CSE344 – Final Project

Muhammet Talha Memişoğlu

210104004009

Introduction

This project implements a robust client-server chat application with file transfer (file transfer was just simulated) capabilities. The system is built using C and utilizes socket programming for network communication, along with multi-threading for concurrent operations. Additionally, this project was also implemented using Python and tested on Debain 11 and 12.

Code Explanation

Client Module

This module implements the client-side logic of a multi-threaded distributed chat and file sharing system. It is responsible for connecting to the server, handling user input, sending commands and files (Sending file is just a simulation. Server takes real data from sender, but recipient takes only a message whether operation is successful), receiving messages asynchronously, and gracefully shutting down on user interruption. The client supports interactive features like joining rooms, sending messages/files, and private messaging.

Initialization and Connection

- **Command-line Parameters:** The program takes two arguments: the server's IP and port. It validates the port range and establishes a TCP connection using sockets (AF_INET, SOCK_STREAM).

```
Usage: %s <server_ip> <port>
```

- **Signal Handling:** The client installs a SIGINT handler using signal(), allowing the program to exit gracefully when Ctrl+C is pressed.
- **Socket Connection:** socket() is used to create the socket, and connect() establishes the connection to the server. Errors are checked and reported.

Username Handling

- **Validation:** The client prompts the user for a username, verifies that it's alphanumeric and within a 16-character limit using is_valid_username().

- **Submission and Feedback:** The username is sent to the server, and feedback is received. The client will continue to prompt until a valid and available username is accepted in loop.

Threaded Message Receiver

- **Asynchronous Reception:** A detached POSIX thread (pthread_create) is created to continuously receive and handle server messages without blocking

```c
// Create receive thread
if (pthread_create(&receive_thread, NULL, receive_messages, NULL) != 0) {
    perror("Failed to create receive thread");
    close(server_socket);
    return 1;
}

// Detach the thread so it can clean up its own resources
pthread_detach(receive_thread);
```

  user input.
- **Special Commands:** The receiver handles special server messages such as:
  - REQUEST_FILE_SIZE — Responds with the size of a requested file.
  - START_UPLOAD — Sends the contents of a file in chunks.
  - General messages — Printed directly to the console.(e.g whisper, broadcast, error, warning messages etc.)

User Command Loop

- The main thread remains in a loop, reading user commands via fgets() and processing them. Supported commands include:
  - /help — Prints available commands.
  - /sendfile <filename> <user> — Sends a validated file to another user.
  - /exit — Sends a disconnect command and terminates the program.
  - All other strings are sent directly to the server.

File Sending Logic

- **Validation:** send_file() checks if the file exists, is readable, has a valid extension (.txt, .pdf, .jpg, .png), and is not empty.
- **Initiation:** Sends a /sendfile command to initiate the upload process.
- **Transmission:** The actual transfer is triggered by a START_UPLOAD message from the server and handled in the receiver thread.

Flow of File Sending Logic

1. In the **main input loop**, the user types:

```
/sendfile <filename> <username>
```

2. This command is matched in:

```
else if (strncmp(command, "/sendfile", 9) == 0) {
```

3. The command is parsed.
4. send_file(filename, recipient) is Called
    - This function performs file validations and file extensions check.
5. Command Sent to Server

```
sprintf(request, "/sendfile %s %s", filename, recipient);
if (send(server_socket, request, strlen(request), 0) < 0)
```

   Up to this point, everything happens in the main thread.
6. The receive thread (created with pthread_create) listens constantly for messages from the server using recv().
7. Server Sends REQUEST_FILE_SIZE:<filename>
    - Client extracts filename and calculates file size:

```
// Send file size back to server
char size_msg[256];
sprintf(size_msg, "FILE_SIZE:%lu", (unsigned long)file_size);
send(server_socket, size_msg, strlen(size_msg), 0);
```

8. Server Sends WAITING_FOR_UPLOAD:<filename>
    - Client prints a waiting message to notify user.
9. Server Sends START_UPLOAD:<filename>
    - This tells the client to start sending file content.
10. File Transmission Begins

```
while ((bytes_read = fread(file_buffer, 1, sizeof(file_buffer), file)) > 0)
{
    ssize_t bytes_sent = send(server_socket, file_buffer, bytes_read, 0);
```

11. Final success message is printed after sending all chunks.
12. Server takes file, check whether it is sent before and send successful message to recipient client. (Sending file from server to recipient part is simulation and is not actually implemented.)

