CSE 344 – Final Project Report
Multi-threaded Distributed Chat and File Server

Cemal BOLAT
Student ID: 210104004010

31'st May 2025

1 Introduction and Problem Definition

This project aims to implement a multithreaded TCP-based distributed chat and file server that supports multiple concurrent clients, real-time messaging, and file sharing.

The central objective is to design a robust and efficient communication system that:

- Handles multiple clients simultaneously using threads.
- Ensures thread-safe operations through proper synchronization mechanisms.
- Allows users to communicate in chat rooms or via private messages.
- Manages file uploads using a queue that simulates limited system resources.

To achieve this, the server must utilize thread management, interprocess communication (IPC), and TCP socket programming. Clients interact with the server through command-line interfaces, using predefined commands for joining rooms, broadcasting messages, private chatting, and sending files.

The system is designed to be reliable and responsive under concurrent usage, with proper validation, logging, and graceful handling of unexpected conditions such as duplicate usernames or client crashes. This project not only deepens the understanding of concurrency and networking, but also reinforces disciplined software engineering practices such as modular design, error handling, and resource management.

2 System Design

2.1 Overall Architecture

The system follows a client-server model:

- The server listens for incoming TCP connections.
- Each client is handled in a dedicated thread.
- Commands are parsed and acted on in real time.
- A shared upload queue with limited capacity (5 slots) is managed with semaphores and mutexes.

3 Design Details

3.1 Architecture Overview

The system is designed using a modular and event-driven architecture that separates responsibilities between multiple threads. The server opens a TCP socket on a given port and starts listening for incoming client connections. Each new client is handled in a dedicated thread to ensure concurrent support for multiple users.

Before handling clients, the server creates a dedicated **file transfer manager thread**, which handles all file delivery logic through a shared upload queue.

3.2 Thread Model

- **Main Thread:** Initializes the server socket and accepts incoming TCP connections using `accept()`.
- **Client Handler Thread:** For each connected client, a new thread is created. This thread is responsible for:
  - Reading incoming messages from the client.
  - Parsing commands such as `/join`, `/broadcast`, `/sendfile`, etc.
  - Communicate with shared resources (rooms, log, file queue) safely.
- **File Transfer Thread:** A dedicated thread handles file transfer coordination.
  1. Waits for a file transfer request in the upload queue.
  2. Receives the file header (e.g., filename, size, destination) from the sender.
  3. Sends acknowledgment to the sender to begin transmission.
  4. Receives the file content and stores it in memory.
  5. Sends the file to the recipient client.
  6. Notifies the sender that the transfer is complete.

3.3 IPC and Upload Queue

To manage limited file upload capacity, a shared file queue is implemented using IPC mechanisms:

- A global queue structure holds pending file uploads.
- Access to the queue is synchronized using a `pthread_mutex_t`.
- A counting semaphore controls the maximum number of concurrent uploads (e.g., 5).
- The file transfer thread dequeues and processes requests in FIFO order.

3.4 Command Processing

All client commands are parsed in their dedicated thread. Based on the command type:

- The server updates internal state (e.g., room membership).
- Sends messages to other clients if necessary.
- Enqueues file uploads when the `/sendfile` command is received.

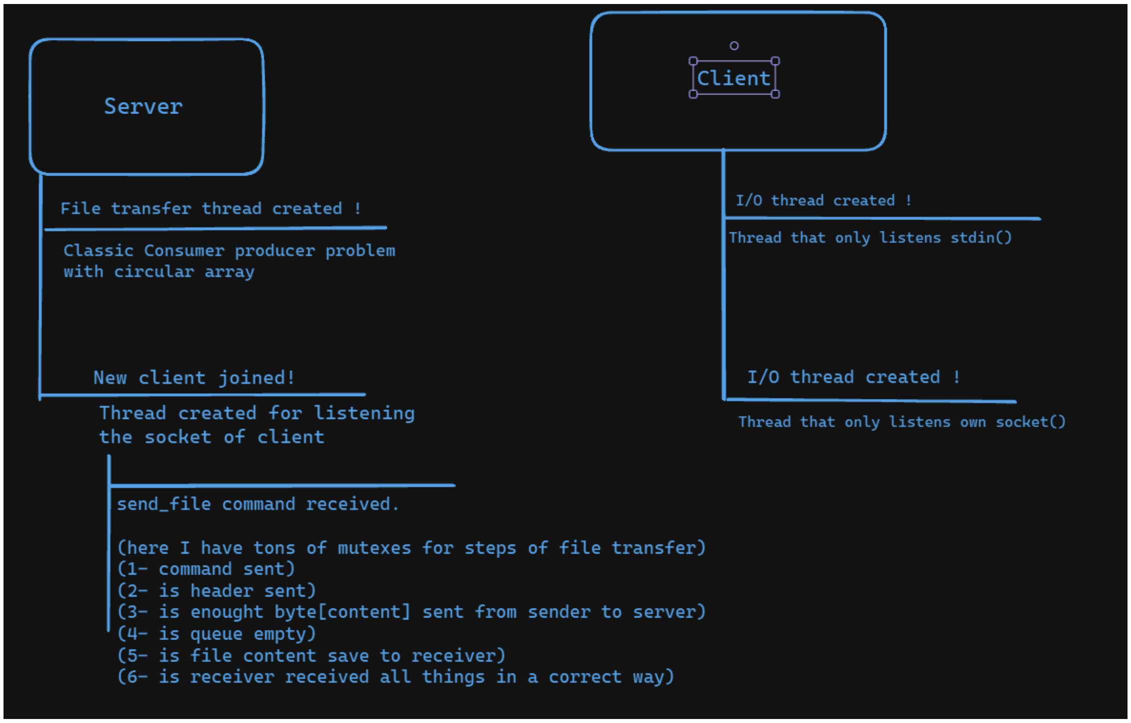

Server
File transfer thread created !
Classic Consumer producer problem
with circular array
New client joined!
Thread created for listening
the socket of client
send_file command received.
(here I have tons of mutexes for steps of file transfer)
(1- command sent)
(2- is header sent)
(3- is enought byte[content] sent from sender to server)
(4- is queue empty)
(5- is file content save to receiver)
(6- is receiver received all things in a correct way)
Client
I/O thread created !
Thread that only listens stdin()
I/O thread created !
Thread that only listens own socket()

Figure 1: Threaded Server-Client Architecture

Issues Faced and How They Were Solved

Throughout the development process, several critical concurrency and synchronization issues were encountered and resolved:

- **Standard output interleaving:** Using `printf` from multiple threads caused race conditions and unreadable logs. This was resolved by implementing a custom `safe_printf()` function that uses a mutex to serialize access to the output stream.
- **Data races despite `sig_atomic_t`:** Even though `sig_atomic_t` was used for shared flags, it failed to prevent data races under concurrent access. This was addressed by switching to atomic operations using `<stdatomic.h>` functions such as `atomic_load()` and `atomic_store()`.
- **Synchronization in file transfer pipeline:** Multiple threads interacting with the file queue led to inconsistencies. To solve this, mutexes and condition variables were added at each stage of the file delivery process to ensure correct sequencing and mutual exclusion.
- **Thread-safe logging and debugging:** Logging from multiple threads created confusion during debugging. Centralized logging was implemented with internal locking to preserve the chronological order of messages.

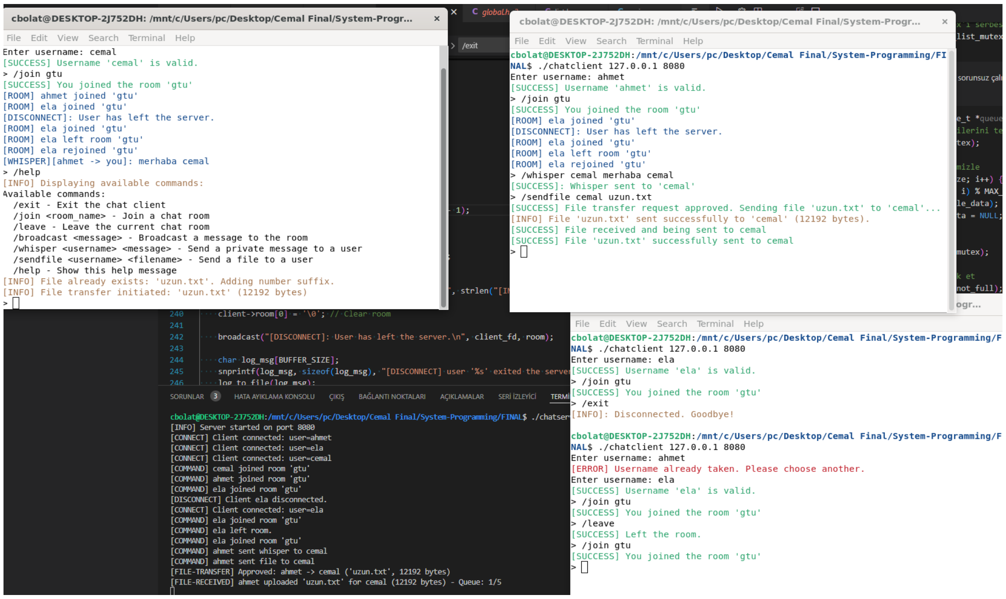
Test Cases and Results

Normal Execution with 3 Clients

Test Description: Three clients join the server successfully, send and receive messages, and disconnect gracefully.

