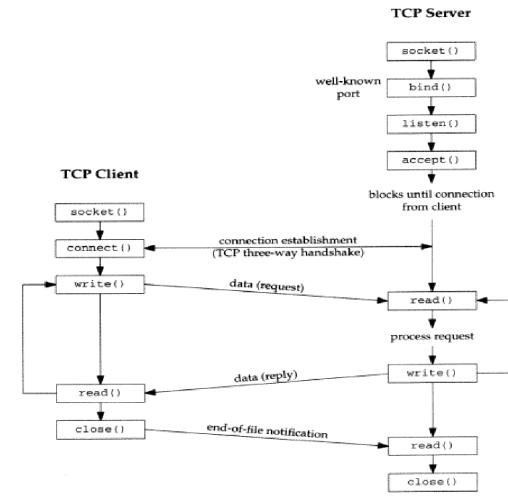
Network File Sharing Server and Client

Flowchart



**1. System Overview**

The networked file-sharing application consists of a server-client architecture that enables users to share files over a network. The server manages file requests and responses, while the client initiates file operations such as listing, downloading, and uploading files.

**2. Functional Requirements**

**2.1 Server Application**

* **2.1.1 Start and Listen on a Port**
  + **Description**: The server should start on a predefined port (default: 8080) and listen for incoming client connections.
  + **Input**: None (Automatically configured to start on the specified port).
  + **Output**: Server is active and ready to accept client connections.
* **2.1.2 Client Authentication**
  + **Description**: The server should require clients to authenticate with a predefined password before allowing any file operations.
  + **Input**: Password from the client.
  + **Output**: Authentication success or failure message sent to the client.
* **2.1.3 List Files**
  + **Description**: The server should provide a list of all files in its current directory upon receiving the LIST command from a client.
  + **Input**: LIST command from the client.
  + **Output**: List of filenames sent to the client. If the list is too large to send in one message, an error message should be sent.
* **2.1.4 Send File to Client**
  + **Description**: The server should send a file to the client upon receiving the GET <filename> command.
  + **Input**: GET <filename> command from the client.
  + **Output**: The requested file is transmitted to the client. If the file does not exist, an error message is sent.
* **2.1.5 Receive File from Client**
  + **Description**: The server should accept a file upload from the client upon receiving the PUT <filename> command.
  + **Input**: PUT <filename> command from the client, followed by file data.
  + **Output**: The file is saved in the server's directory. A confirmation message is logged.
* **2.1.6 Logging**
  + **Description**: The server should log all major activities, including client connections, commands received, files sent/received, and authentication results.
  + **Input**: Commands and events during server operation.
  + **Output**: Log entries in server\_log.txt.
* **2.1.7 Handle Disconnections**
  + **Description**: The server should properly handle client disconnections and clean up any associated resources.
  + **Input**: Client disconnects.
  + **Output**: Log entry noting the disconnection and freeing of resources.

**2.2 Client Application**

* **2.2.1 Establish Connection to Server**
  + **Description**: The client should establish a connection to the server on the predefined IP address (default: 127.0.0.1) and port (default: 8080).
  + **Input**: Server IP address and port.
  + **Output**: Connection established or error message.
* **2.2.2 Authentication**
  + **Description**: The client must send a password to the server upon connection to authenticate itself.
  + **Input**: Password provided by the user.
  + **Output**: Authentication success or failure message received from the server.
* **2.2.3 List Files**
  + **Description**: The client can request a list of files available on the server by issuing the LIST command.
  + **Input**: LIST command from the user.
  + **Output**: List of filenames displayed to the user.
* **2.2.4 Download File**
  + **Description**: The client can download a file from the server by issuing the GET <filename> command.
  + **Input**: GET <filename> command from the user.
  + **Output**: The file is saved to the client’s local directory. If the file does not exist on the server, an error message is displayed.
* **2.2.5 Upload File**
  + **Description**: The client can upload a file to the server by issuing the PUT <filename> command.
  + **Input**: PUT <filename> command from the user.
  + **Output**: The file is sent to the server. A confirmation message is displayed upon successful upload.
* **2.2.6 Logging**
  + **Description**: The client should log all major activities, including connection status, commands sent, files sent/received, and server responses.
  + **Input**: Commands and events during client operation.
  + **Output**: Log entries in client\_log.txt.
* **2.2.7 Command Loop**
  + **Description**: The client should continuously prompt the user for commands until the EXIT command is issued.
  + **Input**: User commands (LIST, GET <filename>, PUT <filename>, EXIT).
  + **Output**: Execution of the respective command or termination of the session.

**3. Non-Functional Requirements**

* **3.1 Performance**
  + The server should handle multiple client connections sequentially without crashing. File transfers should be efficient, with minimal latency.
* **3.2 Security**
  + Password-based authentication is required before any file operations can be performed. The server should not allow unauthenticated access.
* **3.3 Usability**
  + Both the server and client applications should have a straightforward command-line interface, with clear instructions and feedback.
* **3.4 Scalability**
  + The application should be able to handle a reasonable number of simultaneous connections and file operations without significant performance degradation.
* **3.5 Reliability**
  + The application should be robust against common network issues, such as disconnections, and ensure data integrity during file transfers.