Graceful Termination

- On Ctrl+C, the signal handler sets a termination flag (keep_running = 0), sends an /exit command to the server, and closes the socket using shutdown() and close().
- The receive thread observes the flag and exits automatically when the socket is closed.

Server Module

The server is structured into modules handling connection management, command processing, file transfer queuing, and logging. It uses multiple threads: one for each client and one for file transfer. It maintains multiple chat rooms and supports ephemeral messaging (no history saved).

Signal Handling

A SIGINT handler is registered using the signal() function. It uses a self-pipe trick (signal_pipe) to wake the select() loop safely. When triggered, it sets a global keep_running flag to initiate a graceful shutdown process that includes notifying clients and freeing all resources.

Server Initialization

1. **Socket Setup**: Initializes a TCP socket and binds it to the given port.
2. **Log File**: Creates a timestamped log file under the logs/ directory.
3. **Client and Room Structures**: Initializes mutexes and data structures for managing rooms and clients.
4. **File Transfer Worker**: Creates a separate detached thread that processes file transfers from a bounded queue.
5. **Directories**: Creates necessary folders such as uploads/ and received_files/ to track file operations and check unique file name.

Main Server Loop

- Uses select() to monitor:
    - Incoming connections on the server socket.
    - Signals via the self-pipe.
- Accepts client connections and validates usernames.
- Creates a dedicated detached thread for each client to handle interactions.

```c
// Create receive thread
if (pthread_create(&receive_thread, NULL, receive_messages, NULL) != 0) {
    perror("Failed to create receive thread");
    close(server_socket);
    return 1;
}

// Detach the thread so it can clean up its own resources
pthread_detach(receive_thread);
```

Client Handler

Each client is handled in a separate thread:

- Receives commands (e.g., /join, /broadcast, /sendfile, /exit).
- Maintains a current room context.
- Allows room switching and rejoining (tracked with a room history).
- Supports ephemeral messaging (no message storage).
- Disconnects cleanly upon error, exit command, or server shutdown.

Command Handling

- **Supported Commands:**
  - /join <room>: Join or create a chat room.
  - /leave: Leave current room.
  - /broadcast <message>: Sends a message to all users in the room.

```
for (int j = 0; j < rooms[i].num_clients; j++)
{
    if (rooms[i].clients[j] != sender)
    {
        send_message(rooms[i].clients[j]->socket, formatted_msg);
    }
}
```

  - /whisper <username> <message>: Sends a private message.

```
sprintf(formatted_msg, "%s[WHISPER] from %s: %s%s", COLOR_MAGENTA, sender->username, message, COLOR_RESET);
send_message(recipient->socket, formatted_msg);
```

  - /sendfile <filename> <username>: Initiates file transfer.
  - /exit: Disconnects from server.

File Transfer System

Features:

- Only specific file types are allowed (.txt, .pdf, .jpg, .png).
- A bounded queue (semaphore + mutex) ensures only one file transfer at a time.
- Sender receives upload instruction only when transfer starts.

- A separate file transfer worker receives file data and saves it. It is initialized in

```
// Create file transfer worker thread
if (pthread_create(&file_worker_thread, NULL, file_transfer_worker, NULL) != 0)
{
    write_log("ERROR: Failed to create file worker thread");
```

  main.
- Handles name conflicts by generating unique filenames. It is tracked with received_files directory.
- Notifies both sender and recipient about success or failure.
- If queue is full and clients try to send a file, error message is sent to that client.

Synchronization:

- **log_mutex**: Protects the log file from concurrent writes by multiple threads, ensuring clean and readable logs.
- **queue_mutex:** Secures access to the shared file transfer queue. It prevents race conditions when file requests are enqueued or dequeued.
- **room_mutex:** Manages access to shared chat room structures, allowing safe room creation, join, leave, and broadcast operations.
- **clients_mutex:** Guards the global list of active clients. This ensures thread-safe addition and removal of clients during connection and disconnection.
- **client->client_mutex:** Each client structure has its own mutex to synchronize access to its socket and data. This prevents concurrent writes or reads (e.g., sending messages or transferring files) from causing inconsistencies.
- **upload_semaphore:** Controls the maximum number of concurrent file transfers. This prevents the server from being overwhelmed by multiple simultaneous uploads.
- **queue_cond:** Used by the file transfer worker thread to wait efficiently when there are no pending uploads. It is signaled when new file tasks are added to the queue.

