



Multi-Threaded Chat Server and Client

ADARSH KUMAR

24NAG1279_U03

Introduction

- The project involves developing a chat server and client application in Linux using C.
- Key features include user authentication, message formatting with timestamps and server logging.
- The system supports multiple clients simultaneously.

Project Objectives

- Develop a robust chat server and client in C.
- Implement user authentication for secure communication.
- Allow clients to send and receive broadcast and private messages.
- Format messages with timestamps for better readability.
- Log user actions on the server for monitoring and debugging.

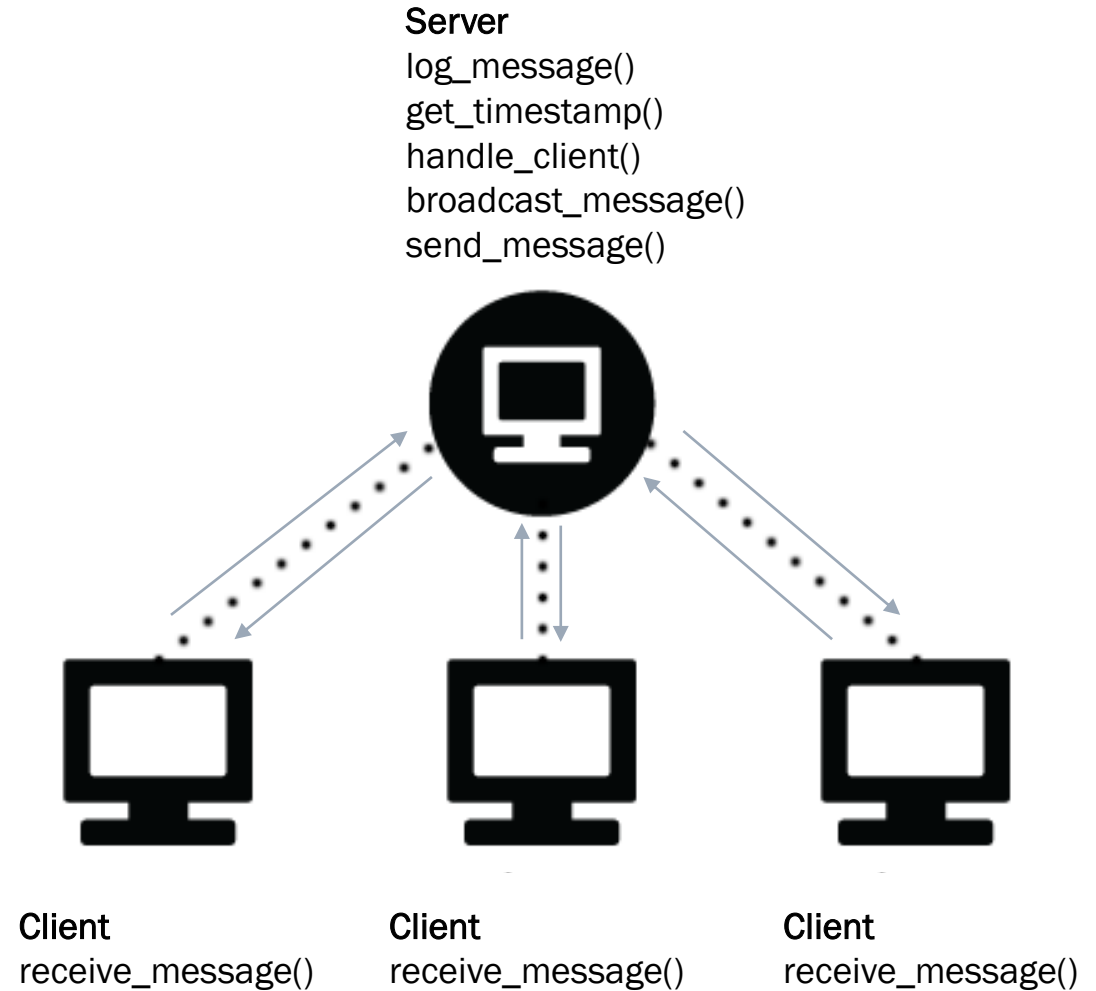
System Design

Chat Server

- `log_message()`
- `get_timestamp()`
- `handle_client()`
- `broadcast_message()`
- `send_message()`

Chat Client

- `receive_message()`



Chat Server Overview

- Handles multiple client connections using threads.
- Authenticates users and assigns usernames.
- Formats and broadcasts messages with timestamps.
- Logs user actions.

Chat Client Overview

- Connects to the server and handles user input.
- Sends messages to the server.
- Displays messages received from the server.
- Supports private messaging.

User authentication

- Users are prompted to enter a username upon connecting.
- Usernames are used to identify clients.
- Ensures secure and personalized communication.

Message Formatting with Timestamps

- Messages are formatted with timestamps before broadcasting.
- Example: [2024-08-07 12:00:05] user1: Hello everyone!
- Helps users track the timing of messages.

