Project
Design and Implementation of a Multi-Functional Multi-Threaded Online Chat Platform

1. Description

This project involves implementing a multi-threaded network application server that supports multiple concurrent client connections for real-time communication and collaboration. The platform will allow users to engage in text-based chat, transfer files, participate in group discussions, and collaboratively edit documents.

The aim of this project is to deepen your understanding of network programming, multithreading, message protocol design, secure data transmission, and user permission management.

By the end of this project, you will have built a high-performance chat and collaboration platform capable of handling multiple users simultaneously while ensuring security, reliability, and scalability.

2. Core Features

2.1 Instant Messaging

- Private Chat: One-to-one text messaging between users.

- Group Chat: Real-time text messaging within groups.

- Message History: The server stores conversation history, allowing users to retrieve past messages.

2.2 File Transfer

- File Upload: Clients can upload files to the server with progress feedback.

- File Download: Clients can download files from the server.

- File Sharing in Chat: Files can be shared directly in group chats.

2.3 Collaborative Editing

- Multiple users can edit the same document simultaneously.
- The server manages version control and conflict resolution.
- Supports version rollback and change comparison.

2.4 User and Permission Management

- User registration and login.
- Authentication using username and password.
- Role-based permissions (Regular User, Administrator, Group Owner).
- Admins can create/delete groups, remove members, and mute users.

3. Server Structure

3.1 Server Initialization:

- Read configurations (port, max connections) from a configuration file.
- Create and bind the server socket to the specified port.
- Initialize a thread pool to handle concurrent connections.

3.2 Main Thread Loop:

- Listen for and accept incoming client connections.
- Dispatch client requests to worker threads.

3.3 Worker Thread Handling:

- Parse and process client requests (chat, file transfer, document editing).
- Return the operation result or an error message.

4. Communication Protocol

A simple, text-based protocol will be used for client-server communication.

Example commands:

MSG PRIVATE target_username message_content

MSG GROUP group_id message_content

FILE UPLOAD filename size

FILE DOWNLOAD file_id

DOC EDIT doc_id change_data version

DOC RESTORE doc_id version

5. Security and Reliability

- TLS Encryption: All client-server communications will be encrypted.

- Message Integrity Check: Use checksums or hashes to verify data integrity.

- Permission Validation: Each request must include a session token to validate the user's rights.

6. Logging

The server logs all operations in detail, including user activities, errors, and system events. Logs should be in an easy-to-analyze format, for example:

[2025-05-01 10:12:30] [INFO] LOGIN: user1 (Success)

[2025-05-01 10:15:45] [INFO] MSG: user1 -> user2 ("Hello") (Success)

[2025-05-01 10:18:10] [ERROR] FILE DOWNLOAD: user3 -> file123 (Failed - No Permission)

7. Performance Testing

1) Concurrent Performance: Use tools such as Apache JMeter or wrk to simulate a high number of concurrent connections and measure latency, throughput, and error rate.
2) File Transfer Performance: Evaluate upload/download speeds for different file sizes.
3) Collaborative Editing Performance: Measure synchronization speed and conflict handling when multiple users edit the same document.