**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**

A screenshot of a computer screen

Description automatically generated

Client#4

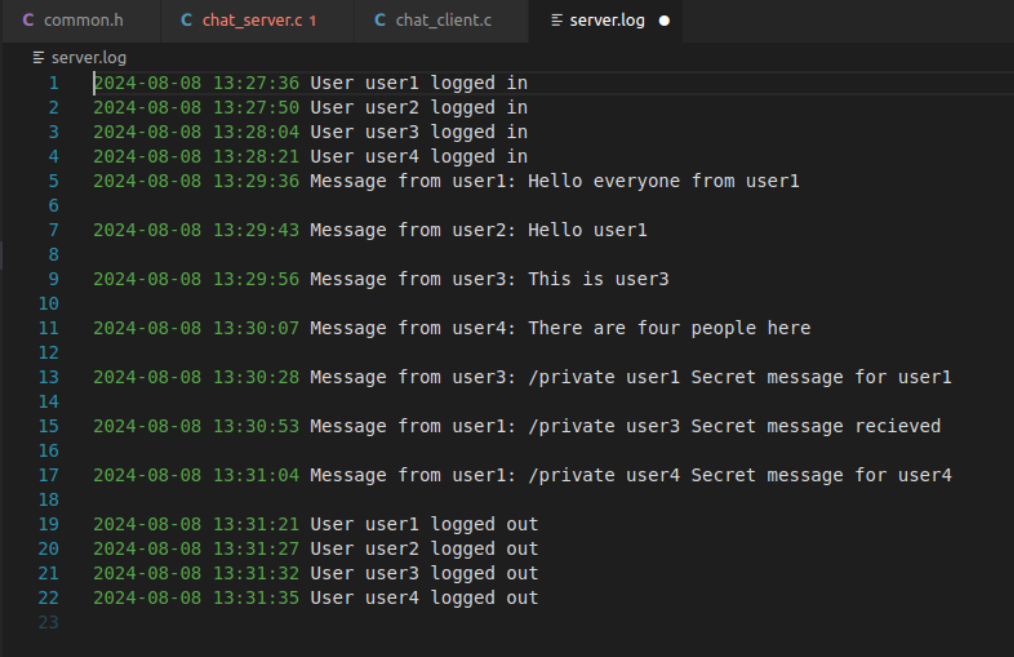
Client#2

Client#3

Client#1

Server

**server.log**

****

**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.