CSE 344 SYSTEM PROGRAMMING FINAL PROJECT REPORT

STUDENT NAME : OĞUZ ALP KELEŞ

STUDENT NUMBER: 220104004834

Contents

Chat Server and Client Implementation Report

1. Introduction and Problem Definition

In this project, I developed a terminal-based chat application consisting of a multi-threaded server (chatserver.c) and a client interface (chatclient.c) using the C programming language and POSIX libraries. The system is designed to support multiple clients connecting simultaneously to the server, where they can participate in chat rooms, send private messages (whispers), and transfer files. The solution aims to address scalability, concurrency, and file transfer management in a distributed messaging environment.

2. Design Details

The project follows a client-server architecture using TCP sockets. The server listens on a specified port and accepts incoming connections. For each new connection, a separate thread is spawned to handle communication with that client, ensuring concurrency.

IPC Mechanism:
The primary form of inter-process communication is through TCP sockets. The server and clients exchange commands and data over the network, formatted as strings and binary packets. Commands such as '/join', '/leave', '/broadcast', '/whisper', and '/sendfile' are interpreted by the server to manage chat behavior.

Thread Model:
The server uses POSIX threads (pthread_create) to handle each client. Shared resources such as the global client list and file transfer slots are

protected with mutexes (pthread_mutex_t). A counting semaphore (sem_t file_transfer_sem) limits the number of concurrent file uploads to 5.

Note: In the client terminal, when connecting to the server ip address should be given as 127.0.0.1 which is the loop back ip address.

3. Key Functions and Their Responsibilities

main() (chatserver.c)

Initializes the server, sets up socket, binds to the port, initializes semaphore and signal handlers. Accepts new client connections and spawns a new thread using pthread_create for each client. Threads are detached using pthread_detach to avoid zombie threads.

handle_client(void *arg)

This is the main handler function for each connected client. It processes commands received from the client, such as /join, /leave, /whisper, /broadcast, and /sendfile. It also ensures thread-safe access to shared resources using mutexes and enforces file upload concurrency using a semaphore.

send_to_client(int sock, const char *message)

Sends a message string to a specific client using the send() system call. Used throughout the server to send feedback or chat messages to individual clients.

broadcast_room(const char *room, const char *sender, const char *message)

Sends a chat message to all clients that are currently in the same room, except for the sender. Used to implement the /broadcast functionality.

whisper_to_user(const char *target, const char *sender, const char *message)

Sends a private message from one user to another. Looks up the target username and sends the message if the user exists.

is_username_taken(const char *username)

Checks whether a given username is already in use by iterating over the global client list. Uses a mutex to ensure thread-safe access.

remove_client(int sock)

Removes a client from the global client list based on their socket. Called when a client disconnects.

log_action(const char *format, ...)

Logs formatted messages to a log file for auditing or debugging purposes. Used to record actions like file transfers, joins, and errors.

save_incoming_file(...) (chatclient.c)

Receives file metadata and content from the server and writes it to the local file system. Handles partial reads and ensures the file is saved in the correct directory.

main() (chatclient.c)

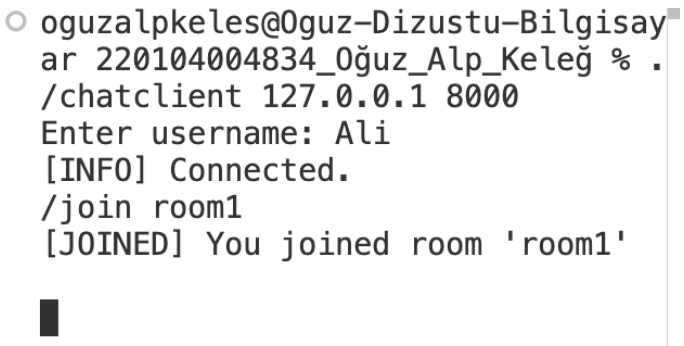
Connects the client to the server, submits the username, and enters a main loop to handle user input and server messages concurrently using select(). Also handles command parsing and sending.

4. TEST CASES AND OUTPUTS

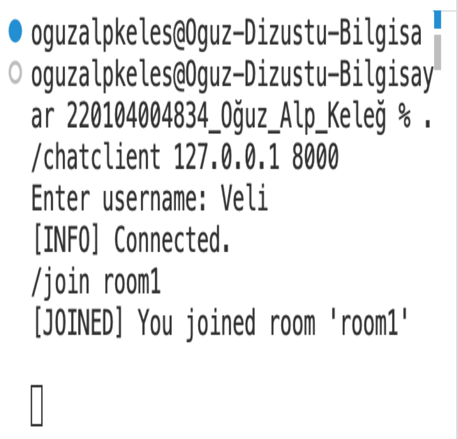
4.1 Test for Join & Broadcast & Whisper

We see that Ali, Veli, and Mehmet are in room 1. Hasan is in room2.

