Alexis Martin, Justin Dong, Michael Rojas, Jonathan Martin

Professor Kurban

CPSC 471

2 December 2023

Socket Project Report

Protocol Design:

1. Connection Establishment:

* Client-Server Handshake:
  + The client initiates a connection to the server using the server's IP address and port.
  + Upon successful connection, the server sends an initial welcome message to the client.

2. Command Handling:

* Command Structure:
  + Commands sent by the client are plain text strings, where the first word represents the command keyword.
  + Commands are case-insensitive for user convenience.
* Supported Commands:
  + GET: get <file name>
    - Client requests a file from the server.
    - Server responds with the content of the requested file or an error message.
  + PUT: put <file name>
    - Client uploads a file to the server.
    - Server responds with a "send\_file" message, indicating that the client should send the file content.
  + LS: ls
    - Client requests the list of files held within the server's directory.
    - Server responds with a newline-separated list of file names.
  + HELP: help
    - Client requests the list of valid commands and their usage.
    - Server responds with a formatted help message.
  + QUIT: quit
    - Client initiates the termination of the connection.
    - Server responds with a "quit" signal, and the connection is closed.

3. File Transfer:

* PUT Command:
  + Upon receiving a "put" command, the server sends a "send\_file" message to the client.
  + The client responds by sending the file size as a 12-byte header.
  + The server reads the header, calculates the total file size, and acknowledges the receipt.
  + The client sends the file content in chunks of 1024 bytes until the entire file is transferred.
  + The server writes the received content to the specified file path.
* GET Command:
  + The client specifies the "get" command with the desired file name.
  + The server attempts to locate the requested file, and either responds with the file content or an error message.

4. Connection Termination:

* QUIT Command:
  + When the client issues a "quit" command, the server responds with a "quit" signal.
  + Both client and server close their sockets, terminating the connection and properly releasing all resources

5. File Storage:

* Resource Folders:
  + The project includes dedicated folders, client\_resources and server\_resources, for storing client and server files, respectively.
  + Files uploaded by clients are stored in the server\_resources folder on the server’s side.
  + No permission handling for accessing stored files; Anyone with access to the server can access the stored files. A system for authentication would be required to improve this insecurity.

6. Error Handling:

* Invalid Commands:
  + If the client sends an invalid command or an incorrectly formatted command, the server responds with an error message.
* File Not Found:
  + In cases where a requested file is not found, the server notifies the client with an appropriate error message.

7. Reliability and Robustness:

* File Transfer Reliability:
  + The protocol ensures reliability in file transfer by acknowledging the receipt of the file size header.
  + Error handling mechanisms are in place to address potential issues during the file transfer.
* Connection Handling:
  + Graceful handling of unexpected client disconnection is implemented, allowing the server to exit without errors.

Difficulties Faced:

* Problem: Having to establish a robust structure and implementing a while loop to handle WebSocket communication posed a challenge in the server-side code.
* Struggle:
  + Setting up the initial structure for WebSocket communication involved considerations such as handling connections, sending and receiving messages, and managing the flow of communication.
  + The implementation of a while loop to continuously listen for incoming WebSocket messages required careful coordination to ensure server responsiveness without blocking other functionalities.
* Solution:
  + Structured the server code to manage WebSocket connections, utilizing the appropriate libraries and methods for handling WebSocket communication.
  + Implemented a well-designed while loop to continuously listen for WebSocket messages, ensuring that the server could efficiently handle multiple connections and respond promptly to client requests.
  + Addressed potential issues such as unexpected disconnections, error handling, and graceful termination of WebSocket communication within the loop.
* Problem: Efficiently processing commands received from clients in the server-side code proved challenging, especially with varying command types and the need for expandability.
* Struggle:
  + Devising a strategy to handle different types of commands, such as "get," "put," "ls," and "quit," presented difficulties in maintaining clean and scalable code.
  + Ensuring flexibility for future command additions or modifications required a solution that could easily integrate new functionalities without disrupting the existing codebase.
* Solution:
  + Implemented a dedicated class, CommandHandler, responsible for interpreting and executing commands received from clients.
  + The CommandHandler class contained methods for handling specific command types, such as "get," "put," and "ls," streamlining command processing and improving code organization.
  + Designed the class to be extensible, allowing straightforward addition of new command-handling methods as the server's functionality expanded.
* Problem: Implementing the "put" command to transfer files from the client to the server posed challenges, particularly in ensuring reliable and complete file transmission.
* Struggle:
  + Encountered issues with file transfer reliability, with incomplete or corrupted files being received on the server side.
  + Difficulty in synchronizing the server to expect incoming file content and ensuring that the client sends the file correctly.
* Solution:
  + Reversed the communication flow so that the server would send a "send\_file" message to the client upon receiving a "put" command. This informs the client to initiate the file transfer.
  + Implemented corresponding logic in both the client and server to handle the file transfer, ensuring data integrity and completeness.