduction

This project involves the development of multi-threaded chat server and client application in Linux using C. The application allows multiple clients to connect to a local server and send broadcast messages and private messages. The primary enhancements implemented in this project include **user authentication**, **message formatting** with timestamps, and **server logging** for user actions.

Project Objectives

The objectives of this project are:

- To develop a robust chat server and client application using C.
- To implement user authentication for secure communication.
- To format messages with timestamps for better readability.
- To log user actions on the server for monitoring and debugging purposes.

System Design

The system architecture consists of two main components:

1. **Chat Server:** Handles multiple client connections using threads, authenticates users, formats messages with timestamps, and logs user actions.
2. **Chat Client:** Connects to the server, handles user input, sends messages to the server, and displays messages received from the server.

Implementation Details

User Authentication

User authentication is implemented by prompting users to enter a username when they connect to the server. This username is then used to identify the user in subsequent communications.

Message Formatting with Timestamps

Messages sent by clients are formatted with a timestamp on the server before being broadcast to other clients. This helps users keep track of when messages were sent.

Server logging for User Actions

The server logs various user actions such as login, logout, and messages sent. The logs are stored in a file named '**server.log**' for monitoring and debugging purposes.

Execution of code

Header File (‘common.h’)

```
#pragma once

#define PORT 8080
#define MAX_CLIENTS 100
#define BUFFER_SIZE 1024
#define USERNAME_SIZE 50
#define LOG_FILE "server.log"

// Structure to hold client information
typedef struct {
    int socket;
    char username[USERNAME_SIZE];
    pthread_t thread;
} client_t;

// Function declarations
void log_message(const char *format, ...);
void send_message(int fd, const char *message);
void broadcast_message(const char *message, int exclude_fd);
char* get_timestamp();
```

Message Formatting with Timestamps (‘chat_server.c’)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <stdarg.h>
#include <time.h>
#include "common.h"

client_t *clients[MAX_CLIENTS];
pthread_mutex_t clients_mutex = PTHREAD_MUTEX_INITIALIZER;

void log_message(const char *format, ...) {
    FILE *log_file = fopen(LOG_FILE, "a");
    if (!log_file) return;

    va_list args;
    va_start(args, format);
    fprintf(log_file, "%s ", get_timestamp());
    vfprintf(log_file, format, args);
    fprintf(log_file, "\n");
    va_end(args);
    fclose(log_file);
```

```
}

char* get_timestamp() {
    static char buffer[20];
    time_t now = time(NULL);
    struct tm *tm_info = localtime(&now);
    strftime(buffer, 20, "%Y-%m-%d %H:%M:%S", tm_info);
    return buffer;
}

void *handle_client(void *arg) {
    client_t *client = (client_t *)arg;
    char buffer[BUFFER_SIZE];
    char message[BUFFER_SIZE];
    int bytes_received;

    // Authenticate user
    send_message(client->socket, "Enter your username: ");
    recv(client->socket, client->username, sizeof(client-
>username), 0);
    client->username[strcspn(client->username, "\n")] = 0; //
Remove newline

    // Log user login
    log_message("User %s logged in", client->username);
    snprintf(message, sizeof(message), "Welcome to the chat
server, %s!\n", client->username);
    send_message(client->socket, message);

    while ((bytes_received = recv(client->socket, buffer,
sizeof(buffer) - 1, 0)) > 0) {
        buffer[bytes_received] = '\0';
        if (strncmp(buffer, "/private", 8) == 0) {
            char target_user[USERNAME_SIZE];
            char private_message[BUFFER_SIZE];
            sscanf(buffer, "/private %s %[^\n]", target_user,
private_message);

            int found = 0;
            pthread_mutex_lock(&clients_mutex);
            for (int i = 0; i < MAX_CLIENTS; i++) {
                if (clients[i] && strcmp(clients[i]->username,
target_user) == 0) {
                    snprintf(message, sizeof(message), "[Private from
%s]: %s\n", client->username, private_message);
                    send_message(clients[i]->socket, message);
                    found = 1;
                    break;
                }
            }
            pthread_mutex_unlock(&clients_mutex);
            if (!found) {
                send_message(client->socket, "User not found.\n");
            }
        } else {
            snprintf(message, sizeof(message), "%s [%s]: %s\n",
client->username, get_timestamp(), buffer);
            broadcast_message(message, client->socket);
        }

        // Log message
        log_message("Message from %s: %s", client->username,
buffer);
    }

