Lab Assignment 5: Simple File Server with Directory Listing and File Download

Objective

Develop a TCP socket-based file server that supports multiple commands within a single session. In addition to transferring files, the server will allow clients to request a list of available files. This assignment builds on previous assignments (remote command execution and file transfer) by requiring session-based communication and simple protocol design.

Server Requirements

1. **Connection and Session Management:**
    - Listen for incoming TCP connections on a configurable port.
    - Upon a client connection, send a welcome message (e.g., “Welcome to Simple File Server”).
    - Enter a command-processing loop until the client issues a quit command.
2. **Command Processing:**
    - LIST Command:
        - When the client sends `LIST`, the server should scan a designated directory (e.g., its current working directory) and return a list of file names (one per line).
    - GET Command:
        - When the client sends `GET <filename>`, the server checks for the file in the designated directory.
        - If the file exists:
            - Send a header containing the file size.

        - Transmit the file in chunks (using a fixed buffer size; note that you cannot use a buffer larger than what was specified in previous labs).
    - If the file does not exist, send an error message (e.g., “ERROR: File Not Found”).
  - QUIT Command:
    - On receiving `QUIT`, the server should close the client connection gracefully.
3. **Error Handling & Robustness:**
  - Ensure robust error checking for file I/O and network operations.
  - Log each command received and any errors encountered for debugging purposes.

Client Requirements

1. **Connection Setup:**
  - Connect to the server’s TCP port.
  - Display the welcome message upon connection.
2. **User Interaction:**
  - Provide a command-line interface that accepts the following commands:
    - LIST: Request and display the list of available files.
    - GET : Request a specific file from the server.
      - On success, save the file locally and display the number of bytes received.
      - On error, display the received error message.
    - QUIT: End the session and close the connection.
3. **Data Handling:**
  - For file transfers, handle reading/writing in small fixed-size chunks (similar to Session 5 constraints).
  - Appropriately display any error messages received from the server.

Additional Specifications

- Language Options: Implementation can be done in C, C++, or Python.
- Resource Management: Ensure all sockets and file descriptors are closed properly, and handle any abrupt client disconnections gracefully.