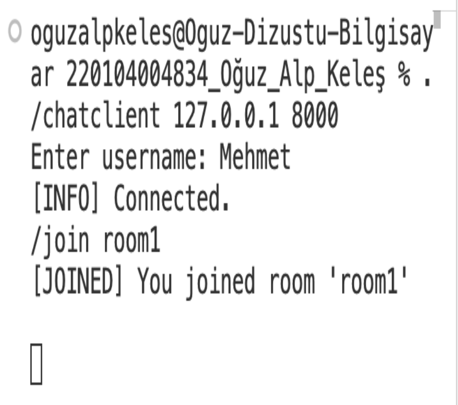
Expectation: When Ali broadcasts, his message will be displayed in Mehmets and Velis terminal whereas Hasan will not be able to see them.



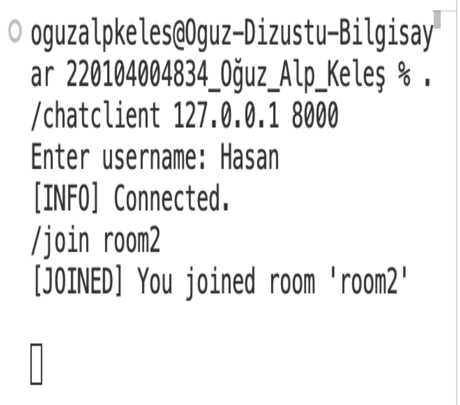
```
oguzalpkeles@Oguz-Dizustu-Bilgisay
ar 220104004834_Oğuz_Alp_Keleğ % .
/chatclient 127.0.0.1 8000
Enter username: Ali
[INFO] Connected.
/join room1
[JOINED] You joined room 'room1'
```



```
oguzalpkeles@Oguz-Dizustu-Bilgisa
oguzalpkeles@Oguz-Dizustu-Bilgisay
ar 220104004834_Oğuz_Alp_Keleğ % .
/chatclient 127.0.0.1 8000
Enter username: Veli
[INFO] Connected.
/join room1
[JOINED] You joined room 'room1'
```

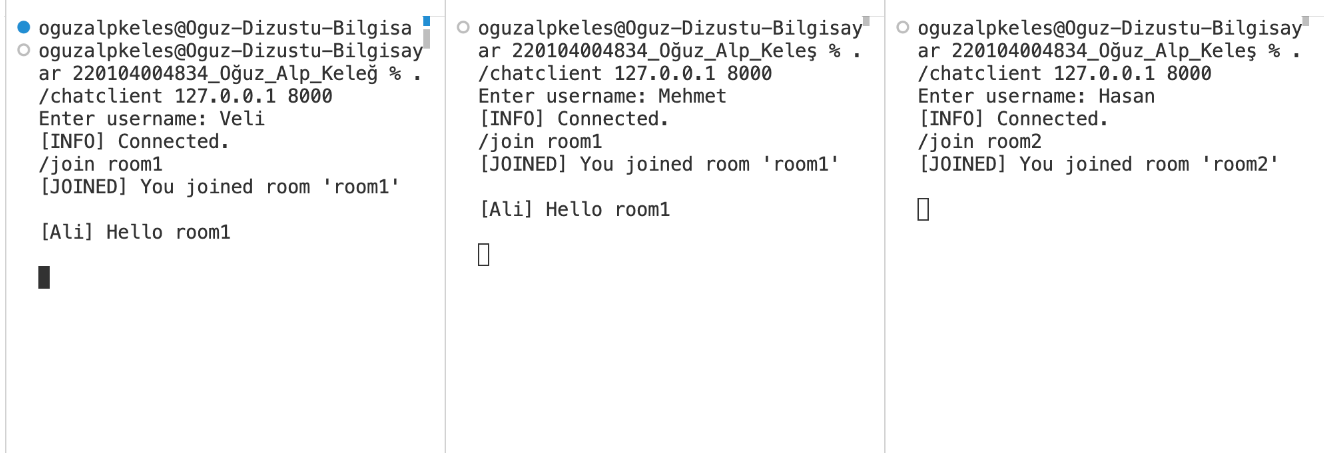


```
oguzalpkeles@Oguz-Dizustu-Bilgisay
ar 220104004834_Oğuz_Alp_Keleş % .
/chatclient 127.0.0.1 8000
Enter username: Mehmet
[INFO] Connected.
/join room1
[JOINED] You joined room 'room1'
```



```
oguzalpkeles@Oguz-Dizustu-Bilgisay
ar 220104004834_Oğuz_Alp_Keleş % .
/chatclient 127.0.0.1 8000
Enter username: Hasan
[INFO] Connected.
/join room2
[JOINED] You joined room 'room2'
```

Below you can see that what we expect is true.