    // Log user logout
    log_message("User %s logged out", client->username);

    close(client->socket);
    pthread_mutex_lock(&clients_mutex);
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i] == client) {
            clients[i] = NULL;
            break;
        }
    }
    pthread_mutex_unlock(&clients_mutex);
    free(client);
    return NULL;
}

void broadcast_message(const char *message, int exclude_fd) {
    pthread_mutex_lock(&clients_mutex);
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i] && clients[i]->socket != exclude_fd) {
            send_message(clients[i]->socket, message);
        }
    }
    pthread_mutex_unlock(&clients_mutex);
}

void send_message(int fd, const char *message) {
    send(fd, message, strlen(message), 0);
}

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int opt = 1;
    socklen_t addr_len = sizeof(address);

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

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

    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");
        exit(EXIT_FAILURE);
    }

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

    while (1) {
        if ((new_socket = accept(server_fd, (struct sockaddr
*)&address, &addr_len)) < 0) {
            perror("Accept failed");
            exit(EXIT_FAILURE);
        }

        client_t *new_client = malloc(sizeof(client_t));
        new_client->socket = new_socket;

        pthread_mutex_lock(&clients_mutex);
        for (int i = 0; i < MAX_CLIENTS; i++) {
            if (clients[i] == NULL) {
                clients[i] = new_client;
                pthread_create(&new_client->thread, NULL,
handle_client, (void *)new_client);
                break;
            }
        }
        pthread_mutex_unlock(&clients_mutex);
    }

    return 0;
}
```

Server logging for User Actions (‘chat_client.c’)

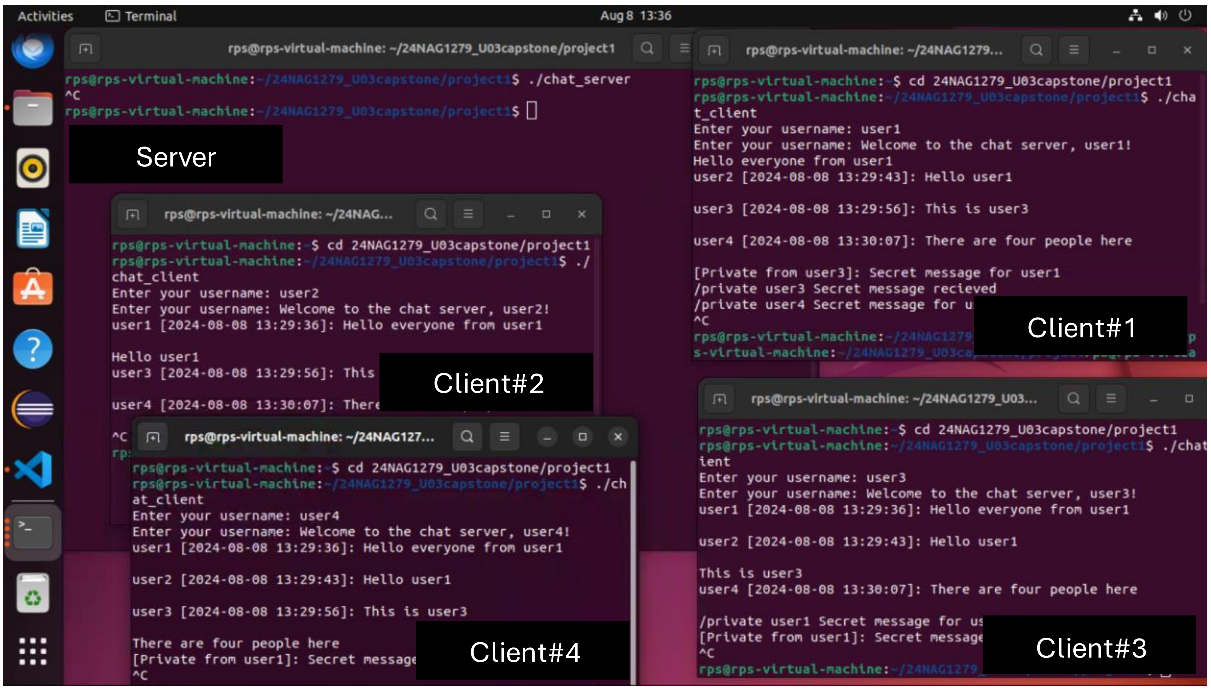
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>
#include "common.h"

void *receive_messages(void *sockfd) {
    int socket = *(int *)sockfd;
    char buffer[BUFFER_SIZE];
    int bytes_received;

    while ((bytes_received = recv(socket, buffer, sizeof(buffer) - 1,
0)) > 0) {
        buffer[bytes_received] = '\0';
        printf("%s", buffer);
    }

    printf("Server disconnected.\n");
    exit(0);
}

int main() {
    int sockfd;
    struct sockaddr_in server_address;
    char username[USERNAME_SIZE];
    char message[BUFFER_SIZE];
    pthread_t recv_thread;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

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

    if (inet_pton(AF_INET, "127.0.0.1", &server_address.sin_addr)
<= 0) {
        perror("Invalid address");
        exit(EXIT_FAILURE);
    }

    if (connect(sockfd, (struct sockaddr *)&server_address,
sizeof(server_address)) < 0) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }
    // Authenticate the user
    printf("Enter your username: ");
    fgets(username, USERNAME_SIZE, stdin);
    username[strcspn(username, "\n")] = 0; // Remove newline
    send(sockfd, username, strlen(username), 0);

    pthread_create(&recv_thread, NULL, receive_messages, (void
*)&sockfd);

    while (1) {
        fgets(message, sizeof(message), stdin);

        if (strncmp(message, "/private", 8) == 0) {
            send(sockfd, message, strlen(message), 0);
        } else {
            send(sockfd, message, strlen(message), 0);
        }
    }

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

Output

Execution of code