Expected Behavior: All usernames should be accepted, message delivery should be complete, and no errors should appear in either client or server outputs.

Observed Behavior: The system functioned as expected. Each client joined the chat, sent messages, and received messages from others without issue. The server logged the session accurately.



cbolat@DESKTOP-2J752DH: /mnt/c/Users/pc/Desktop/Cemal Final/System-Progr...
File Edit View Search Terminal Help

```
Enter username: cemal
[SUCCESS] Username 'cemal' is valid.
> /join gtu
[SUCCESS] You joined the room 'gtu'
[ROOM] ahmet joined 'gtu'
[ROOM] ela joined 'gtu'
[DISCONNECT]: User has left the server.
[ROOM] ela joined 'gtu'
[ROOM] ela left room 'gtu'
[ROOM] ela rejoined 'gtu'
[WHISPER][ahmet -> you]: merhaba cemal
> /help
[INFO] Displaying available commands:
Available commands:
  /exit - Exit the chat client
  /join <room_name> - Join a chat room
  /leave - Leave the current chat room
  /broadcast <message> - Broadcast a message to the room
  /whisper <username> <message> - Send a private message to a user
  /sendfile <username> <filename> - Send a file to a user
  /help - Show this help message
[INFO] File already exists: 'uzun.txt'. Adding number suffix.
[INFO] File transfer initiated: 'uzun.txt' (12192 bytes)
> 
```

cbolat@DESKTOP-2J752DH: /mnt/c/Users/pc/Desktop/Cemal Final/System-Progr...
File Edit View Search Terminal Help

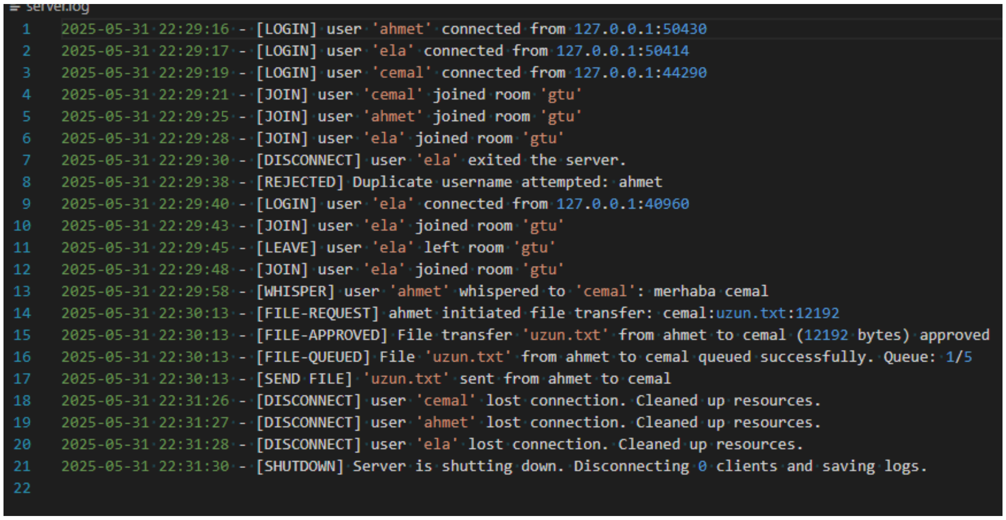
```
cbolat@DESKTOP-2J752DH:/mnt/c/Users/pc/Desktop/Cemal Final/System-Programming/FI
NAL$ ./chatclient 127.0.0.1 8080
Enter username: ahmet
[SUCCESS] Username 'ahmet' is valid.
> /join gtu
[SUCCESS] You joined the room 'gtu'
[ROOM] ela joined 'gtu'
[DISCONNECT]: User has left the server.
[ROOM] ela joined 'gtu'
[ROOM] ela left room 'gtu'
[ROOM] ela rejoined 'gtu'
> /whisper cemal merhaba cemal
[SUCCESS]: Whisper sent to 'cemal'
> /sendfile cemal uzun.txt
[SUCCESS] File transfer request approved. Sending file 'uzun.txt' to 'cemal'...
[INFO] File 'uzun.txt' sent successfully to 'cemal' (12192 bytes).
[SUCCESS] File received and being sent to cemal
[SUCCESS] File 'uzun.txt' successfully sent to cemal
> 
```

```
    client->room[0] = '\0'; // Clear room

    broadcast("[DISCONNECT]: User has left the server.\n", client_fd, room);

    char log_msg[BUFFER_SIZE];
    snprintf(log_msg, sizeof(log_msg), "[DISCONNECT] user '%s' exited the server
    log to file(log msg);
```

```
cbolat@DESKTOP-2J752DH:/mnt/c/Users/pc/Desktop/Cemal Final/System-Programming/FINAL$ ./chatser
[INFO] Server started on port 8080
[CONNECT] Client connected: user=ahmet
[CONNECT] Client connected: user=ela
[CONNECT] Client connected: user=cemal
[COMMAND] cemal joined room 'gtu'
[COMMAND] ahmet joined room 'gtu'
[COMMAND] ela joined room 'gtu'
[DISCONNECT] Client ela disconnected.
[CONNECT] Client connected: user=ela
[COMMAND] ela joined room 'gtu'
[COMMAND] ela left room.
[COMMAND] ela joined room 'gtu'
[COMMAND] ahmet sent whisper to cemal
[COMMAND] ahmet sent file to cemal
[FILE-TRANSFER] Approved: ahmet -> cemal ('uzun.txt', 12192 bytes)
[FILE-RECEIVED] ahmet uploaded 'uzun.txt' for cemal (12192 bytes) - Queue: 1/5
```

```
cbolat@DESKTOP-2J752DH:/mnt/c/Users/pc/Desktop/Cemal Final/System-Programming/F
NAL$ ./chatclient 127.0.0.1 8080
Enter username: ela
[SUCCESS] Username 'ela' is valid.
> /join gtu
[SUCCESS] You joined the room 'gtu'
> /exit
[INFO]: Disconnected. Goodbye!

cbolat@DESKTOP-2J752DH:/mnt/c/Users/pc/Desktop/Cemal Final/System-Programming/F
NAL$ ./chatclient 127.0.0.1 8080
Enter username: ahmet
[ERROR] Username already taken. Please choose another.
Enter username: ela
[SUCCESS] Username 'ela' is valid.
> /join gtu
[SUCCESS] You joined the room 'gtu'
> /leave
[SUCCESS] Left the room.
> /join gtu
[SUCCESS] You joined the room 'gtu'
> 
```

Figure 2: Three clients joined and exchanged messages successfully.



```
2025-05-31 22:29:16 - [LOGIN] user 'ahmet' connected from 127.0.0.1:50430
2025-05-31 22:29:17 - [LOGIN] user 'ela' connected from 127.0.0.1:50414
2025-05-31 22:29:19 - [LOGIN] user 'cemal' connected from 127.0.0.1:44290
2025-05-31 22:29:21 - [JOIN] user 'cemal' joined room 'gtu'
2025-05-31 22:29:25 - [JOIN] user 'ahmet' joined room 'gtu'
2025-05-31 22:29:28 - [JOIN] user 'ela' joined room 'gtu'
2025-05-31 22:29:30 - [DISCONNECT] user 'ela' exited the server.
2025-05-31 22:29:38 - [REJECTED] Duplicate username attempted: ahmet
2025-05-31 22:29:40 - [LOGIN] user 'ela' connected from 127.0.0.1:40960
2025-05-31 22:29:43 - [JOIN] user 'ela' joined room 'gtu'
2025-05-31 22:29:45 - [LEAVE] user 'ela' left room 'gtu'
2025-05-31 22:29:48 - [JOIN] user 'ela' joined room 'gtu'
2025-05-31 22:29:58 - [WHISPER] user 'ahmet' whispered to 'cemal': merhaba cemal
2025-05-31 22:30:13 - [FILE-REQUEST] ahmet initiated file transfer: cemal:uzun.txt:12192
2025-05-31 22:30:13 - [FILE-APPROVED] File transfer 'uzun.txt' from ahmet to cemal (12192 bytes) approved
2025-05-31 22:30:13 - [FILE-QUEUED] File 'uzun.txt' from ahmet to cemal queued successfully. Queue: 1/5
2025-05-31 22:30:13 - [SEND FILE] 'uzun.txt' sent from ahmet to cemal
2025-05-31 22:31:26 - [DISCONNECT] user 'cemal' lost connection. Cleaned up resources.
2025-05-31 22:31:27 - [DISCONNECT] user 'ahmet' lost connection. Cleaned up resources.
2025-05-31 22:31:28 - [DISCONNECT] user 'ela' lost connection. Cleaned up resources.
2025-05-31 22:31:30 - [SHUTDOWN] Server is shutting down. Disconnecting 0 clients and saving logs.
```

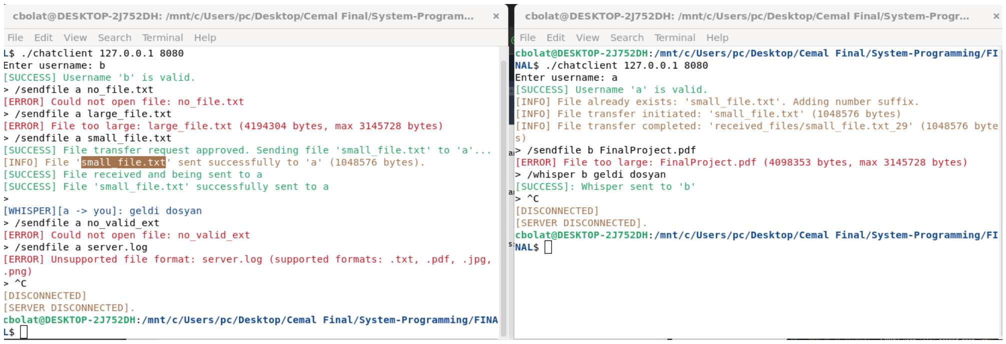
Figure 3: Server log file showing orderly connection, messaging, and disconnection.