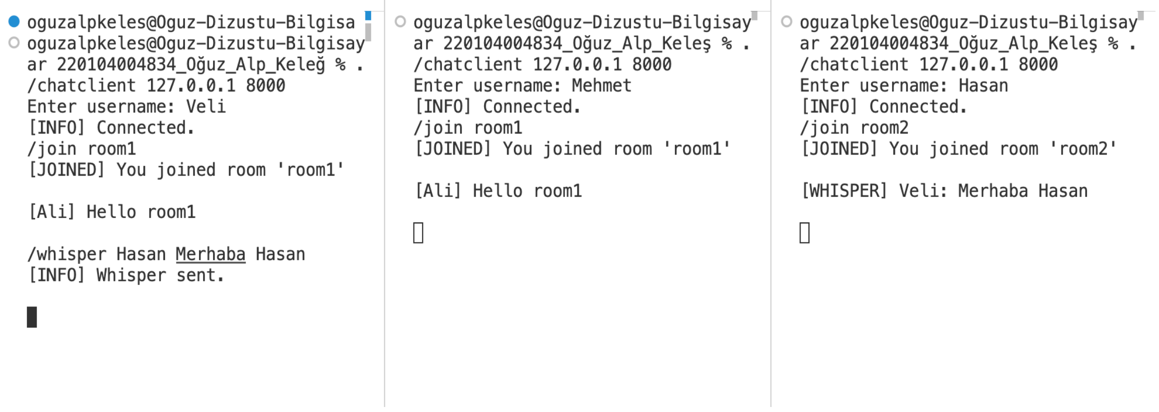


```
● oguzalpkeles@Oguz-Dizustu-Bilgisa
○ oguzalpkeles@Oguz-Dizustu-Bilgisay
  ar 220104004834_Oğuz_Alp_Keleğ % .
  /chatclient 127.0.0.1 8000
  Enter username: Veli
  [INFO] Connected.
  /join room1
  [JOINED] You joined room 'room1'

  [Ali] Hello room1
```

```
○ oguzalpkeles@Oguz-Dizustu-Bilgisay
  ar 220104004834_Oğuz_Alp_Keleş % .
  /chatclient 127.0.0.1 8000
  Enter username: Mehmet
  [INFO] Connected.
  /join room1
  [JOINED] You joined room 'room1'

  [Ali] Hello room1
```

```
○ oguzalpkeles@Oguz-Dizustu-Bilgisay
  ar 220104004834_Oğuz_Alp_Keleş % .
  /chatclient 127.0.0.1 8000
  Enter username: Hasan
  [INFO] Connected.
  /join room2
  [JOINED] You joined room 'room2'
```

Expectation: When Veli whispers to Hasan, only Hasan should see it and we can see that below.



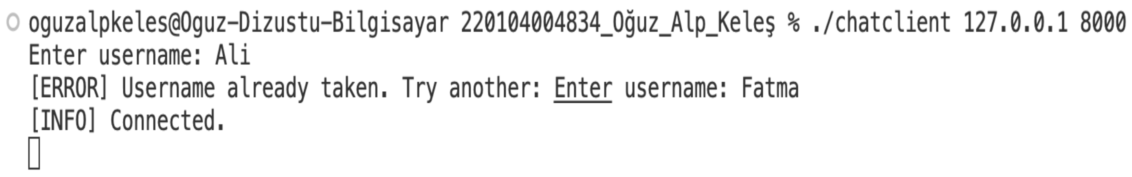
```
● oguzalpkeles@Oguz-Dizustu-Bilgisa
○ oguzalpkeles@Oguz-Dizustu-Bilgisay
  ar 220104004834_Oğuz_Alp_Keleğ % .
  /chatclient 127.0.0.1 8000
  Enter username: Veli
  [INFO] Connected.
  /join room1
  [JOINED] You joined room 'room1'

  [Ali] Hello room1

  /whisper Hasan Merhaba Hasan
  [INFO] Whisper sent.
```

```
○ oguzalpkeles@Oguz-Dizustu-Bilgisay
  ar 220104004834_Oğuz_Alp_Keleş % .
  /chatclient 127.0.0.1 8000
  Enter username: Mehmet
  [INFO] Connected.
  /join room1
  [JOINED] You joined room 'room1'

  [Ali] Hello room1
```

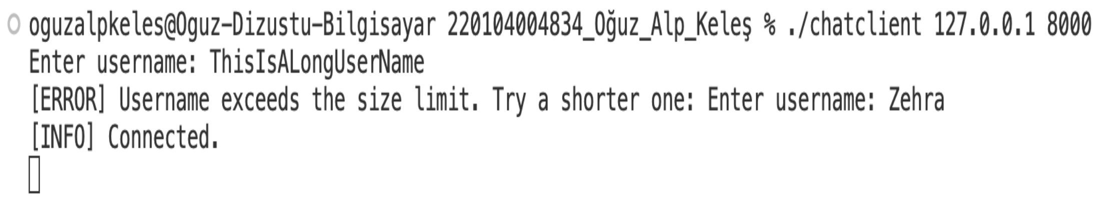
```
○ oguzalpkeles@Oguz-Dizustu-Bilgisay
  ar 220104004834_Oğuz_Alp_Keleş % .
  /chatclient 127.0.0.1 8000
  Enter username: Hasan
  [INFO] Connected.
  /join room2
  [JOINED] You joined room 'room2'

  [WHISPER] Veli: Merhaba Hasan
```

4.2 Test for Long or Already Taken User Names

Expectation: When a user enters an already taken user name or a long name, user should be given an error and notified. We can see that below.



```
○ oguzalpkeles@Oguz-Dizustu-Bilgisayar 220104004834_Oğuz_Alp_Keleş % ./chatclient 127.0.0.1 8000
  Enter username: Ali
  [ERROR] Username already taken. Try another: Enter username: Fatma
  [INFO] Connected.
```

