01. TCP File Transfer

Le Nhat Anh

December 11, 2024

Goal

The objective of this practical work is to implement a 1-to-1 file transfer system over TCP/IP in a command-line interface (CLI) environment. This is based on an existing chat system and includes:

- A server to handle file operations.
- A client to request file uploads, downloads, and listing files.
- Communication over TCP sockets.

Protocol Design

The protocol is simple and command-driven, with three main operations:

- **UPLOAD filename content**: Sends a file to the server.
- **DOWNLOAD filename**: Requests a file from the server.
- **LIST**: Requests a list of available files on the server.

Each command is sent as a string over the socket. Responses are sent back to the client as strings (except for file content in binary during download).

System Organization

The system consists of two main components:

- **Server:** Listens for client connections and handles file operations.
- **Client:** Connects to the server to perform upload, download, and list operations.

Implementation

File Transfer Logic

The file transfer is implemented using Python's `socket` module. The server processes incoming requests and performs file I/O operations, while the client sends commands and processes responses.

Code Snippets

Client Implementation

Listing 1: Client Implementation

```
def send_request(request):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((ip, port))
        s.sendall(request.encode())

        response = s.recv(1024).decode()
        s.close()
        return response
    except ConnectionRefusedError:
        return "Connection␣refused.␣Make␣sure␣the␣server␣is␣running."
    except socket.error as e:
        return f"Socket␣error:␣{e}"
    finally:
        s.close()


def upload():
    filename = input("Enter␣filename␣to␣upload:␣").strip()

    if not os.path.exists(filename):
        return "File␣does␣not␣exist"

    with open(filename, "r") as file:
        content = file.read()

    request = f"UPLOAD␣{filename}␣{content}"
    response = send_request(request)

    return response


def download():
    filename = input("Enter␣filename␣to␣download:␣").strip()
    request = f"DOWNLOAD␣{filename}"
    response = send_request(request)

    if "File␣not␣found" not in response:
        with open(filename, "wb") as f:
            f.write(response.encode())
        response = f"File␣{filename}␣downloaded␣successfully."

    return response


def list():
    request = "LIST"
    response = send_request(request)

    return response
```

Server Implementation

Listing 2: Server Implementation

```
def handle_request(request, conn):
    parts = request.split("␣", 2)
    operation = parts[0]

    if operation == "UPLOAD":
        filename = parts[1]
        content = parts[2]

        with open(filename, "w") as f:
            f.write(content)
        return "File␣uploaded␣successfully"

    elif operation == "DOWNLOAD":
        filename = parts[1]

        if os.path.exists(filename):
            with open(filename, "rb") as f:
                content = f.read()
            conn.sendall(content)
            return ""
        else:
            return "File␣not␣found"

    elif operation == "LIST":
        files = os.listdir(".")
        return "␣".join(files)
    else:
        return "Invalid␣operation"


def start_server(port):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.bind(("0.0.0.0", port))  # Bind to all available interfaces
        s.listen(5)
        print(f"Server␣started␣and␣listening␣on␣port␣{port}")

        while True:
            conn, addr = s.accept()
            try:
                request = conn.recv(1024).decode()
                response = handle_request(request, conn)
                if response:
                    conn.sendall(response.encode())
                print(f"Request:␣{request}␣|␣Response:␣{response}")
            except Exception as e:
                conn.sendall(f"Error␣processing␣request:␣{e}".encode("utf-8"))
                print(f"Request:␣{request}␣|␣Error␣processing␣request:␣{e}")
            finally:
                conn.close()
    except socket.error as e:
        print(f"Socket␣error:␣{e}")
    except Exception as e:
        print(f"Server␣error:␣{e}")
    finally:
        s.close()
```

Summary

This system demonstrates a basic file transfer protocol over TCP/IP using Python sockets. The implementation supports essential file operations and provides a foundation for more advanced features such as authentication, encryption, and robust error handling.