File Transfer Error Handling

Test Description: This test case covers various error scenarios during file transfer, including invalid file extensions, files exceeding the 3MB limit, and duplicate filenames.

Expected Behavior: All validation checks (extension, size, and filename conflicts) should be handled on the client side. The server should only receive valid files and store them in a designated directory.

Observed Behavior: As expected, all invalid file uploads were blocked by the client before transmission. No error logs related to extension or size were recorded on the server. For successfully transferred files, the server saved each file under the `received_files` directory. If a file with the same name already existed, the new file was renamed automatically with an incrementing suffix (e.g., `file.txt`, `file_1.txt`, `file_2.txt`).


cbolat@DESKTOP-2J752DH: /mnt/c/Users/pc/Desktop/Cemal Final/System-Program...
File Edit View Search Terminal Help

```
L$ ./chatclient 127.0.0.1 8080
Enter username: b
[SUCCESS] Username 'b' is valid.
> /sendfile a no_file.txt
[ERROR] Could not open file: no_file.txt
> /sendfile a large_file.txt
[ERROR] File too large: large_file.txt (4194304 bytes, max 3145728 bytes)
> /sendfile a small_file.txt
[SUCCESS] File transfer request approved. Sending file 'small_file.txt' to 'a'...
[INFO] File 'small_file.txt' sent successfully to 'a' (1048576 bytes).
[SUCCESS] File received and being sent to a
[SUCCESS] File 'small_file.txt' successfully sent to a
>
[WHISPER][a -> you]: geldi dosyan
> /sendfile a no_valid_ext
[ERROR] Could not open file: no_valid_ext
> /sendfile a server.log
[ERROR] Unsupported file format: server.log (supported formats: .txt, .pdf, .jpg,
.png)
> ^C
[DISCONNECTED]
[SERVER DISCONNECTED].
cbolat@DESKTOP-2J752DH:/mnt/c/Users/pc/Desktop/Cemal Final/System-Programming/FINA
L$
```

cbolat@DESKTOP-2J752DH: /mnt/c/Users/pc/Desktop/Cemal Final/System-Progr...
File Edit View Search Terminal Help

```
cbolat@DESKTOP-2J752DH:/mnt/c/Users/pc/Desktop/Cemal Final/System-Programming/FI
NAL$ ./chatclient 127.0.0.1 8080
Enter username: a
[SUCCESS] Username 'a' is valid.
[INFO] File already exists: 'small_file.txt'. Adding number suffix.
[INFO] File transfer initiated: 'small_file.txt' (1048576 bytes)
[INFO] File transfer completed: 'received_files/small_file.txt_29' (1048576 byte
s)
> /sendfile b FinalProject.pdf
[ERROR] File too large: FinalProject.pdf (4098353 bytes, max 3145728 bytes)
> /whisper b geldi dosyan
[SUCCESS]: Whisper sent to 'b'
> ^C
[DISCONNECTED]
[SERVER DISCONNECTED].
cbolat@DESKTOP-2J752DH:/mnt/c/Users/pc/Desktop/Cemal Final/System-Programming/FI
NAL$
```

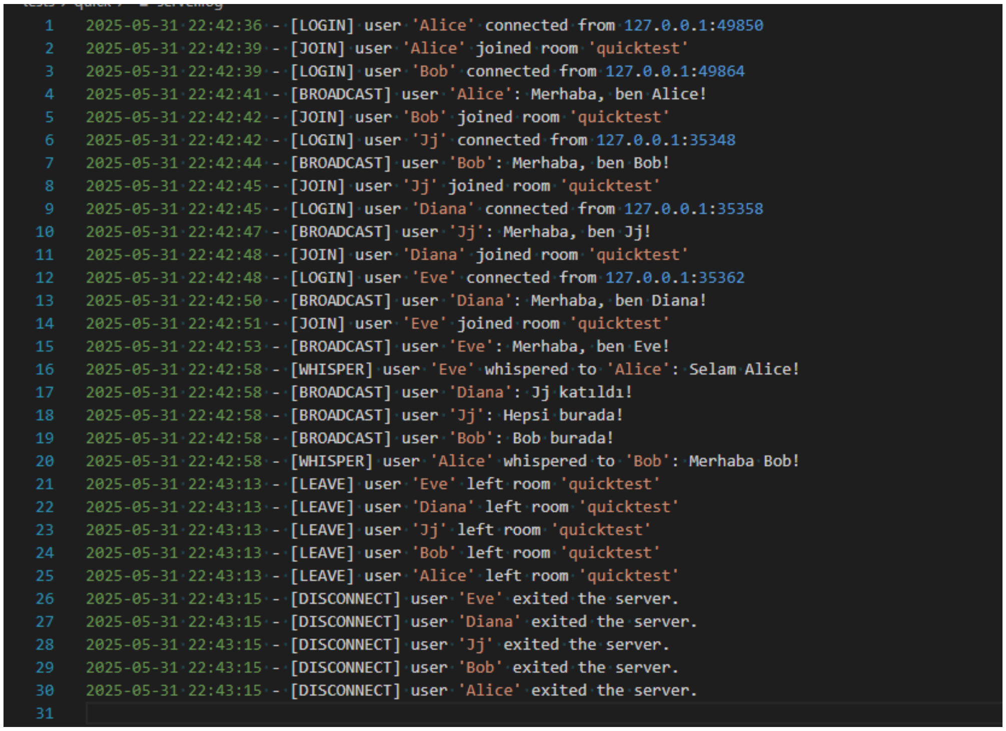
Figure 4: Client-side rejection of invalid file extension.

Quick Functional Test with 5 Clients

Test Description: This script performs an automated functional test of the server using 5 clients. Each client joins a shared room, sends broadcast and whisper messages, and then leaves the room in sequence. The goal is to validate core chat functionalities under realistic timing and concurrency.

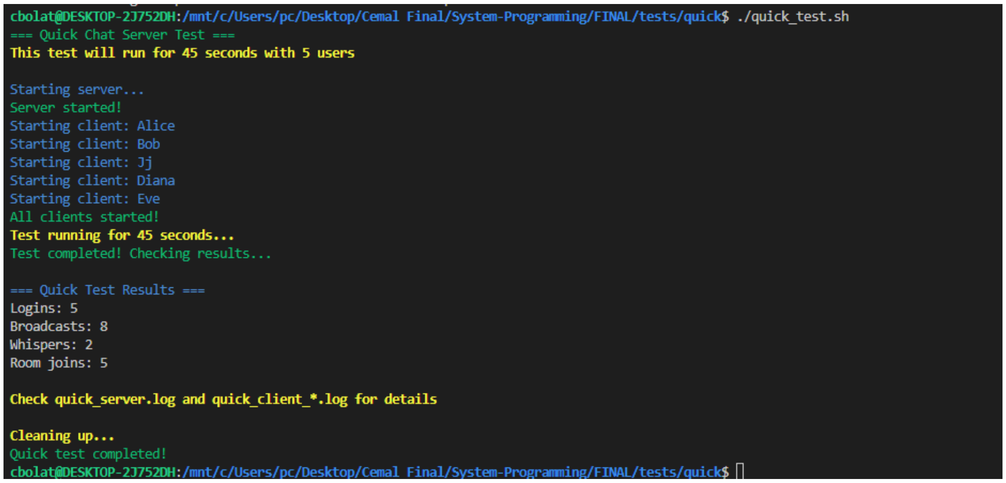
Expected Behavior: All clients should connect successfully, join the room, send and receive messages correctly, and leave gracefully without any crashes or synchronization issues. The server should log all actions with proper order and timestamps.

Observed Behavior: All clients were successfully initialized and performed their actions as expected. Synchronization using a temporary lock file ensured orderly message exchanges after all clients had joined. The server and each client generated separate log files which confirmed the correctness of joins, broadcasts, whispers, and exits.