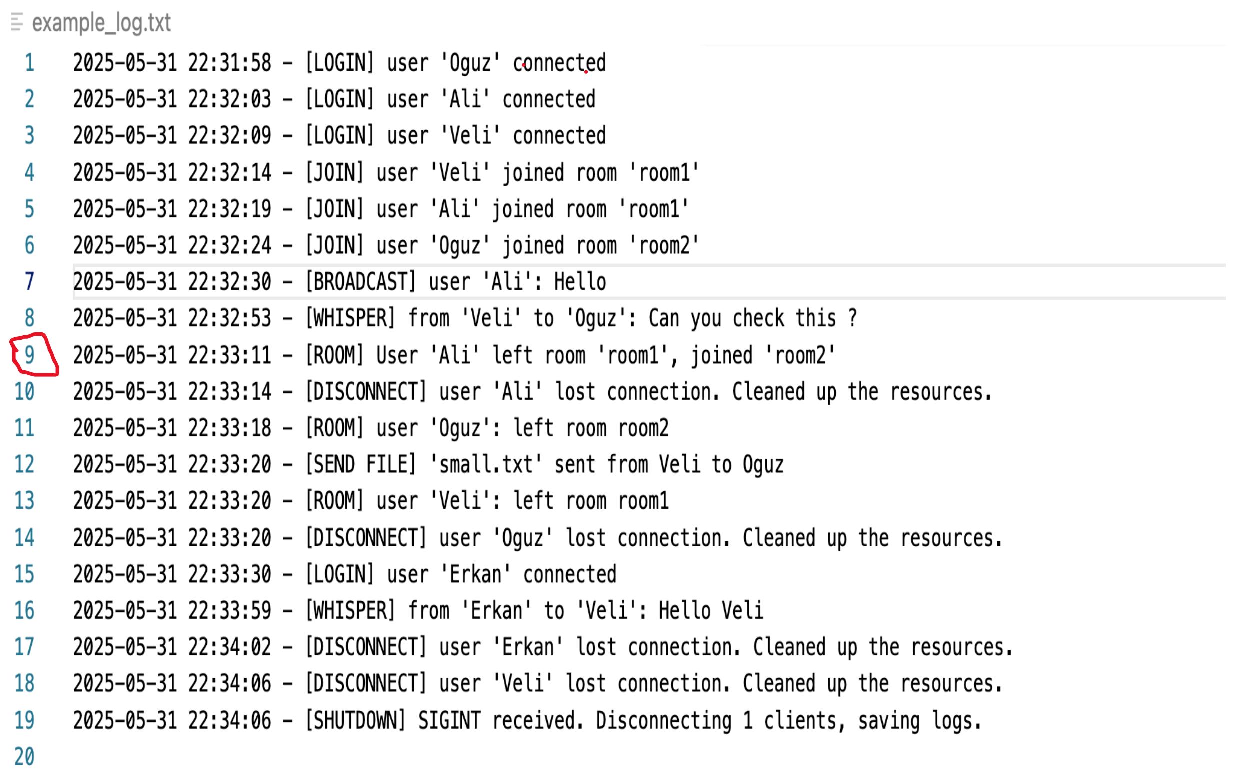


```
○ oguzalpkeles@Oguz-Dizustu-Bilgisayar 220104004834_Oğuz_Alp_Keleş % ./chatclient 127.0.0.1 8000
  Enter username: ThisIsALongUserName
  [ERROR] Username exceeds the size limit. Try a shorter one: Enter username: Zehra
  [INFO] Connected.
  ▯
```

4.3 Test for Room Switching and Server Disconnect

Expectation: When a user switched room it should be printed in the log file. In this example Ali changes its room fromroom1 to room2. We can see that it is printed in the log file example. We can also see that when server receives CRTL+C it cleans up the resources and prints it.