Graceful Shutdown

On SIGINT:

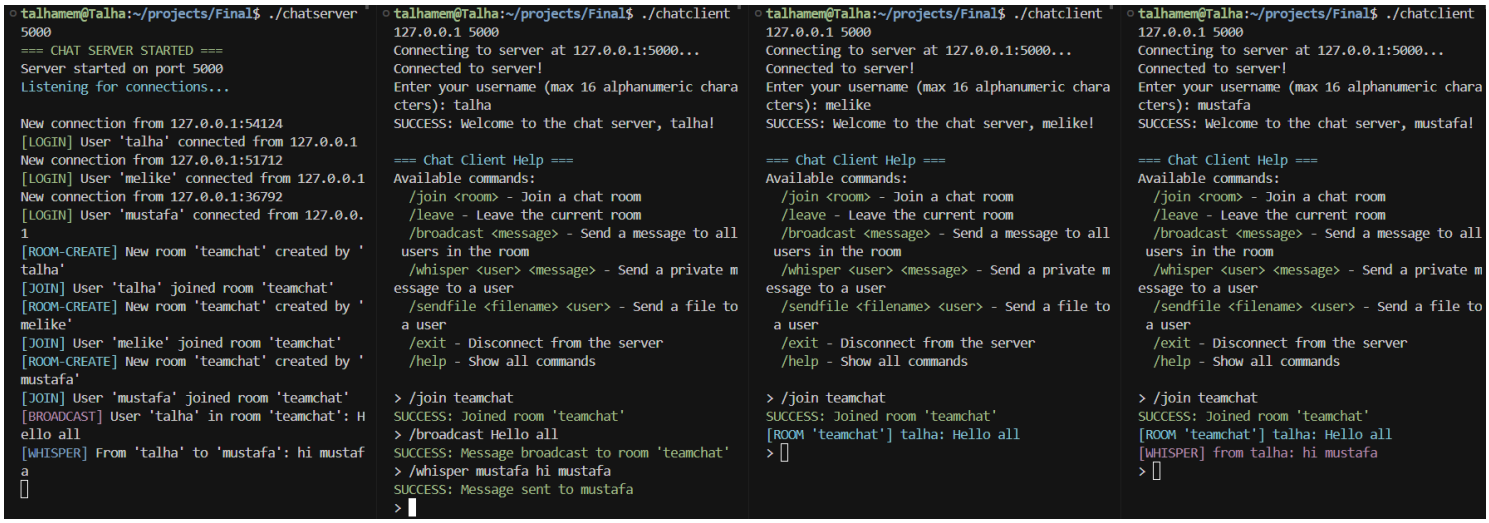
- Stops accepting new clients.
- Notifies all clients about shutdown.
- Closes sockets and frees memory.
- Ensures threads terminate cleanly.
- Closes log file and exits process.