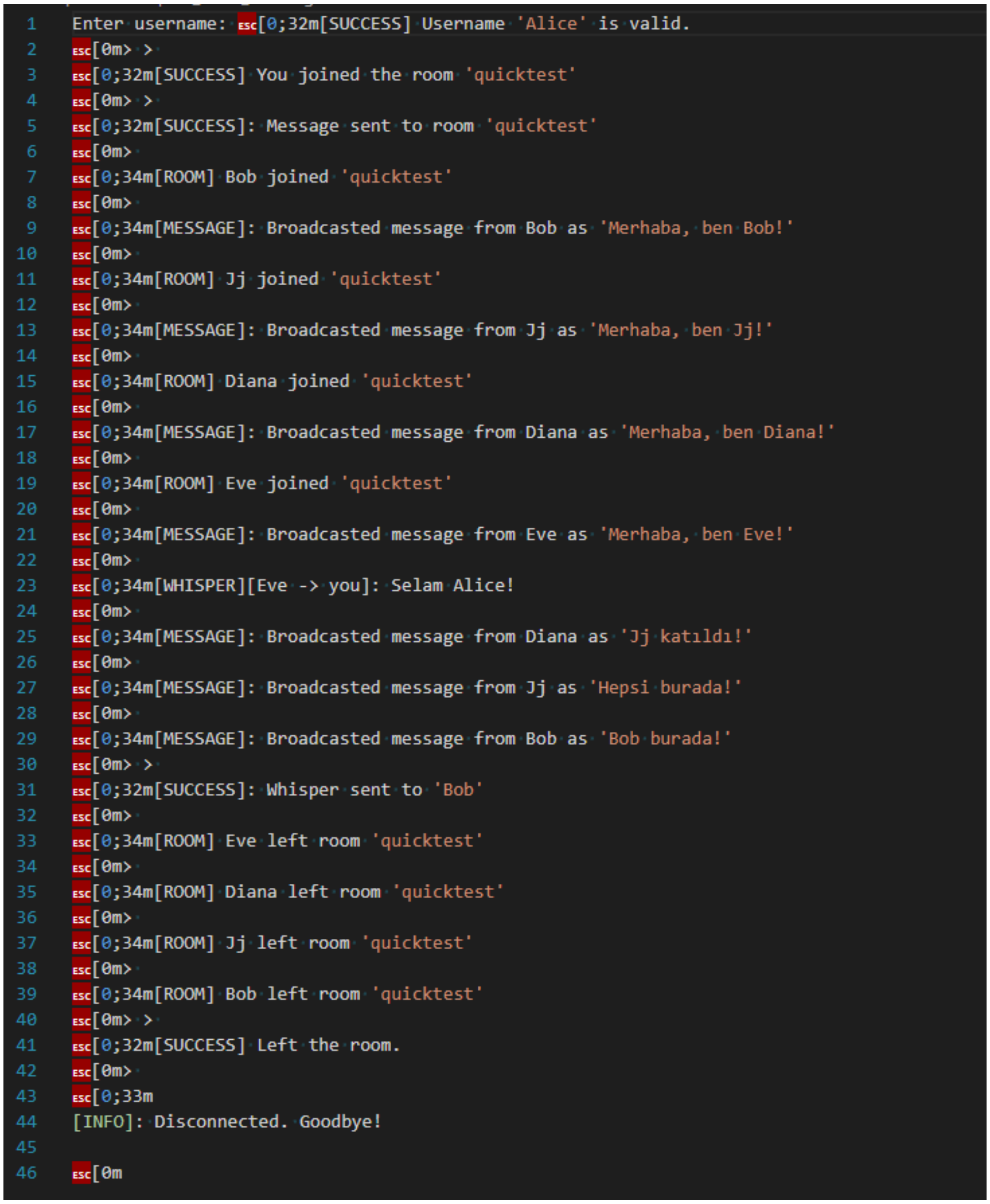
```
2025-05-31 22:42:36 - [LOGIN] user 'Alice' connected from 127.0.0.1:49850
2025-05-31 22:42:39 - [JOIN] user 'Alice' joined room 'quicktest'
2025-05-31 22:42:39 - [LOGIN] user 'Bob' connected from 127.0.0.1:49864
2025-05-31 22:42:41 - [BROADCAST] user 'Alice': Merhaba, ben Alice!
2025-05-31 22:42:42 - [JOIN] user 'Bob' joined room 'quicktest'
2025-05-31 22:42:42 - [LOGIN] user 'Jj' connected from 127.0.0.1:35348
2025-05-31 22:42:44 - [BROADCAST] user 'Bob': Merhaba, ben Bob!
2025-05-31 22:42:45 - [JOIN] user 'Jj' joined room 'quicktest'
2025-05-31 22:42:45 - [LOGIN] user 'Diana' connected from 127.0.0.1:35358
2025-05-31 22:42:47 - [BROADCAST] user 'Jj': Merhaba, ben Jj!
2025-05-31 22:42:48 - [JOIN] user 'Diana' joined room 'quicktest'
2025-05-31 22:42:48 - [LOGIN] user 'Eve' connected from 127.0.0.1:35362
2025-05-31 22:42:50 - [BROADCAST] user 'Diana': Merhaba, ben Diana!
2025-05-31 22:42:51 - [JOIN] user 'Eve' joined room 'quicktest'
2025-05-31 22:42:53 - [BROADCAST] user 'Eve': Merhaba, ben Eve!
2025-05-31 22:42:58 - [WHISPER] user 'Eve' whispered to 'Alice': Selam Alice!
2025-05-31 22:42:58 - [BROADCAST] user 'Diana': Jj katıldı!
2025-05-31 22:42:58 - [BROADCAST] user 'Jj': Hepsi burada!
2025-05-31 22:42:58 - [BROADCAST] user 'Bob': Bob burada!
2025-05-31 22:42:58 - [WHISPER] user 'Alice' whispered to 'Bob': Merhaba Bob!
2025-05-31 22:43:13 - [LEAVE] user 'Eve' left room 'quicktest'
2025-05-31 22:43:13 - [LEAVE] user 'Diana' left room 'quicktest'
2025-05-31 22:43:13 - [LEAVE] user 'Jj' left room 'quicktest'
2025-05-31 22:43:13 - [LEAVE] user 'Bob' left room 'quicktest'
2025-05-31 22:43:13 - [LEAVE] user 'Alice' left room 'quicktest'
2025-05-31 22:43:15 - [DISCONNECT] user 'Eve' exited the server.
2025-05-31 22:43:15 - [DISCONNECT] user 'Diana' exited the server.
2025-05-31 22:43:15 - [DISCONNECT] user 'Jj' exited the server.
2025-05-31 22:43:15 - [DISCONNECT] user 'Bob' exited the server.
2025-05-31 22:43:15 - [DISCONNECT] user 'Alice' exited the server.
```

Figure 5: Server log output during the quick functional test.



```
cbolat@DESKTOP-2J752DH:/mnt/c/Users/pc/Desktop/Cemal Final/System-Programming/FINAL/tests/quick$ ./quick_test.sh
=== Quick Chat Server Test ===
This test will run for 45 seconds with 5 users

Starting server...
Server started!
Starting client: Alice
Starting client: Bob
Starting client: Jj
Starting client: Diana
Starting client: Eve
All clients started!
Test running for 45 seconds...
Test completed! Checking results...

=== Quick Test Results ===
Logins: 5
Broadcasts: 8
Whispers: 2
Room joins: 5

Check quick_server.log and quick_client_*.log for details

Cleaning up...
Quick test completed!
cbolat@DESKTOP-2J752DH:/mnt/c/Users/pc/Desktop/Cemal Final/System-Programming/FINAL/tests/quick$
```

Figure 6: Shell programs terminal



```
Enter username: ESC[0;32m[SUCCESS] Username 'Alice' is valid.
ESC[0m> > 
ESC[0;32m[SUCCESS] You joined the room 'quicktest'
ESC[0m> > 
ESC[0;32m[SUCCESS]: Message sent to room 'quicktest'
ESC[0m> 
ESC[0;34m[ROOM] Bob joined 'quicktest'
ESC[0m> 
ESC[0;34m[MESSAGE]: Broadcasted message from Bob as 'Merhaba, ben Bob!'
ESC[0m> 
ESC[0;34m[ROOM] Jj joined 'quicktest'
ESC[0m> 
ESC[0;34m[MESSAGE]: Broadcasted message from Jj as 'Merhaba, ben Jj!'
ESC[0m> 
ESC[0;34m[ROOM] Diana joined 'quicktest'
ESC[0m> 
ESC[0;34m[MESSAGE]: Broadcasted message from Diana as 'Merhaba, ben Diana!'
ESC[0m> 
ESC[0;34m[ROOM] Eve joined 'quicktest'
ESC[0m> 
ESC[0;34m[MESSAGE]: Broadcasted message from Eve as 'Merhaba, ben Eve!'
ESC[0m> 
ESC[0;34m[WHISPER][Eve -> you]: Selam Alice!
ESC[0m> 
ESC[0;34m[MESSAGE]: Broadcasted message from Diana as 'Jj katıldı!'
ESC[0m> 
ESC[0;34m[MESSAGE]: Broadcasted message from Jj as 'Hepsi burada!'
ESC[0m> 
ESC[0;34m[MESSAGE]: Broadcasted message from Bob as 'Bob burada!'
ESC[0m> > 
ESC[0;32m[SUCCESS]: Whisper sent to 'Bob'
ESC[0m> 
ESC[0;34m[ROOM] Eve left room 'quicktest'
ESC[0m> 
ESC[0;34m[ROOM] Diana left room 'quicktest'
ESC[0m> 
ESC[0;34m[ROOM] Jj left room 'quicktest'
ESC[0m> 
ESC[0;34m[ROOM] Bob left room 'quicktest'
ESC[0m> > 
ESC[0;32m[SUCCESS] Left the room.
ESC[0m> 
ESC[0;33m
[INFO]: Disconnected. Goodbye!

ESC[0m
```