```
gcc -o chat_server chat_server.c -pthread
gcc -o chat_client chat_client.c -pthread
```

Server and Client output



```
rps@rps-virtual-machine:~/24NAG1279_U03capstone/project1$ ./chat_server
^C
rps@rps-virtual-machine:~/24NAG1279_U03capstone/project1$
```

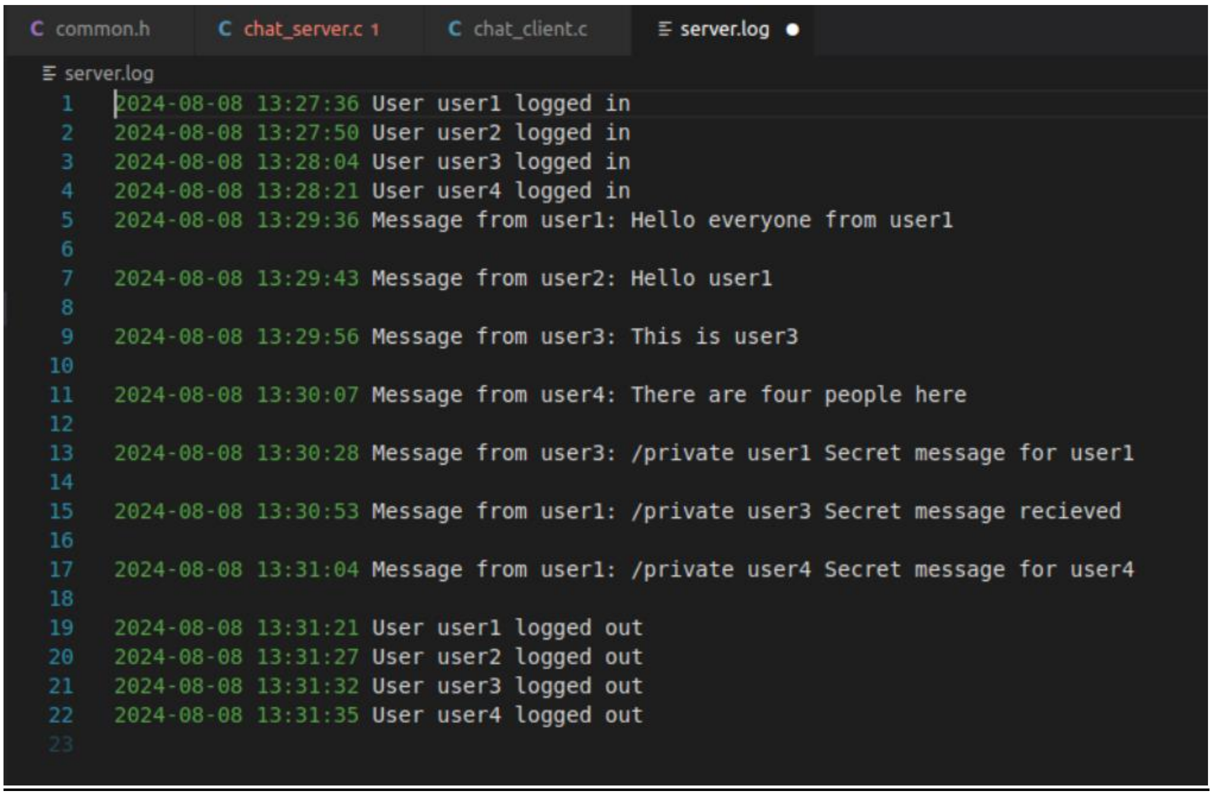
```
rps@rps-virtual-machine:~$ cd 24NAG1279_U03capstone/project1
rps@rps-virtual-machine:~/24NAG1279_U03capstone/project1$ ./cha
t_client
Enter your username: user1
Enter your username: Welcome to the chat server, user1!
Hello everyone from user1
user2 [2024-08-08 13:29:43]: Hello user1