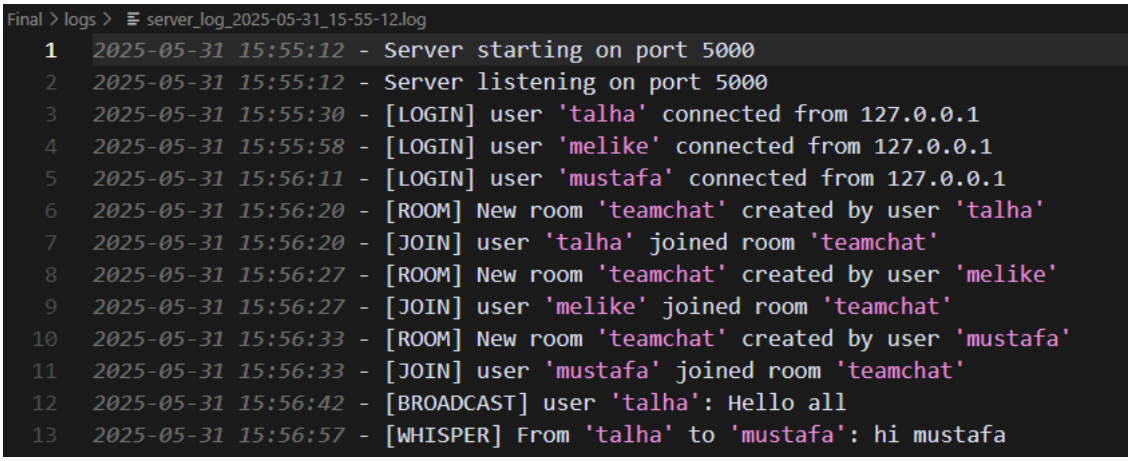
Usage: ./chatserver <port>

./chatclient <server_ip> <port>

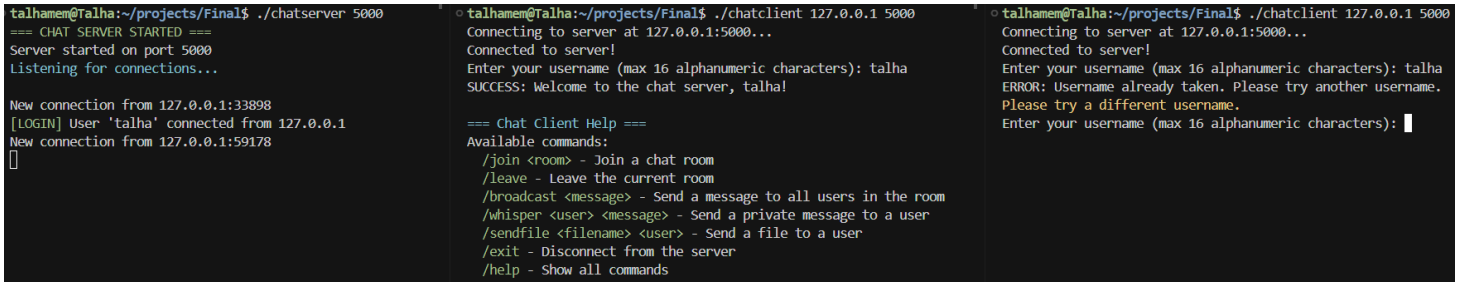
Screenshots

1.Login, Join, Whisper and Broadcast





2. Duplicate Username



```
2025-05-31 16:00:26 - [LOGIN] user 'talha' connected from 127.0.0.1
2025-05-31 16:00:31 - [REJECTED] Duplicate username attempted: talha from 127.0.0.1
```

3.File Upload Queue Limit

```
> /sendfile deneme.txt melike
ERROR: Upload queue is full, try again later
```

```
- [FILE-QUEUE] Upload queue full - rejected 'deneme.txt' from mustafa to melike
```

(For testing purposes, I added a delay in the file transfer using sleep (sleep is dangerous and may cause critical problems for this project so I used for only testing purpose) and set the queue size to 1. Then I tested with 2 users. However, in the actual implementation, there is no sleep call, and the queue size is set to 5.)

4.Unexpected Disconnection

```
talhamem@Talha:~/projects/Final$ ./chatserver 5000
=== CHAT SERVER STARTED ===
Server started on port 5000
Listening for connections...

New connection from 127.0.0.1:60498
[LOGIN] User 'talha' connected from 127.0.0.1
[ROOM-CREATE] New room 'teamchat' created by 'talha'
[JOIN] User 'talha' joined room 'teamchat'
[LOGOUT] User 'talha' disconnected
```

```
Connected to server!
Enter your username (max 16 alphanumeric characters): talha
SUCCESS: Welcome to the chat server, talha!

=== Chat Client Help ===
Available commands:
  /join <room> - Join a chat room
  /leave - Leave the current room
  /broadcast <message> - Send a message to all users in the room
  /whisper <user> <message> - Send a private message to a user
  /sendfile <filename> <user> - Send a file to a user
  /exit - Disconnect from the server
  /help - Show all commands

> /join teamchat
SUCCESS: Joined room 'teamchat'
> ^C
Received Ctrl+C, exiting gracefully...
Disconnected from server
talhamem@Talha:~/projects/Final$
```

```
2025-05-31 16:04:49 - Server starting on port 5000
2025-05-31 16:04:49 - Server listening on port 5000
2025-05-31 16:04:52 - [LOGIN] user 'talha' connected from 127.0.0.1
2025-05-31 16:04:56 - [ROOM] New room 'teamchat' created by user 'talha'
2025-05-31 16:04:56 - [JOIN] user 'talha' joined room 'teamchat'
2025-05-31 16:04:58 - [LOGOUT] user 'talha' disconnected. Cleaned up resources.
2025-05-31 16:04:58 - [ROOM] Room 'teamchat' is now empty and marked inactive
```