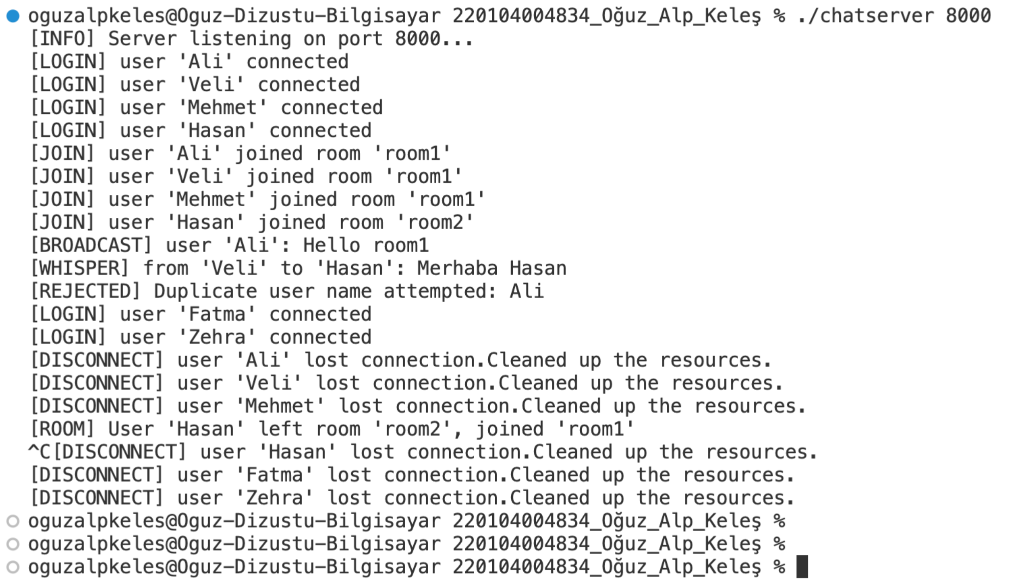
Example Log File



example_log.txt

```
2025-05-31 22:31:58 - [LOGIN] user 'Oguz' connected
2025-05-31 22:32:03 - [LOGIN] user 'Ali' connected
2025-05-31 22:32:09 - [LOGIN] user 'Veli' connected
2025-05-31 22:32:14 - [JOIN] user 'Veli' joined room 'room1'
2025-05-31 22:32:19 - [JOIN] user 'Ali' joined room 'room1'
2025-05-31 22:32:24 - [JOIN] user 'Oguz' joined room 'room2'
2025-05-31 22:32:30 - [BROADCAST] user 'Ali': Hello
2025-05-31 22:32:53 - [WHISPER] from 'Veli' to 'Oguz': Can you check this ?
2025-05-31 22:33:11 - [ROOM] User 'Ali' left room 'room1', joined 'room2'
2025-05-31 22:33:14 - [DISCONNECT] user 'Ali' lost connection. Cleaned up the resources.
2025-05-31 22:33:18 - [ROOM] user 'Oguz': left room room2
2025-05-31 22:33:20 - [SEND FILE] 'small.txt' sent from Veli to Oguz
2025-05-31 22:33:20 - [ROOM] user 'Veli': left room room1
2025-05-31 22:33:20 - [DISCONNECT] user 'Oguz' lost connection. Cleaned up the resources.
2025-05-31 22:33:30 - [LOGIN] user 'Erkan' connected
2025-05-31 22:33:59 - [WHISPER] from 'Erkan' to 'Veli': Hello Veli
2025-05-31 22:34:02 - [DISCONNECT] user 'Erkan' lost connection. Cleaned up the resources.
2025-05-31 22:34:06 - [DISCONNECT] user 'Veli' lost connection. Cleaned up the resources.
2025-05-31 22:34:06 - [SHUTDOWN] SIGINT received. Disconnecting 1 clients, saving logs.
```

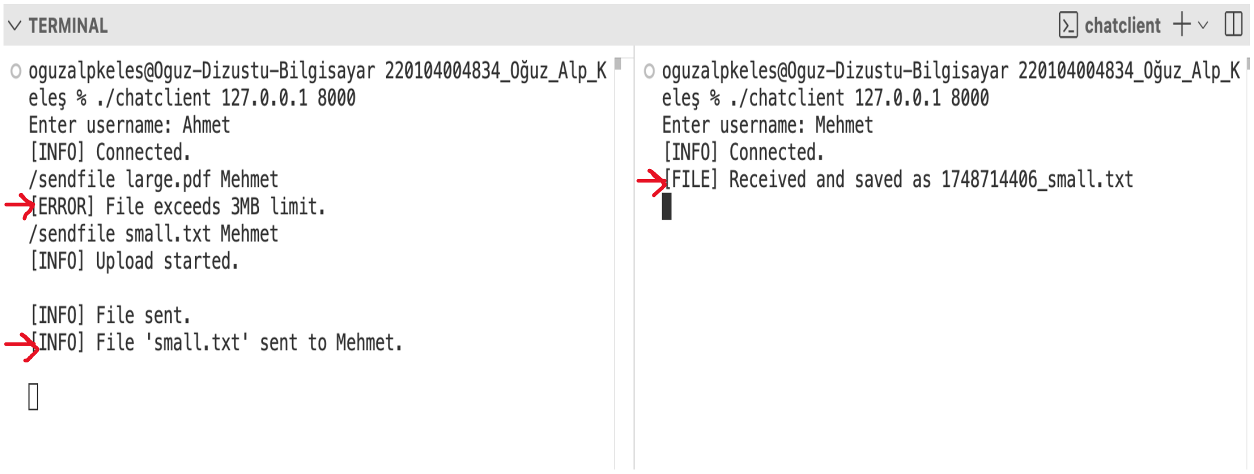
Example Server Terminal



```
● oguzalpkeles@Oguz-Dizustu-Bilgisayar 220104004834_Oğuz_Alp_Keleş % ./chatserver 8000
  [INFO] Server listening on port 8000...
  [LOGIN] user 'Ali' connected
  [LOGIN] user 'Veli' connected
  [LOGIN] user 'Mehmet' connected
  [LOGIN] user 'Hasan' connected
  [JOIN] user 'Ali' joined room 'room1'
  [JOIN] user 'Veli' joined room 'room1'
  [JOIN] user 'Mehmet' joined room 'room1'
  [JOIN] user 'Hasan' joined room 'room2'
  [BROADCAST] user 'Ali': Hello room1
  [WHISPER] from 'Veli' to 'Hasan': Merhaba Hasan
  [REJECTED] Duplicate user name attempted: Ali
  [LOGIN] user 'Fatma' connected
  [LOGIN] user 'Zehra' connected
  [DISCONNECT] user 'Ali' lost connection.Cleaned up the resources.
  [DISCONNECT] user 'Veli' lost connection.Cleaned up the resources.
  [DISCONNECT] user 'Mehmet' lost connection.Cleaned up the resources.
  [ROOM] User 'Hasan' left room 'room2', joined 'room1'
  ^C[DISCONNECT] user 'Hasan' lost connection.Cleaned up the resources.
  [DISCONNECT] user 'Fatma' lost connection.Cleaned up the resources.
  [DISCONNECT] user 'Zehra' lost connection.Cleaned up the resources.
○ oguzalpkeles@Oguz-Dizustu-Bilgisayar 220104004834_Oğuz_Alp_Keleş %
○ oguzalpkeles@Oguz-Dizustu-Bilgisayar 220104004834_Oğuz_Alp_Keleş %
○ oguzalpkeles@Oguz-Dizustu-Bilgisayar 220104004834_Oğuz_Alp_Keleş %
```