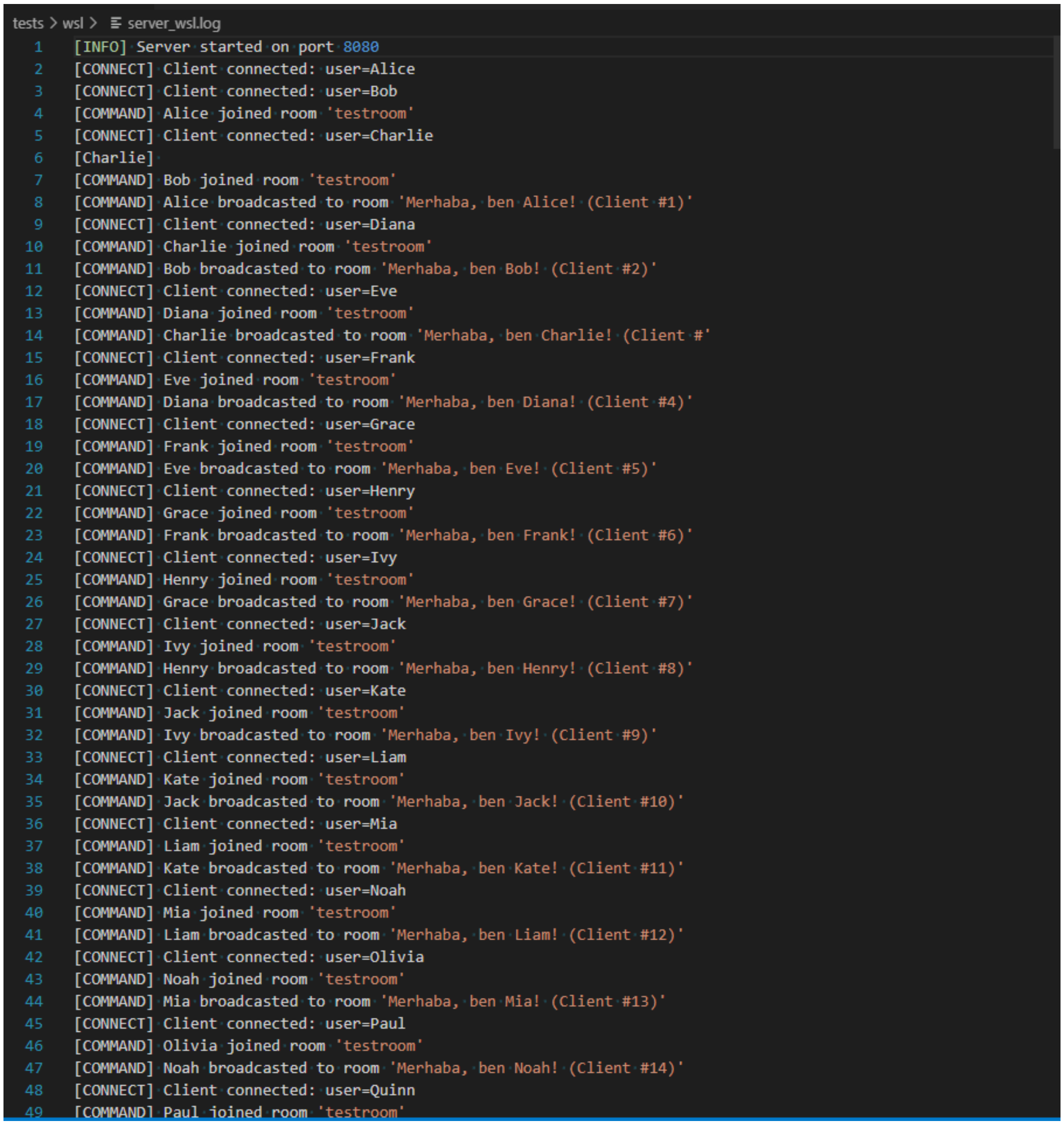
Figure 7: A clients log file for instance

WSL-Based Large Scale Test with 30 Clients

Test Description: This stress test launches 30 clients using a Bash script optimized for the Windows Subsystem for Linux (WSL). Each client connects to the chat server, joins a common room, sends a set of broadcast and whisper messages, then leaves the room. Clients are started with slight delays to simulate realistic staggered connections and interactions.

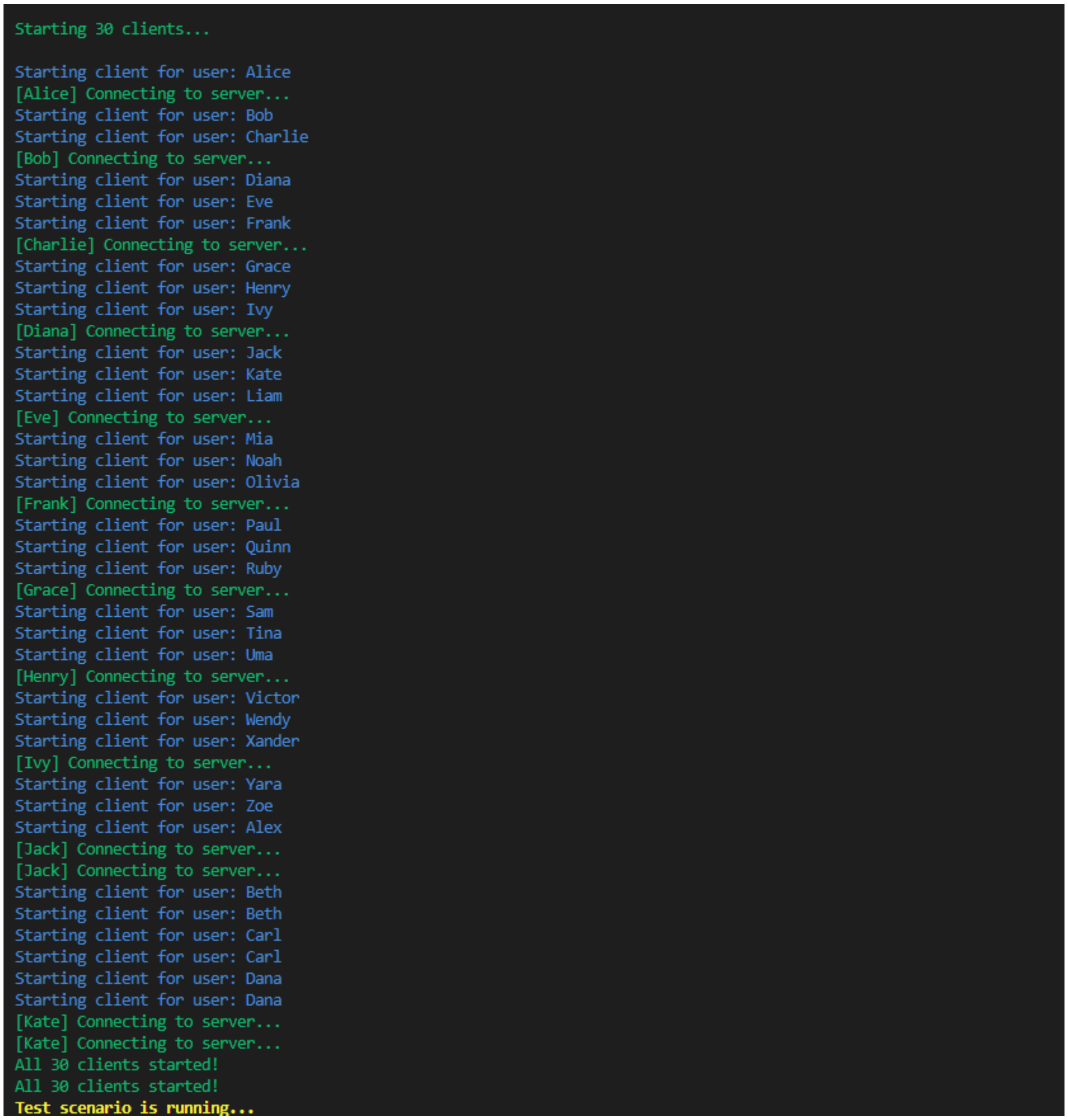
Expected Behavior: The server should handle all 30 clients concurrently without crashes or message delivery failures. All room joins, broadcasts, whispers, and exits must be logged accurately and in proper order.