5.Room Switching

```
talhamem@Talha:~/projects/Final$ ./chatserver 5000
=== CHAT SERVER STARTED ===
Server started on port 5000
Listening for connections...

New connection from 127.0.0.1:55920
[LOGIN] User 'talha' connected from 127.0.0.1
[ROOM-CREATE] New room 'teamchat' created by 'talha'
[JOIN] User 'talha' joined room 'teamchat'
[ROOM-CREATE] New room 'teamchat2' created by 'talha'
[ROOM] User 'talha' left 'teamchat', joined 'teamchat2'
```

```
talhamem@Talha:~/projects/Final$ ./chatclient 127.0.0.1 5000
Connecting to server at 127.0.0.1:5000...
Connected to server!
Enter your username (max 16 alphanumeric characters): talha
SUCCESS: Welcome to the chat server, talha!

=== Chat Client Help ===
Available commands:
  /join <room> - Join a chat room
  /leave - Leave the current room
  /broadcast <message> - Send a message to all users in the room
  /whisper <user> <message> - Send a private message to a user
  /sendfile <filename> <user> - Send a file to a user
  /exit - Disconnect from the server
  /help - Show all commands

> /join teamchat
SUCCESS: Joined room 'teamchat'
> /join teamchat2
SUCCESS: Joined room 'teamchat2'
>
```

```
2025-05-31 16:12:20 - Server starting on port 5000
2025-05-31 16:12:20 - Server listening on port 5000
2025-05-31 16:12:24 - [LOGIN] user 'talha' connected from 127.0.0.1
2025-05-31 16:12:35 - [ROOM] New room 'teamchat' created by user 'talha'
2025-05-31 16:12:35 - [JOIN] user 'talha' joined room 'teamchat'
2025-05-31 16:12:41 - [ROOM] Room 'teamchat' is now empty and marked inactive
2025-05-31 16:12:41 - [ROOM] New room 'teamchat2' created by user 'talha'
2025-05-31 16:12:41 - [ROOM] user 'talha' left room 'teamchat', joined 'teamchat2'
```

6.Oversized File Rejection

```
> /sendfile FinalProject.pdf melike
ERROR: File size exceeds limit (3MB)
>
```

```
2025-05-31 16:15:11 - [LOGIN] user 'melike' connected from 127.0.0.1
2025-05-31 16:15:28 - [ERROR] File 'FinalProject.pdf' from user 'talha' exceeds size limit.
```

7.SIGINT Server Shutdown

```
talhamem@Talha:~/projects/Final$ ./chatserver 5000
=== CHAT SERVER STARTED ===
Server started on port 5000
Listening for connections...

New connection from 127.0.0.1:44700
[LOGIN] User 'talha' connected from 127.0.0.1
New connection from 127.0.0.1:44710
[LOGIN] User 'melike' connected from 127.0.0.1
^C
=== SERVER SHUTTING DOWN ===
=== SERVER SHUTDOWN COMPLETE ===
talhamem@Talha:~/projects/Final$
```

```
Enter your username (max 16 alphanumeric characters): talha
SUCCESS: Welcome to the chat server, talha!

=== Chat Client Help ===
Available commands:
  /join <room> - Join a chat room
  /leave - Leave the current room
  /broadcast <message> - Send a message to all users in the room
  /whisper <user> <message> - Send a private message to a user
  /sendfile <filename> <user> - Send a file to a user
  /exit - Disconnect from the server
  /help - Show all commands

> /sendfile FinalProject.pdf melike
ERROR: File size exceeds limit (3MB)
[SERVER] Server is shutting down. Goodbye!
>
Connection closed by server
```

```
Connecting to server at 127.0.0.1:5000...
Connected to server!
Enter your username (max 16 alphanumeric characters): melike
SUCCESS: Welcome to the chat server, melike!

=== Chat Client Help ===
Available commands:
  /join <room> - Join a chat room
  /leave - Leave the current room
  /broadcast <message> - Send a message to all users in the room
  /whisper <user> <message> - Send a private message to a user
  /sendfile <filename> <user> - Send a file to a user
  /exit - Disconnect from the server
  /help - Show all commands

[SERVER] Server is shutting down. Goodbye!
>
Connection closed by server
```

```
2025-05-31 16:15:03 - Server starting on port 5000
2025-05-31 16:15:03 - Server listening on port 5000
2025-05-31 16:15:09 - [LOGIN] user 'talha' connected from 127.0.0.1
2025-05-31 16:15:11 - [LOGIN] user 'melike' connected from 127.0.0.1
2025-05-31 16:15:28 - [ERROR] File 'FinalProject.pdf' from user 'talha' exceeds size limit.
Server shutdown complete
```