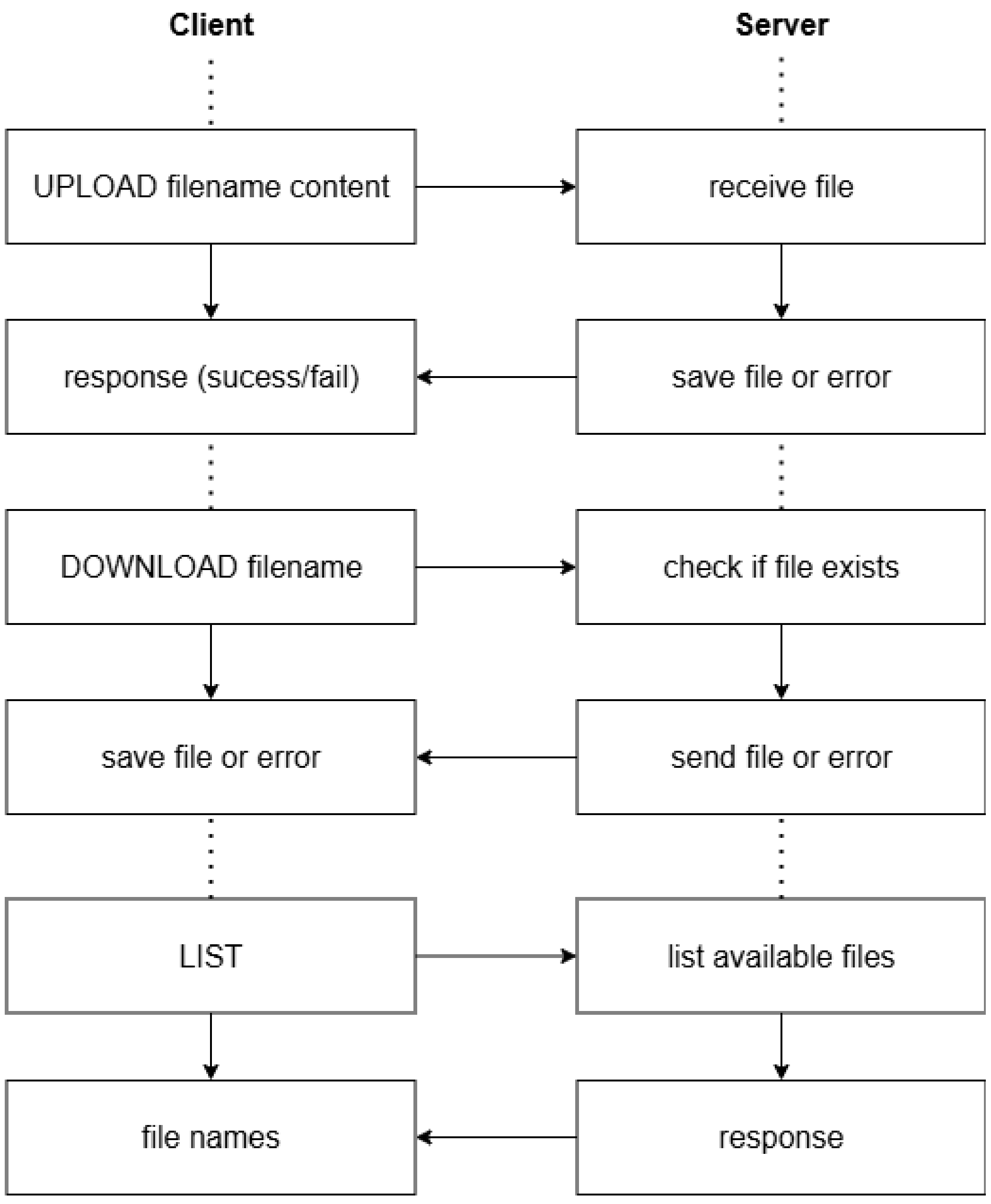

Figure 1: Protocol Design