Server Logging for User Actions

- Logs various user actions like login, logout, and messages.
- Example log entry: [2024-08-07 12:00:00] User user1 logged in.
- Stored in 'server.log' for monitoring and debugging.

chat_server.c functions

log_message()

- ❖ logs user messages and actions to a server.log file
- ❖ Parameter – a format string followed by variable arguments
- ❖ Functionality – opens log file in append mode, writes current timestamp and formatted message to the log file and then closes the log file

get_timestamp()

- ❖ Generates a timestamp string.
Returns a pointer to a static buffer containing the formatted timestamp
- ❖ Gets the current time, converts it to a local time and formats the time as a string ('YYYY-MM-DD HH:MM:SS').

handle_client()

- ❖ Handles communication with a connected client.
- ❖ Parameters – '**arg**' a pointer to struct client
- ❖ Prompts user for a username and logs the user's login.
- ❖ Displays a welcome message.
- ❖ Receives messages from the client, formats them, and broadcasts them to other clients.
- ❖ Logs each message.
- ❖ Handles client disconnection, logging the logout and cleaning up of resources.

broadcast_message()

- ❖ Broadcasts a message to all connected clients except the sender.
- ❖ Parameters- '**message**' to be broadcasted and '**exclude_fd**' the file descriptor of the client to exclude (the sender)
- ❖ Iterates over all connected clients and sends message to all clients except '**exclude_fd**'.

`send_message()`

- ❖ Sends a message to a specific client.
- ❖ Parameters: **'fd'** the file descriptor of the client's socket.
'message' the message to be sent.
- ❖ Sends the message to the client using the **'send'** system call.

`main()`

- ❖ Creates a socket for the server.
- ❖ Configures the socket to reuse the address and port.
- ❖ Binds the socket to the specified port.
- ❖ Listens for incoming client connections.
- ❖ Accepts new client connections and spawns a new thread to handle each client.
- ❖ Manages client connections using a mutex for thread safety.

chat_client.c functions

receive_message()

- ❖ Continuously receives and displays messages from the server.
- ❖ Parameters- '**arg**' a pointer to the socket file descriptor.
- ❖ Receives messages from the server in a loop.
- ❖ Displays the received messages on the client's terminal.
- ❖ Closes the socket and exits when the server closes the connection.

main()

- ❖ Creates a socket for the client.
- ❖ Sets up the server address structure.
- ❖ Connects to the server using the specified IP address and port.
- ❖ Prompts the user to enter a username and sends it to the server.
- ❖ Creates a thread to handle incoming messages from the server.
- ❖ Continuously reads user input from the terminal and sends it to the server.
- ❖ Handles special commands (like private messaging) or regular messages.

The screenshot displays a Linux desktop environment with a sidebar of application icons on the left. Four terminal windows are open, each running a different program related to a chat application. The top-left window, titled 'Server', shows the execution of the `./chat_server` script. The other three windows, labeled 'Client#1', 'Client#2', and 'Client#4', show the execution of the `./chat_client` script. Each client window displays a series of messages, including usernames, timestamps, and system messages like 'Welcome to the chat server' and 'Hello everyone from user1'. The messages are interleaved, showing the real-time interaction between the server and the clients. For example, Client#1 sends a message, followed by Client#2, then Client#4, and so on. The server window shows the receipt of these messages and the corresponding responses sent back to the clients.

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

Server

```
rps@rps-virtual-machine: ~/24NAG1279_U03capstone/project1
rps@rps-virtual-machine:~/24NAG1279_U03capstone/project1$ 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 is user3
user4 [2024-08-08 13:30:07]: There are four people here

^C
rps@rps-virtual-machine:~/24NAG1279_U03capstone/project1$ cd 24NAG1279_U03capstone/project1
rps@rps-virtual-machine:~/24NAG1279_U03capstone/project1$ ./chat_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 for user1
^C
```

Client#2

Client#4

```
rps@rps-virtual-machine:~/24NAG1279_U03capstone/project1
rps@rps-virtual-machine:~/24NAG1279_U03capstone/project1$ cd 24NAG1279_U03capstone/project1
rps@rps-virtual-machine:~/24NAG1279_U03capstone/project1$ ./chat_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 user4
^C
rps@rps-virtual-machine:~/24NAG1279_U03capstone/project1$ cd 24NAG1279_U03capstone/project1
rps@rps-virtual-machine:~/24NAG1279_U03capstone/project1$ ./chat_client
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 user1
[Private from user1]: Secret message recieved
^C
rps@rps-virtual-machine:~/24NAG1279_U03capstone/project1$
```