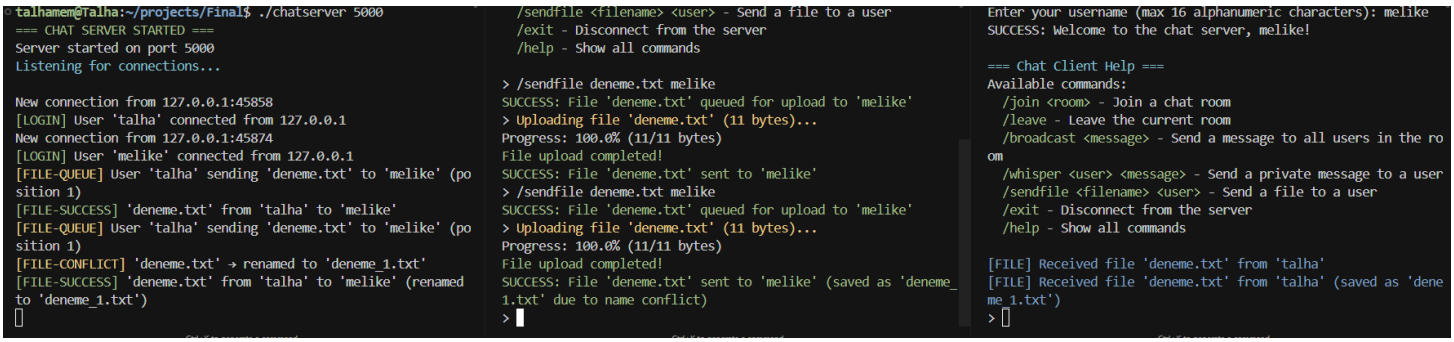
8.Rejoining rooms

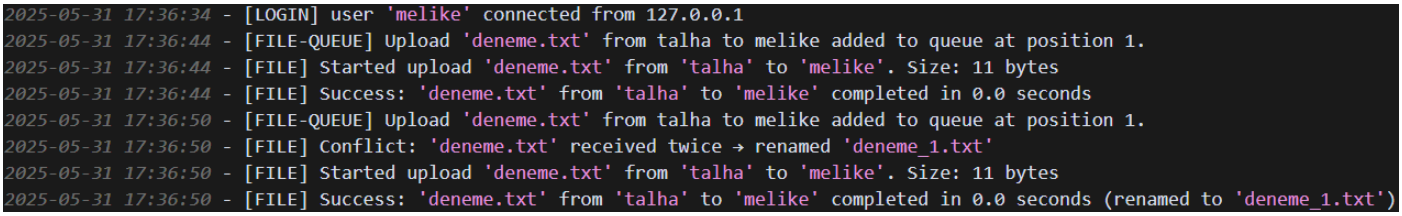
```
talhamem@Talha:~/projects/Final$ ./chatserver 5000
=== CHAT SERVER STARTED ===
Server started on port 5000
Listening for connections...

New connection from 127.0.0.1:55740
[LOGIN] User 'talha' connected from 127.0.0.1
New connection from 127.0.0.1:55756
[LOGIN] User 'melike' connected from 127.0.0.1
[ROOM-CREATE] New room 'teamchat' created by 'talha'
[JOIN] User 'talha' joined room 'teamchat'
[ROOM-CREATE] New room 'teamchat' created by 'melike'
[JOIN] User 'melike' joined room 'teamchat'
[LEAVE] User 'talha' left room 'teamchat'
[ROOM-CREATE] New room 'teamchat' created by 'talha'
[ROOM] User 'talha' rejoined 'teamchat'
```

```
talhamem@Talha:~/projects/Final$ ./chatclient 127.0.0.1 5000
Connecting to server at 127.0.0.1:5000...
Connected to server!
Enter your username (max 16 alphanumeric characters): talha
SUCCESS: Welcome to the chat server, talha!

=== Chat Client Help ===
Available commands:
  /join <room> - Join a chat room
  /leave - Leave the current room
  /broadcast <message> - Send a message to all users in the room
  /whisper <user> <message> - Send a private message to a user
  /sendfile <filename> <user> - Send a file to a user
  /exit - Disconnect from the server
  /help - Show all commands

> /join teamchat
SUCCESS: Joined room 'teamchat'
> /leave
SUCCESS: Left room 'teamchat'
> /join teamchat
SUCCESS: Rejoined room 'teamchat' (previous messages not shown)
>
```