4.4 Test for File Sending & File Upload Queue & File Waiting Duration

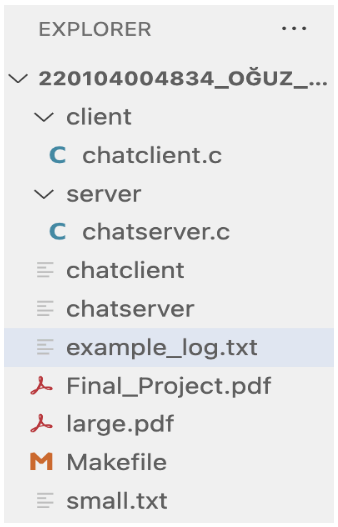
Expectation: **Ahmet will try to send file to Mehmet. First, he will send the “large.pdf” (example of a big file), it is ecpected to give an error. Then he will send “small.txt” (example of a small file) which will be sent correctly. In my implementation, files are actually sent. Their name is prefixed with a stamp to ensure that no file name collisions occur. File will be sent to a directory that the client runs.**



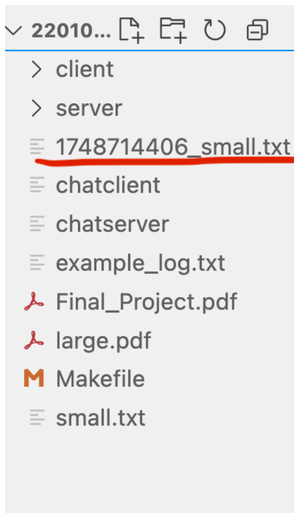
```
○ oguzalpkeles@Oguz-Dizustu-Bilgisayar 220104004834_Oğuz_Alp_K
  eleş % ./chatclient 127.0.0.1 8000
  Enter username: Ahmet
  [INFO] Connected.
  /sendfile large.pdf Mehmet
  [ERROR] File exceeds 3MB limit.
  /sendfile small.txt Mehmet
  [INFO] Upload started.

  [INFO] File sent.
  [INFO] File 'small.txt' sent to Mehmet.
```

```
○ oguzalpkeles@Oguz-Dizustu-Bilgisayar 220104004834_Oğuz_Alp_K
  eleş % ./chatclient 127.0.0.1 8000
  Enter username: Mehmet
  [INFO] Connected.
  [FILE] Received and saved as 1748714406_small.txt
```

Below we see that initially only small.txt exists, but after file sending operation is completed, the prefixed file can be seen in the explorer.