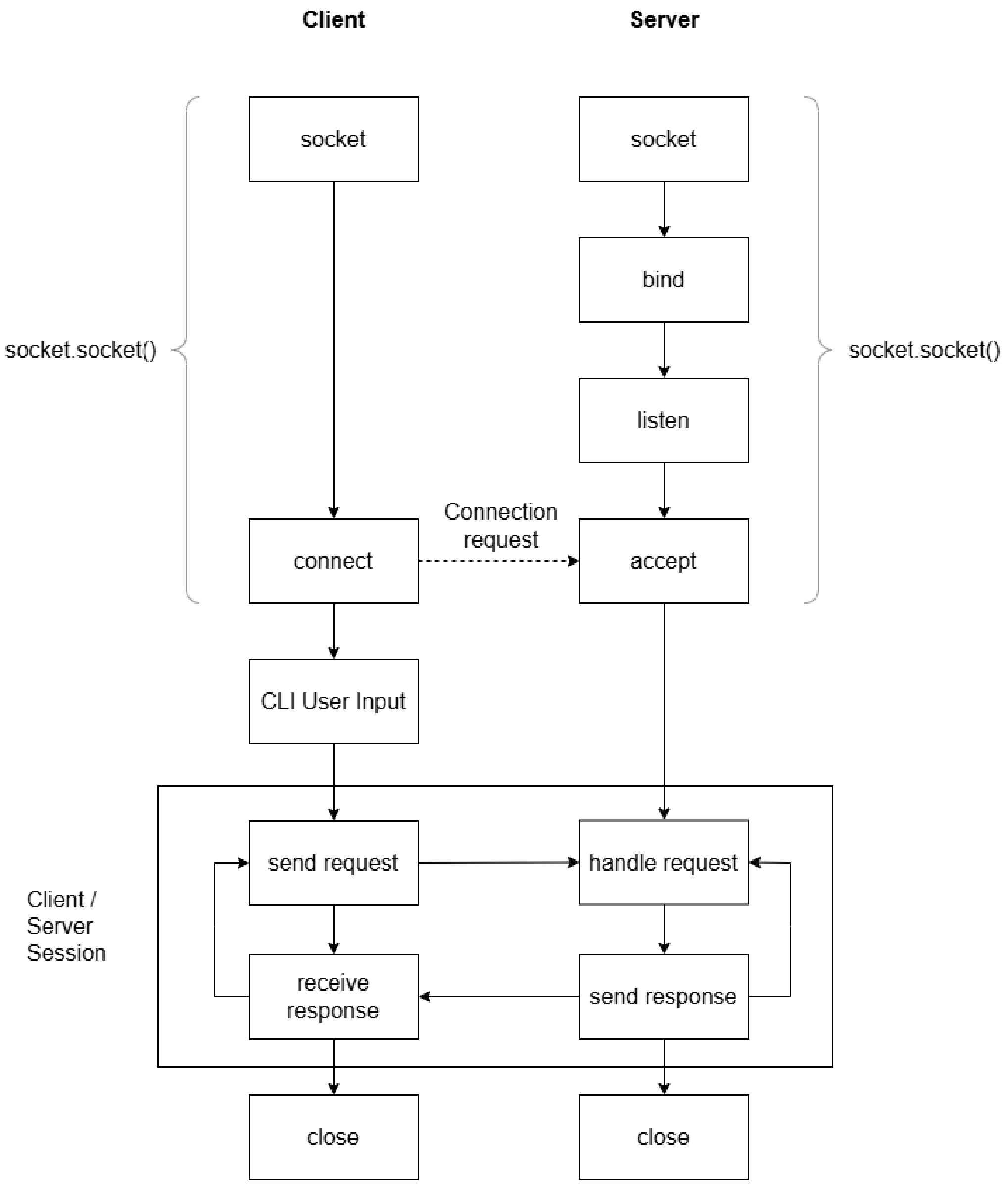

Figure 2: System Organization