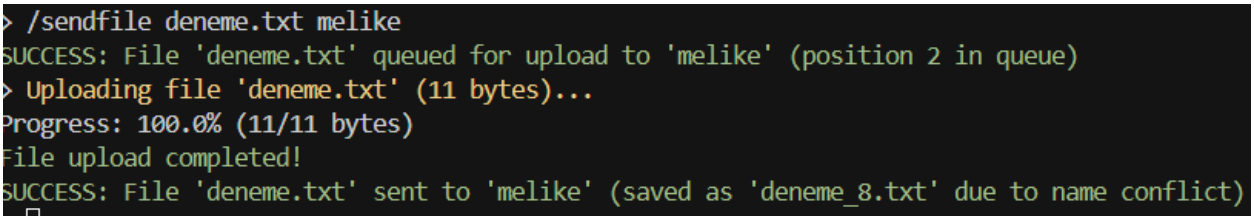
```
2025-05-31 16:34:26 - [ROOM] user 'talha' rejoined 'teamchat'
```

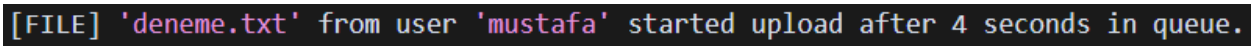
9.Same Filename Collision





10.Full Queue Wait Duration





(For testing purposes, I added a delay in the file transfer using sleep then I tested with 2 users. However, in the actual implementation, there is no sleep call (sleep is dangerous and may cause critical problems for this project so I used for only testing purpose).

Implementation of same logic in Python

After I wrote the code in c, I was helped a lot for converting C to Python from AI. This is the brief report that focuses on differences between C and Python implementation.

Architecture & Memory Management

- **C**: Manual memory allocation/deallocation with malloc()/free()
- **Python**: Automatic garbage collection, no manual memory management

Threading & Synchronization

- **C**: pthread library with manual mutex/semaphore management
- **Python**: Built-in threading module with simpler Lock/Queue objects

```
self.clients_lock = threading.Lock()
self.rooms_lock = threading.Lock()
self.log_lock = threading.Lock()
```

Signal Handling

- **C**: Self-pipe trick for signal-safe operations
- **Python**: Direct signal handling (simpler but less robust)

