Assignment -1

Course: Internet Architecture and Protocols [CS60008]

Language: Python 3.9+

Allowed libraries: socket, threading, ssl, hashlib, bcrypt, redis, docker (no high-level chat frameworks)

Delivery: Git repository with source, Docker artifacts, and a short PDF report.

Overview: - A basic chat server code is provided as the starting point for this assignment. The specified problems have to be implemented to enhance and modify the functionalities of the server and client in accordance with the stated requirements.

Problem 1: - Thread-Based Chat Server

Objective: Replace asyncio with a thread-based server and understand blocking I/O, thread lifecycle, and shared state.

Requirements: - TCP-based server using socket. - One thread per connected client (threading.Thread). - Broadcast messages to all connected clients. - Handle client disconnects gracefully.

Problem 2: - Authentication

Objective: - Add authentication and secure credential handling.

Requirements: - Clients must send LOGIN <username> <password> on connect. - Passwords must be securely hashed (recommend bcrypt). - Prevent duplicate logins; reject or force logout. - Maintain authenticated session per connection.

Problem 3: - Handling **duplicate login attempts**

Objective: - Design and implement a policy for handling duplicate login attempts in a multi-client chat server, ensuring consistency, security, and correct session management in a concurrent environment.

Requirements: - Students must implement one of the following duplicate login handling policies. The chosen policy must be clearly documented.

1. Reject Duplicate Login: - If a user attempts to log in with a username that already has an active session, the server must reject the new login attempt. The server should return an appropriate error message to the client. The existing session must remain unaffected.

OR

2. Force Logout Existing Session: - If a user logs in with a username that already has an active session, the server must terminate the existing session. The earlier client must be notified of the forced logout before disconnection. The new login attempt must succeed and establish a valid session.

Problem 4: - Chat Rooms

Objective: - Implement rooms so messages are routed only to members of a room.

Requirements: - Support commands: /join <room>, /leave, /rooms. - Default lobby on login. - Messages broadcast only within current room.

Problem 5: - Publish–Subscribe communication model

Objective: - Implement a publish–subscribe communication model for the chat system, where clients can subscribe to other clients and receive messages only from those they have subscribed to.

Requirements: - Clients must be able to subscribe and unsubscribe to other clients using explicit commands (e.g., /subscribe <username>).
The server must maintain subscription information and enforce it centrally.
When a client publishes a message, the server must multicast the message only to its subscribers.
Message ordering must be preserved per publisher.
The implementation must be thread-safe and handle client disconnects gracefully.
Existing authentication and session management features must continue to work.

Problem 6: - Redis Integration (Distributed State)

Objective: - Move shared state to Redis and support multiple server instances.

Redis usage (mandatory): - Store user sessions in Redis hashes. - Store room membership in Redis sets. - Use Redis Pub/Sub for cross-server message broadcasting. - Servers should be stateless with respect to global session/room data.

Behavioral requirement: - If two server instances are running and clients connect to different instances, users in the same room must still receive each other's messages.

Problem 7: - TLS / Encrypted Transport

Objective: - Secure the client-server channel with TLS.

Requirements: - Use Python ssl module to wrap sockets with TLS (use self-signed certs for testing). - Clients must verify the server certificate (CA or pinned cert). - Reject plaintext connections.

Problem 8: - Dockerized Deployment

Objective: - Provide reproducible deployment using Docker and docker-compose.

Requirements: - Provide Dockerfile for server and docker-compose.yml that brings up server(s) and Redis. - docker-compose up --build should start the entire system for testing.

Deliverables-

1. Git repository with full source and clear README.
2. Docker artifacts (Dockerfile(s), docker-compose.yml).
3. Credentials & setup instructions for testing (how to create accounts, test rooms).
4. Short PDF report (2–4 pages) describing design choices, Redis schema, thread-safety issues, TLS decisions, and test results.