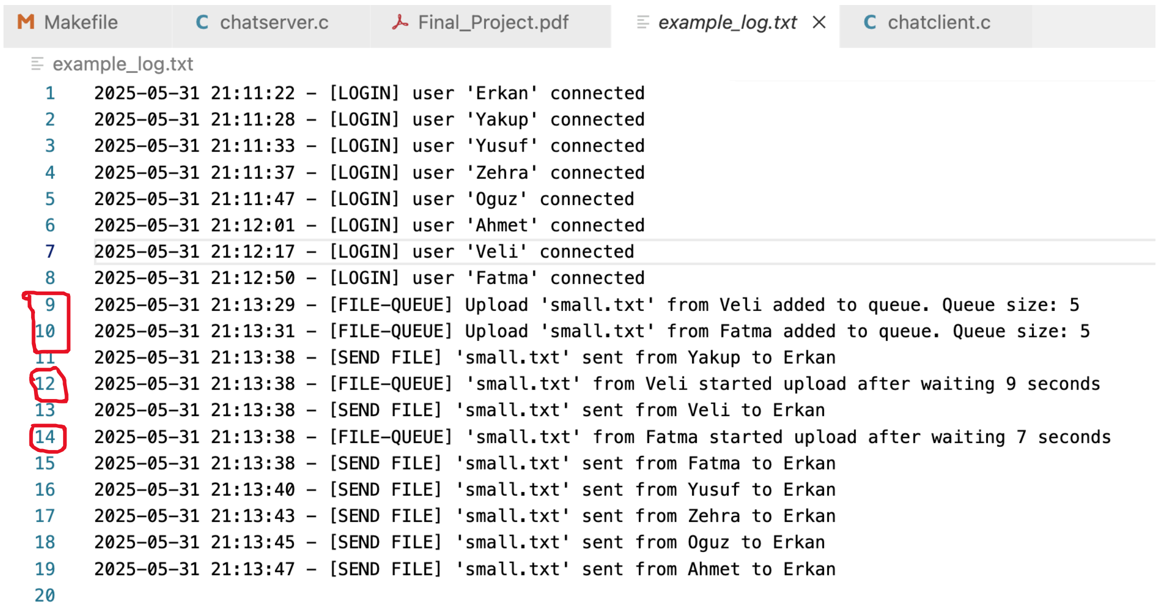


```
EXPLORER ...
∨ 220104004834_OĞUZ_...
  ∨ client
    C chatclient.c
  ∨ server
    C chatserver.c
  chatclient
  chatserver
  example_log.txt
  Final_Project.pdf
  large.pdf
  Makefile
  small.txt
```



```
∨ 22010...
  > client
  > server
  1748714406_small.txt
  chatclient
  chatserver
  example_log.txt
  Final_Project.pdf
  large.pdf
  Makefile
  small.txt
```

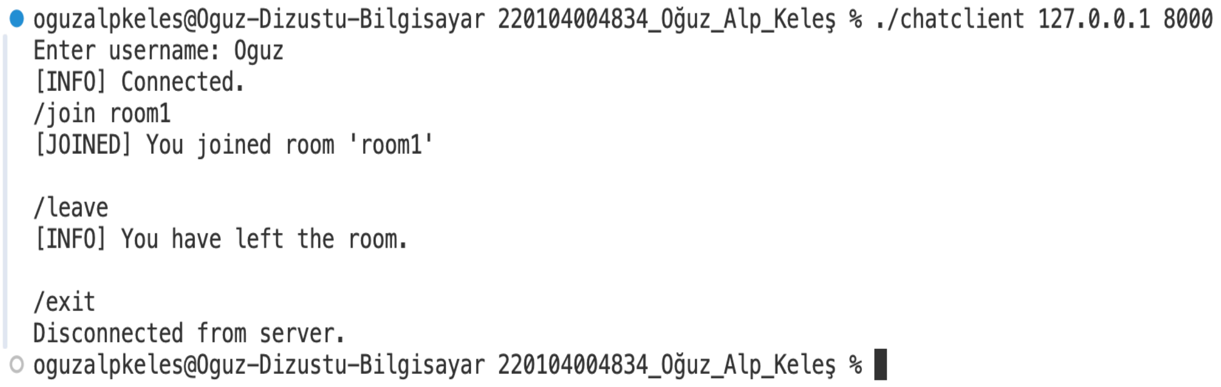
Below you can see the implementation of the file waiting queue and file queue duration. To simulate this, I put a sleep() inside the related function, in order to test it please uncomment it inside the code !!!!!!



```
2025-05-31 21:11:22 - [LOGIN] user 'Erkan' connected
2025-05-31 21:11:28 - [LOGIN] user 'Yakup' connected
2025-05-31 21:11:33 - [LOGIN] user 'Yusuf' connected
2025-05-31 21:11:37 - [LOGIN] user 'Zehra' connected
2025-05-31 21:11:47 - [LOGIN] user 'Oguz' connected
2025-05-31 21:12:01 - [LOGIN] user 'Ahmet' connected
2025-05-31 21:12:17 - [LOGIN] user 'Veli' connected
2025-05-31 21:12:50 - [LOGIN] user 'Fatma' connected
2025-05-31 21:13:29 - [FILE-QUEUE] Upload 'small.txt' from Veli added to queue. Queue size: 5
2025-05-31 21:13:31 - [FILE-QUEUE] Upload 'small.txt' from Fatma added to queue. Queue size: 5
2025-05-31 21:13:38 - [SEND FILE] 'small.txt' sent from Yakup to Erkan
2025-05-31 21:13:38 - [FILE-QUEUE] 'small.txt' from Veli started upload after waiting 9 seconds
2025-05-31 21:13:38 - [SEND FILE] 'small.txt' sent from Veli to Erkan
2025-05-31 21:13:38 - [FILE-QUEUE] 'small.txt' from Fatma started upload after waiting 7 seconds
2025-05-31 21:13:38 - [SEND FILE] 'small.txt' sent from Fatma to Erkan
2025-05-31 21:13:40 - [SEND FILE] 'small.txt' sent from Yusuf to Erkan
2025-05-31 21:13:43 - [SEND FILE] 'small.txt' sent from Zehra to Erkan
2025-05-31 21:13:45 - [SEND FILE] 'small.txt' sent from Oguz to Erkan
2025-05-31 21:13:47 - [SEND FILE] 'small.txt' sent from Ahmet to Erkan
```

