Project Specification: "PyLingo" Minimalist CLI Chat System

1. Objective

To design and implement a multi-user, room-based chat server and client using Python's advanced networking and concurrency primitives. This project synthesizes concepts from **Modules 1 through 6**, specifically targeting socket programming, thread/process synchronization, and asynchronous I/O.

2. Technical Requirements

- **Language:** Python 3.12+
- **Concurrency Model:** Students may choose between:
  - **Multi-threaded:** One thread per client connection (using threading and queue).
  - **Asynchronous:** Single-threaded event loop (using asyncio or uvloop).
- **Networking:** TCP/IP sockets (socket module).
- **Data Format:** Messages must be serialized (using json or pickle) to support structured data (e.g., separating the command from the message body).

---

3. Functional Requirements

3.1 The Server

The server must be a standalone script that listens on a configurable port (default: 5000).

- **State Management:** Maintain a registry of connected users, their chosen nicknames, and the "rooms" they currently occupy.
- **Concurrency Safety:** If using threads, all access to the global state (user lists/rooms) **must** be protected by synchronization primitives (Lock or RLock).
- **Broadcasting:** When a user sends a message, the server must identify all other users in the same room and relay the message to them.

3.2 The Client

The client must be a CLI application that handles **simultaneous** user input and incoming network messages.

- **Interface:** Must provide a prompt for user commands while cleanly printing incoming

messages from other users without overlapping the input line.
- **Connection:** Gracefully handle server disconnects or "Connection Refused" errors.

---

4. Supported Command Set

The server must parse and react to the following "Slash Commands":

Command	Argument	Description
/name	[username]	Sets or changes the user's display name.
/list	None	Returns a list of all active chat rooms.
/join	[room]	Moves the user into a specific room. Leave previous room if applicable.
/users	None	Lists all users currently in the user's current room.
/leave	None	Removes the user from the current room.
/help	None	Displays a summary of available commands.
/close	None	Closes the socket and exits the application.

---

5. Implementation Challenges (The "Advanced" Part)

5.1 The "Partial Read" Problem

Students must implement a **Framing Protocol**. Since TCP is a stream, they cannot assume one send() equals one recv(). They should implement:

- **Length-Prefixed Messaging:** Prepend the size of the message (e.g., 4-byte header).
- **Delimiter-based Messaging:** Use a unique character (like \n) to terminate messages.

5.2 Thread-Safe State

If using the threading module, students must demonstrate proper use of threading.Lock() when modifying the room dictionary.

5.3 Graceful Teardown

The server should catch KeyboardInterrupt (Ctrl+C) and notify all connected clients before shutting down the socket.

---

6. Deliverables

1. server.py: The core chat engine.
2. client.py: The user interface.
3. protocol.py (Optional): A shared module for common message parsing or constants.
4. A short README.md explaining the chosen concurrency model (Threads vs. Async) and why it was selected.