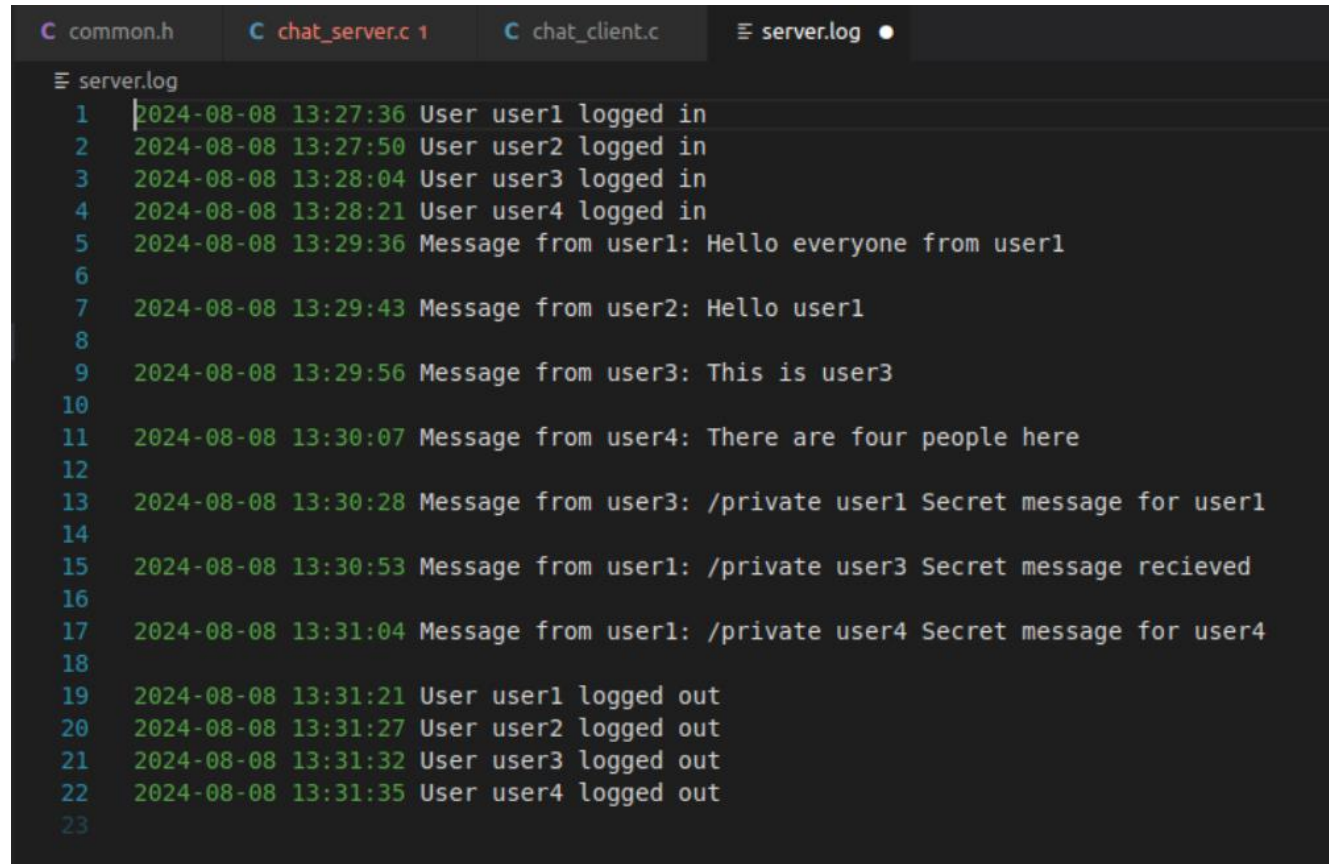
Client#1

Client#3

Output

Chat Server and 4 Clients

server.log



The screenshot shows a code editor with four tabs: `common.h`, `chat_server.c 1`, `chat_client.c`, and `server.log`. The `server.log` tab is active, displaying a log of chat server events. The log entries are numbered 1 through 23, with the last line (23) being empty. The log contains the following entries:

```
1 2024-08-08 13:27:36 User user1 logged in
2 2024-08-08 13:27:50 User user2 logged in
3 2024-08-08 13:28:04 User user3 logged in
4 2024-08-08 13:28:21 User user4 logged in
5 2024-08-08 13:29:36 Message from user1: Hello everyone from user1
6
7 2024-08-08 13:29:43 Message from user2: Hello user1
8
9 2024-08-08 13:29:56 Message from user3: This is user3
10
11 2024-08-08 13:30:07 Message from user4: There are four people here
12
13 2024-08-08 13:30:28 Message from user3: /private user1 Secret message for user1
14
15 2024-08-08 13:30:53 Message from user1: /private user3 Secret message recieved
16
17 2024-08-08 13:31:04 Message from user1: /private user4 Secret message for user4
18
19 2024-08-08 13:31:21 User user1 logged out
20 2024-08-08 13:31:27 User user2 logged out
21 2024-08-08 13:31:32 User user3 logged out
22 2024-08-08 13:31:35 User user4 logged out
23
```


Future Enhancements

- Encrypted communication using SSL/TLS.
- Persistent user accounts and password storage.
- Graphical user interface (GUI) for the client application.
- Group chat functionality.
- Message history retrieval.

Conclusion

- Developed a multi-threaded chat server and client with enhanced features.
- Implemented user authentication, message formatting, and server logging.
- The system is scalable and can be extended with additional functionalities.

The End