Observed Behavior: The system demonstrated strong stability under high concurrency. All clients were able to connect, send and receive messages, and exit as expected. Whisper and broadcast messages were delivered correctly. Server output was logged in `server_wsl.log`, while each client wrote to its own `client_<username>.log` file.



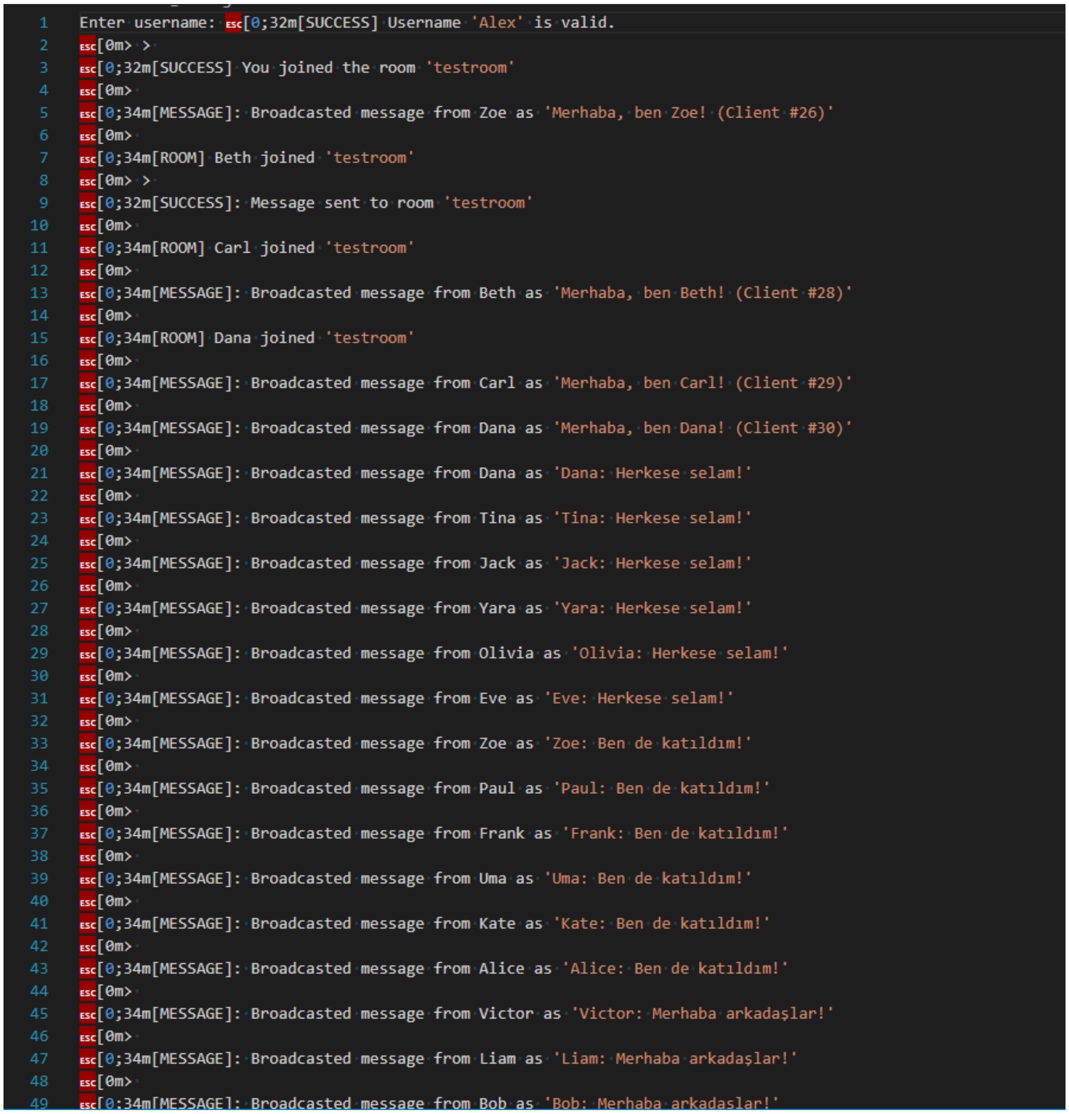
```
[INFO] Server started on port 8080
[CONNECT] Client connected: user=Alice
[CONNECT] Client connected: user=Bob
[COMMAND] Alice joined room 'testroom'
[CONNECT] Client connected: user=Charlie
[Charlie]
[COMMAND] Bob joined room 'testroom'
[COMMAND] Alice broadcasted to room 'Merhaba, ben Alice! (Client #1)'
[CONNECT] Client connected: user=Diana
[COMMAND] Charlie joined room 'testroom'
[COMMAND] Bob broadcasted to room 'Merhaba, ben Bob! (Client #2)'
[CONNECT] Client connected: user=Eve
[COMMAND] Diana joined room 'testroom'
[COMMAND] Charlie broadcasted to room 'Merhaba, ben Charlie! (Client #'
[CONNECT] Client connected: user=Frank
[COMMAND] Eve joined room 'testroom'
[COMMAND] Diana broadcasted to room 'Merhaba, ben Diana! (Client #4)'
[CONNECT] Client connected: user=Grace
[COMMAND] Frank joined room 'testroom'
[COMMAND] Eve broadcasted to room 'Merhaba, ben Eve! (Client #5)'
[CONNECT] Client connected: user=Henry
[COMMAND] Grace joined room 'testroom'
[COMMAND] Frank broadcasted to room 'Merhaba, ben Frank! (Client #6)'
[CONNECT] Client connected: user=Ivy
[COMMAND] Henry joined room 'testroom'
[COMMAND] Grace broadcasted to room 'Merhaba, ben Grace! (Client #7)'
[CONNECT] Client connected: user=Jack
[COMMAND] Ivy joined room 'testroom'
[COMMAND] Henry broadcasted to room 'Merhaba, ben Henry! (Client #8)'
[CONNECT] Client connected: user=Kate
[COMMAND] Jack joined room 'testroom'
[COMMAND] Ivy broadcasted to room 'Merhaba, ben Ivy! (Client #9)'
[CONNECT] Client connected: user=Liam
[COMMAND] Kate joined room 'testroom'
[COMMAND] Jack broadcasted to room 'Merhaba, ben Jack! (Client #10)'
[CONNECT] Client connected: user=Mia
[COMMAND] Liam joined room 'testroom'
[COMMAND] Kate broadcasted to room 'Merhaba, ben Kate! (Client #11)'
[CONNECT] Client connected: user=Noah
[COMMAND] Mia joined room 'testroom'
[COMMAND] Liam broadcasted to room 'Merhaba, ben Liam! (Client #12)'
[CONNECT] Client connected: user=Olivia
[COMMAND] Noah joined room 'testroom'
[COMMAND] Mia broadcasted to room 'Merhaba, ben Mia! (Client #13)'
[CONNECT] Client connected: user=Paul
[COMMAND] Olivia joined room 'testroom'
[COMMAND] Noah broadcasted to room 'Merhaba, ben Noah! (Client #14)'
[CONNECT] Client connected: user=Quinn
[COMMAND] Paul joined room 'testroom'
```

Figure 8: Server log showing activity from 30 clients including joins, broadcasts, and whispers.



```
Starting 30 clients...

Starting client for user: Alice
[Alice] Connecting to server...
Starting client for user: Bob
Starting client for user: Charlie
[Bob] Connecting to server...
Starting client for user: Diana
Starting client for user: Eve
Starting client for user: Frank
[Charlie] Connecting to server...
Starting client for user: Grace
Starting client for user: Henry
Starting client for user: Ivy
[Diana] Connecting to server...
Starting client for user: Jack
Starting client for user: Kate
Starting client for user: Liam
[Eve] Connecting to server...
Starting client for user: Mia
Starting client for user: Noah
Starting client for user: Olivia
[Frank] Connecting to server...
Starting client for user: Paul
Starting client for user: Quinn
Starting client for user: Ruby
[Grace] Connecting to server...
Starting client for user: Sam
Starting client for user: Tina
Starting client for user: Uma
[Henry] Connecting to server...
Starting client for user: Victor
Starting client for user: Wendy
Starting client for user: Xander
[Ivy] Connecting to server...
Starting client for user: Yara
Starting client for user: Zoe
Starting client for user: Alex
[Jack] Connecting to server...
[Jack] Connecting to server...
Starting client for user: Beth
Starting client for user: Beth
Starting client for user: Carl
Starting client for user: Carl
Starting client for user: Dana
Starting client for user: Dana
[Kate] Connecting to server...
[Kate] Connecting to server...
All 30 clients started!
All 30 clients started!
Test scenario is running...
```