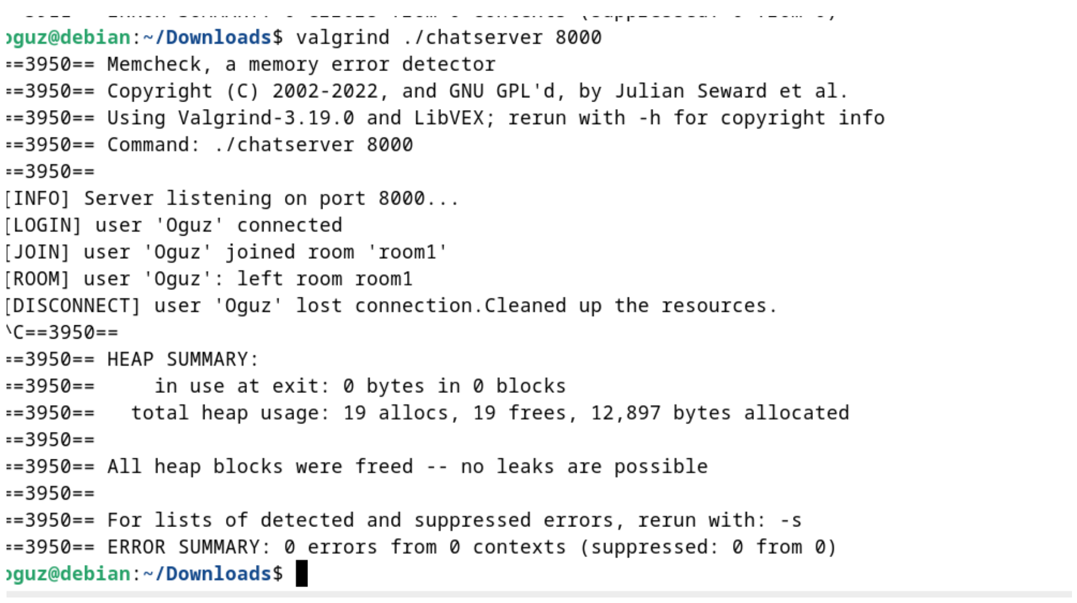
4.5 Test for leave and exit commands

Expectation: **When user leaves it should leave and be informed, exit should disconnect it from the terminal. You can see that below.**



```
oguzalpkeles@Oguz-Dizustu-Bilgisayar 220104004834_Oğuz_Alp_Keleş % ./chatclient 127.0.0.1 8000
Enter username: Oguz
[INFO] Connected.
/join room1
[JOINED] You joined room 'room1'

/leave
[INFO] You have left the room.

/exit
Disconnected from server.
oguzalpkeles@Oguz-Dizustu-Bilgisayar 220104004834_Oğuz_Alp_Keleş % 
```

4.6 Valgrind Resource Leak Test



```
oguz@debian:~/Downloads$ valgrind ./chatserver 8000
==3950== Memcheck, a memory error detector
==3950== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==3950== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==3950== Command: ./chatserver 8000
==3950==
[INFO] Server listening on port 8000...
[LOGIN] user 'Oguz' connected
[JOIN] user 'Oguz' joined room 'room1'
[ROOM] user 'Oguz': left room room1
[DISCONNECT] user 'Oguz' lost connection.Cleaned up the resources.
^C==3950==
==3950== HEAP SUMMARY:
==3950==     in use at exit: 0 bytes in 0 blocks
==3950==   total heap usage: 19 allocs, 19 frees, 12,897 bytes allocated
==3950==
==3950== All heap blocks were freed -- no leaks are possible
==3950==
==3950== For lists of detected and suppressed errors, rerun with: -s
==3950== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
oguz@debian:~/Downloads$ 
```

5. Issues Faced and How They Were Solved

Zombie Threads: Threads were not detached, leading to resource leakage. pthread_detach() was added immediately after thread creation to ensure system resources were freed.

Incorrect File Queue Behavior: The semaphore for limiting concurrent uploads was not functioning as expected. Error checking on initialization of the semaphore resulted in solving the problem. Problem was that semaphore was not initialized properly, sem_open() is used instead of sem_init() for system to be more reliable and that solved the problem.

Username Length: Input longer than the max username limit was being truncated silently. A check using strcspn() was added to catch and reject overlong usernames before assignment.

Improper File Storage: Files were originally saved on the server side regardless of the target. Functionality was adjusted to ensure files are sent directly to the target client's working directory. (the path in which the client executable is executed)

6. Conclusion and Potential Improvements

This project tries to implement a multi-client chat system using C, with support for room-based communication, private messaging, and file transfer between clients. The system enforces username uniqueness, maintains server-side logging, and controls concurrent file transfers using a counting semaphore. The use of POSIX threads and proper synchronization mechanisms ensures stable concurrent handling of clients.

The client implementation cleanly supports command parsing and allows seamless interaction with the server. The file transfer logic is robust, using time-stamped filenames to avoid collisions and safely storing files in the receiver's working directory.

Throughout the project, several practical issues were encountered and addressed, including thread management, username validation, file queue enforcement, and resource leaks. These solutions enhanced the system’s reliability, maintainability, and scalability.