user3 [2024-08-08 13:29:56]: This is user3

user4 [2024-08-08 13:30:07]: There are four people here

[Private from user3]: Secret message for user1
/private user3 Secret message recieved
/private user4 Secret message for u
^C
rps@rps-virtual-machine:~/24NAG1279
s-virtual-machine:~/24NAG1279_U03ca
```

```
rps@rps-virtual-machine:~$ cd 24NAG1279_U03capstone/project1
rps@rps-virtual-machine:~/24NAG1279_U03capstone/project1$ ./
chat_client
Enter your username: user2
Enter your username: Welcome to the chat server, user2!
user1 [2024-08-08 13:29:36]: Hello everyone from user1

Hello user1
user3 [2024-08-08 13:29:56]: This

user4 [2024-08-08 13:30:07]: There

^C
rp
```

```
rps@rps-virtual-machine:~$ cd 24NAG1279_U03capstone/project1
rps@rps-virtual-machine:~/24NAG1279_U03capstone/project1$ ./chat
ient
Enter your username: user3
Enter your username: Welcome to the chat server, user3!
user1 [2024-08-08 13:29:36]: Hello everyone from user1

user2 [2024-08-08 13:29:43]: Hello user1

This is user3
user4 [2024-08-08 13:30:07]: There are four people here

/private user1 Secret message for us
[Private from user1]: Secret message
^C
rps@rps-virtual-machine:~/24NAG1279
```

```
rps@rps-virtual-machine:~$ cd 24NAG1279_U03capstone/project1
rps@rps-virtual-machine:~/24NAG1279_U03capstone/project1$ ./ch
at_client
Enter your username: user4
Enter your username: Welcome to the chat server, user4!
user1 [2024-08-08 13:29:36]: Hello everyone from user1

user2 [2024-08-08 13:29:43]: Hello user1

user3 [2024-08-08 13:29:56]: This is user3

There are four people here
[Private from user1]: Secret message
^C
```

server.log



common.h | chat_server.c 1 | chat_client.c | server.log

server.log

```
2024-08-08 13:27:36 User user1 logged in
2024-08-08 13:27:50 User user2 logged in
2024-08-08 13:28:04 User user3 logged in
2024-08-08 13:28:21 User user4 logged in
2024-08-08 13:29:36 Message from user1: Hello everyone from user1

2024-08-08 13:29:43 Message from user2: Hello user1

2024-08-08 13:29:56 Message from user3: This is user3

2024-08-08 13:30:07 Message from user4: There are four people here

2024-08-08 13:30:28 Message from user3: /private user1 Secret message for user1

2024-08-08 13:30:53 Message from user1: /private user3 Secret message recieved

2024-08-08 13:31:04 Message from user1: /private user4 Secret message for user4

2024-08-08 13:31:21 User user1 logged out
2024-08-08 13:31:27 User user2 logged out
2024-08-08 13:31:32 User user3 logged out
2024-08-08 13:31:35 User user4 logged out

```

Future Enhancements

- **Encrypted Communication:** Implementing SSL/TLS to encrypt messages between the client and the server.
- **Persistent User Accounts:** Storing user accounts and passwords in a database for more secure and persistent authentication.
- **Improves User Interface:** Developing a Graphical User Interface (GUI) for the client application.
- **Group Chats:** Adding functionality for users to create and join group chats.
- **Message History:** Storing message history on the server and allowing clients to retrieve past messages.

Conclusion

This project demonstrates a multi-threaded chat server and client application with enhanced features such as user authentication, message formatting with timestamps and server logging. The project is designed to be scalable and can be further extended with additional functionalities in the future.