Figure 9: Shell programs terminal



```
Enter username: esc[0;32m[SUCCESS] Username 'Alex' is valid.
esc[0m> >
esc[0;32m[SUCCESS] You joined the room 'testroom'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Zoe as 'Merhaba, ben Zoe! (Client #26)'
esc[0m> 
esc[0;34m[ROOM] Beth joined 'testroom'
esc[0m> >
esc[0;32m[SUCCESS]: Message sent to room 'testroom'
esc[0m> 
esc[0;34m[ROOM] Carl joined 'testroom'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Beth as 'Merhaba, ben Beth! (Client #28)'
esc[0m> 
esc[0;34m[ROOM] Dana joined 'testroom'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Carl as 'Merhaba, ben Carl! (Client #29)'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Dana as 'Merhaba, ben Dana! (Client #30)'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Dana as 'Dana: Herkese selam!'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Tina as 'Tina: Herkese selam!'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Jack as 'Jack: Herkese selam!'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Yara as 'Yara: Herkese selam!'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Olivia as 'Olivia: Herkese selam!'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Eve as 'Eve: Herkese selam!'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Zoe as 'Zoe: Ben de katıldım!'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Paul as 'Paul: Ben de katıldım!'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Frank as 'Frank: Ben de katıldım!'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Uma as 'Uma: Ben de katıldım!'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Kate as 'Kate: Ben de katıldım!'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Alice as 'Alice: Ben de katıldım!'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Victor as 'Victor: Merhaba arkadaşlar!'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Liam as 'Liam: Merhaba arkadaşlar!'
esc[0m> 
esc[0;34m[MESSAGE]: Broadcasted message from Bob as 'Bob: Merhaba arkadaşlar!'
```

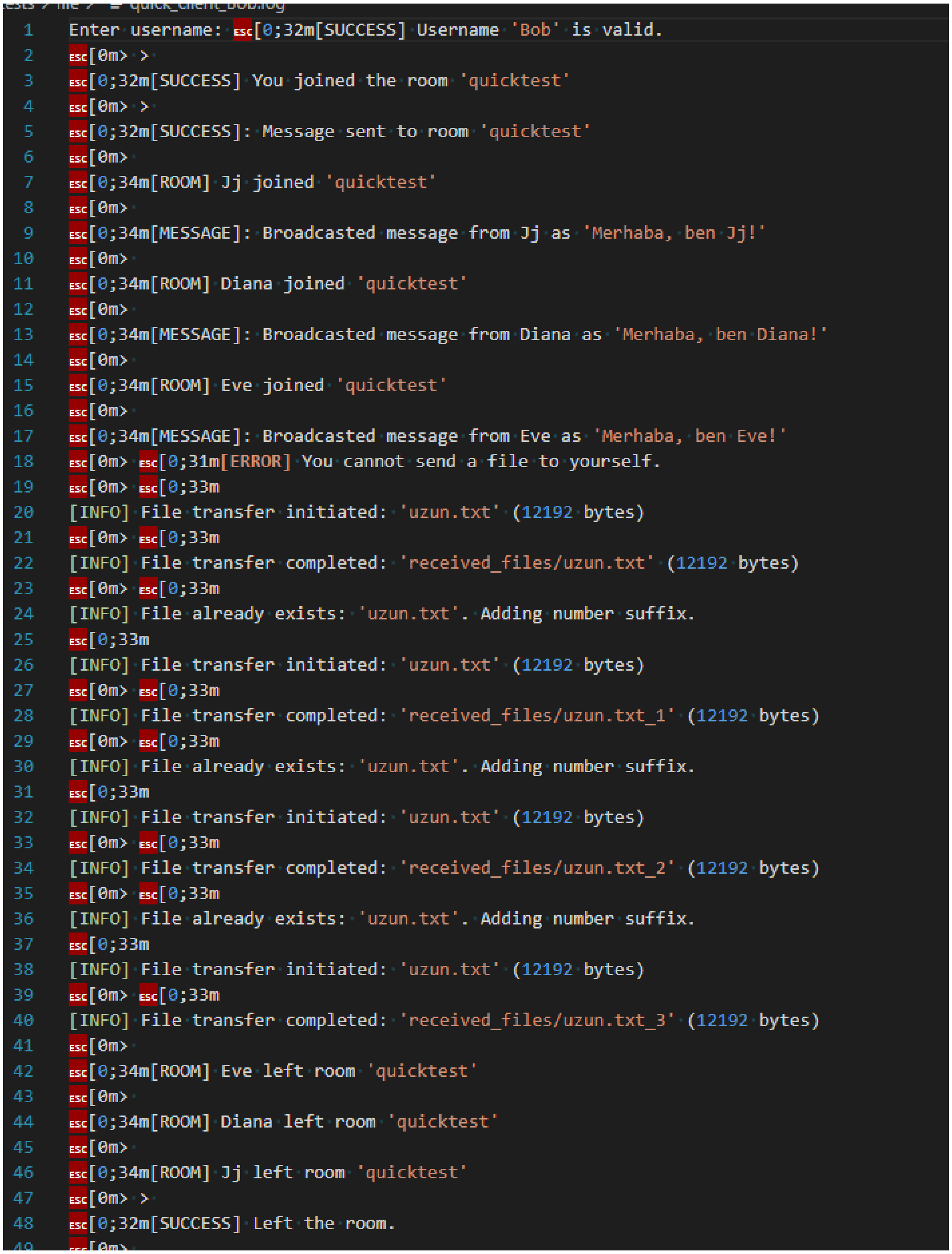
Figure 10: A sample client log demonstrating full participation in the test scenario.

File Transfer Stress Test with Multiple Clients

Test Description: This scenario evaluates the server's performance and consistency under a heavy load of simultaneous file transfers. Multiple clients attempt to send files concurrently, triggering file validation, queue management, and delivery mechanisms.

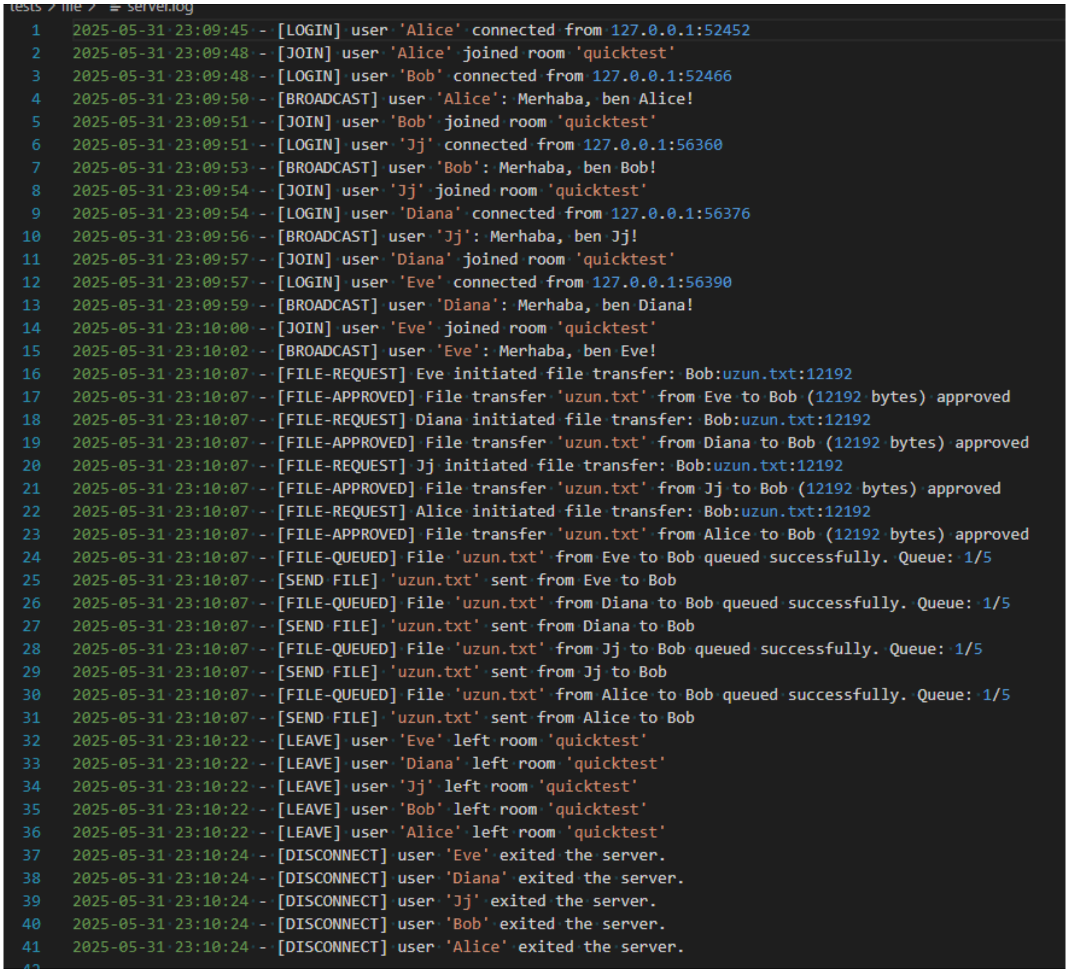
Expected Behavior: The server should queue incoming file transfer requests, validate file sizes and extensions, and deliver files in correct order without data corruption or thread conflicts. All actions should be logged clearly and accurately.