Network Programming

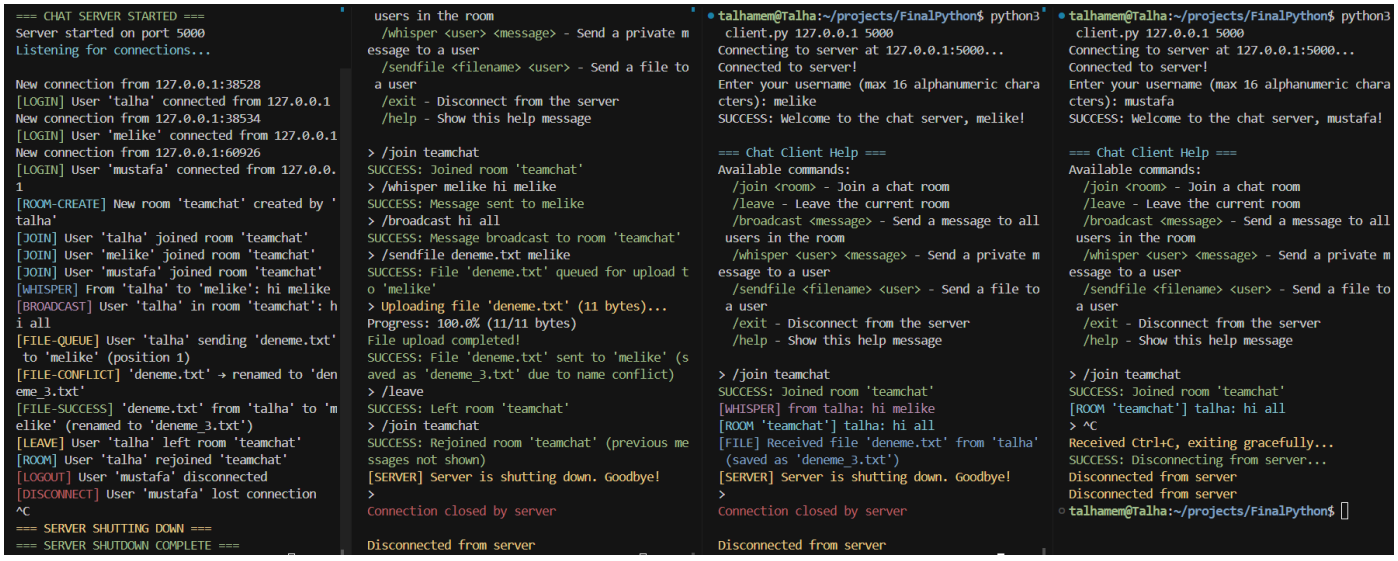
- **C**: Low-level socket APIs with manual address structures
- **Python**: Higher-level socket interface with simplified operations

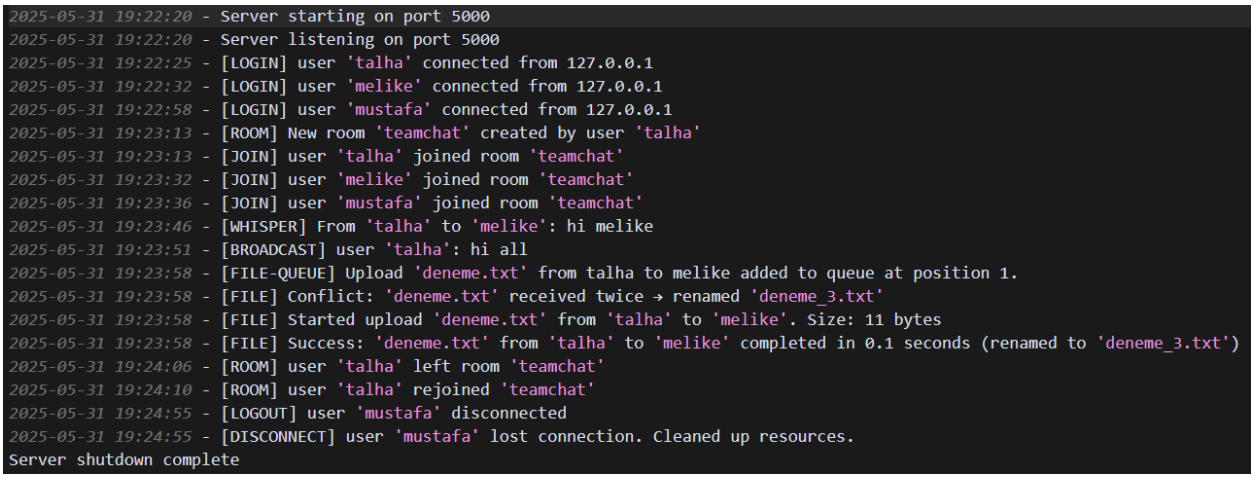
Main structure and functionality same and outputs are correct but there can be issues with file transfer implementation.

Usage: python3 server.py <port>

python3 client.py <server_ip> <port>

Screenshot for Python





Conclusion

This project demonstrates the implementation of a multi-threaded chat and file server with a corresponding client in C. It highlights real-time messaging, room management, file transfer queuing, and safe concurrency using threads, mutexes, and semaphores. The client interacts smoothly via commands, while the server handles multiple users and transfers efficiently.

I had a hard time when I tried to implement file sending from server to recipient. Then I decided to change this part to a simulation and just notify the recipient because I did not have so much time, and real implementation is not required for this project. I was also helped from AI especially for TCP implementations, error handling and converting C code to Python.