Observed Behavior: The server successfully handled concurrent file transfers. Each file was received, verified, and routed to the intended recipient without any race conditions or deadlocks. Filename collisions were resolved by appending suffixes, and no file loss or duplication was observed. Both client-side and server-side logs confirmed the integrity and order of transfers.



```
Enter username: ESC[0;32m[SUCCESS] Username 'Bob' is valid.
ESC[0m> > 
ESC[0;32m[SUCCESS] You joined the room 'quicktest'
ESC[0m> > 
ESC[0;32m[SUCCESS]: Message sent to room 'quicktest'
ESC[0m> 
ESC[0;34m[ROOM] Jj joined 'quicktest'
ESC[0m> 
ESC[0;34m[MESSAGE]: Broadcasted message from Jj as 'Merhaba, ben Jj!'
ESC[0m> 
ESC[0;34m[ROOM] Diana joined 'quicktest'
ESC[0m> 
ESC[0;34m[MESSAGE]: Broadcasted message from Diana as 'Merhaba, ben Diana!'
ESC[0m> 
ESC[0;34m[ROOM] Eve joined 'quicktest'
ESC[0m> 
ESC[0;34m[MESSAGE]: Broadcasted message from Eve as 'Merhaba, ben Eve!'
ESC[0m> ESC[0;31m[ERROR] You cannot send a file to yourself.
ESC[0m> ESC[0;33m
[INFO] File transfer initiated: 'uzun.txt' (12192 bytes)
ESC[0m> ESC[0;33m
[INFO] File transfer completed: 'received_files/uzun.txt' (12192 bytes)
ESC[0m> ESC[0;33m
[INFO] File already exists: 'uzun.txt'. Adding number suffix.
ESC[0;33m
[INFO] File transfer initiated: 'uzun.txt' (12192 bytes)
ESC[0m> ESC[0;33m
[INFO] File transfer completed: 'received_files/uzun.txt_1' (12192 bytes)
ESC[0m> ESC[0;33m
[INFO] File already exists: 'uzun.txt'. Adding number suffix.
ESC[0;33m
[INFO] File transfer initiated: 'uzun.txt' (12192 bytes)
ESC[0m> ESC[0;33m
[INFO] File transfer completed: 'received_files/uzun.txt_2' (12192 bytes)
ESC[0m> ESC[0;33m
[INFO] File already exists: 'uzun.txt'. Adding number suffix.
ESC[0;33m
[INFO] File transfer initiated: 'uzun.txt' (12192 bytes)
ESC[0m> ESC[0;33m
[INFO] File transfer completed: 'received_files/uzun.txt_3' (12192 bytes)
ESC[0m> 
ESC[0;34m[ROOM] Eve left room 'quicktest'
ESC[0m> 
ESC[0;34m[ROOM] Diana left room 'quicktest'
ESC[0m> 
ESC[0;34m[ROOM] Jj left room 'quicktest'
ESC[0m> > 
ESC[0;32m[SUCCESS] Left the room.
ESC[0m> 
```

Figure 11: Client log showing successful upload requests and transfer acknowledgements.



```
2025-05-31 23:09:45 - [LOGIN] user 'Alice' connected from 127.0.0.1:52452
2025-05-31 23:09:48 - [JOIN] user 'Alice' joined room 'quicktest'
2025-05-31 23:09:48 - [LOGIN] user 'Bob' connected from 127.0.0.1:52466
2025-05-31 23:09:50 - [BROADCAST] user 'Alice': Merhaba, ben Alice!
2025-05-31 23:09:51 - [JOIN] user 'Bob' joined room 'quicktest'
2025-05-31 23:09:51 - [LOGIN] user 'Jj' connected from 127.0.0.1:56360
2025-05-31 23:09:53 - [BROADCAST] user 'Bob': Merhaba, ben Bob!
2025-05-31 23:09:54 - [JOIN] user 'Jj' joined room 'quicktest'
2025-05-31 23:09:54 - [LOGIN] user 'Diana' connected from 127.0.0.1:56376
2025-05-31 23:09:56 - [BROADCAST] user 'Jj': Merhaba, ben Jj!
2025-05-31 23:09:57 - [JOIN] user 'Diana' joined room 'quicktest'
2025-05-31 23:09:57 - [LOGIN] user 'Eve' connected from 127.0.0.1:56390
2025-05-31 23:09:59 - [BROADCAST] user 'Diana': Merhaba, ben Diana!
2025-05-31 23:10:00 - [JOIN] user 'Eve' joined room 'quicktest'
2025-05-31 23:10:02 - [BROADCAST] user 'Eve': Merhaba, ben Eve!
2025-05-31 23:10:07 - [FILE-REQUEST] Eve initiated file transfer: Bob:uzun.txt:12192
2025-05-31 23:10:07 - [FILE-APPROVED] File transfer 'uzun.txt' from Eve to Bob (12192 bytes) approved
2025-05-31 23:10:07 - [FILE-REQUEST] Diana initiated file transfer: Bob:uzun.txt:12192
2025-05-31 23:10:07 - [FILE-APPROVED] File transfer 'uzun.txt' from Diana to Bob (12192 bytes) approved
2025-05-31 23:10:07 - [FILE-REQUEST] Jj initiated file transfer: Bob:uzun.txt:12192
2025-05-31 23:10:07 - [FILE-APPROVED] File transfer 'uzun.txt' from Jj to Bob (12192 bytes) approved
2025-05-31 23:10:07 - [FILE-REQUEST] Alice initiated file transfer: Bob:uzun.txt:12192
2025-05-31 23:10:07 - [FILE-APPROVED] File transfer 'uzun.txt' from Alice to Bob (12192 bytes) approved
2025-05-31 23:10:07 - [FILE-QUEUED] File 'uzun.txt' from Eve to Bob queued successfully. Queue: 1/5
2025-05-31 23:10:07 - [SEND FILE] 'uzun.txt' sent from Eve to Bob
2025-05-31 23:10:07 - [FILE-QUEUED] File 'uzun.txt' from Diana to Bob queued successfully. Queue: 1/5
2025-05-31 23:10:07 - [SEND FILE] 'uzun.txt' sent from Diana to Bob
2025-05-31 23:10:07 - [FILE-QUEUED] File 'uzun.txt' from Jj to Bob queued successfully. Queue: 1/5
2025-05-31 23:10:07 - [SEND FILE] 'uzun.txt' sent from Jj to Bob
2025-05-31 23:10:07 - [FILE-QUEUED] File 'uzun.txt' from Alice to Bob queued successfully. Queue: 1/5
2025-05-31 23:10:07 - [SEND FILE] 'uzun.txt' sent from Alice to Bob
2025-05-31 23:10:22 - [LEAVE] user 'Eve' left room 'quicktest'
2025-05-31 23:10:22 - [LEAVE] user 'Diana' left room 'quicktest'
2025-05-31 23:10:22 - [LEAVE] user 'Jj' left room 'quicktest'
2025-05-31 23:10:22 - [LEAVE] user 'Bob' left room 'quicktest'
2025-05-31 23:10:22 - [LEAVE] user 'Alice' left room 'quicktest'
2025-05-31 23:10:24 - [DISCONNECT] user 'Eve' exited the server.
2025-05-31 23:10:24 - [DISCONNECT] user 'Diana' exited the server.
2025-05-31 23:10:24 - [DISCONNECT] user 'Jj' exited the server.
2025-05-31 23:10:24 - [DISCONNECT] user 'Bob' exited the server.
2025-05-31 23:10:24 - [DISCONNECT] user 'Alice' exited the server.
```

Figure 12: Server log detailing file queue processing and delivery to receivers.

Conclusion

This project successfully demonstrates the design and implementation of a multi-threaded, TCP-based chat server capable of handling concurrent client connections, real-time messaging, and file transfers. The system was tested under various conditions including normal usage, invalid command handling, and stress scenarios involving up to 30 clients and simultaneous file transfers. Custom synchronization mechanisms, thread-safe logging, and atomic operations were integrated to ensure consistency, responsiveness, and reliability.

All core functionalities—such as room joining, broadcasting, whispering, and file transfer—performed correctly across diverse test cases. Error conditions were detected and handled gracefully, and server stability remained intact throughout the testing process. The modular thread structure and custom protocol ensured scalability and maintainability.

Potential Improvements

Although the project met all initial goals, several enhancements can be considered for future iterations:

- **Asynchronous File Delivery:** Currently, file delivery is synchronous and may delay message handling under high load. Moving to a fully asynchronous or non-blocking model could improve throughput.
- **GUI Client Interface:** A graphical client application would improve usability and accessibility for non-technical users.
- **Robust Timeout and Reconnection Handling:** Clients that disconnect unexpectedly currently rely on thread exit for cleanup. Implementing reconnection logic and timeout detection would improve resilience.
- **Persistent Message Storage:** Currently, all messages exist only in memory during runtime. Persistent chat history and file delivery records could be added via database integration.