TCP File Transfer Protocol

Distributed Systems Practical

1 Protocol Design

Our TCP-based file transfer protocol is designed to allow clients to:

- List available files on the server.
- Request specific files for download.
- Download files reliably using TCP.
- Exit the connection gracefully, signaling the server to shut down if needed.

1.1

Protocol Design Diagram

```
+---------------------------+                          +---------------------------+
|          Client           |                          |          Server           |
+---------------------------+                          +---------------------------+
| 1. Create socket          |                          | 1. Create socket          |
|                           |                          | 2. Bind socket to address |
|                           |                          | 3. Start listening        |
|---------------------------|                          |---------------------------|
| 2. Connect to server      | -----------------------> | 4. Accept connection      |
|                           | Connection Established   |                           |
|                           | <----------------------- |                           |
|---------------------------|                          |---------------------------|
| 3. Receive file list      | <----------------------- | 5. Send file list         |
| (Display files to user)   |                          |                           |
|---------------------------|                          |---------------------------|
| 4. Request file name      | -----------------------> | 6. Check file existence   |
|                           |                          | If exists:                |
|                           |                          |    - Respond "OK"         |
|                           |                          |    - Transfer file chunks |
|                           |                          | Else:                     |
|                           |                          |    - Respond "File Not    |
|                           |                          |       Found"              |
| Receive file or error     | <----------------------- |                           |
|---------------------------|                          |---------------------------|
| 5. Save file locally      |                          |                           |
|                           |                          |                           |
|---------------------------|                          |---------------------------|
| 6. Send "exit" command    | -----------------------> | 7. Close client socket    |
| (Terminate session)       |                          |                           |
|                           |                          |                           |
|---------------------------|                          |---------------------------|
| Close client socket       |                          | 8. Optionally shut down   |
| Exit program              |                          | the server if required.   |
+---------------------------+                          +---------------------------+
```

2 System Organization

The system consists of two components:

1. **Server:** Hosts the files and serves client requests. Runs continuously and manages multiple clients using threads.
2. **Client:** Connects to the server, lists available files, downloads files, and exits when done.

2.1 System Organization Diagram

```
+---------------------+      TCP Connection       +---------------------+
|      Client         |-------------------------->|       Server        |
|                     |                           |                     |
|  1. Create Socket   |<--------------------------|  1. Create Socket   |
|  2. Connect         |                           |  2. Bind & Listen   |
|                     |        File List          |                     |
|  3. Display List    |<--------------------------|  3. Send File List  |
|  4. Request File    |                           |                     |
|---------------------|        File Data          |                     |
|      File Data      |<--------------------------|  4. Transfer File   |
|  5. Save File       |                           |                     |
|  6. Exit Session    |-------------------------->|  5. Close Socket    |
+---------------------+                           +---------------------+
```

3 File Transfer Implementation

The server and client are implemented using Python's `socket` library. Code snippets of the implementation are shown below.

3.1 Server Code

```python
def handle_client(client_socket, client_address):
    files = os.listdir(SERVER_DIRECTORY)
    file_list = "\n".join(files)
    client_socket.send(file_list.encode())

    while True:
        requested_file = client_socket.recv(BUFFER_SIZE).decode().strip()
        if requested_file.lower() == 'exit':
            break

        file_path = os.path.join(SERVER_DIRECTORY, requested_file)
        if os.path.exists(file_path):
            client_socket.send("OK".encode())
            with open(file_path, 'rb') as f:
                while chunk := f.read(BUFFER_SIZE):
                    client_socket.send(chunk)
        else:
            client_socket.send("ERROR: File not found.".encode())

    client_socket.close()
```

Listing 1: Server File Handling

3.2 Client Code

```
def start_client():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((SERVER_HOST, SERVER_PORT))

    file_list = client_socket.recv(BUFFER_SIZE).decode()
    print("Available files:\n" + file_list)

    while True:
        requested_file = input("Enter the file name to download (or 'exit' to quit): ").
            strip()
        client_socket.send(requested_file.encode())
        if requested_file.lower() == 'exit':
            break

        response = client_socket.recv(BUFFER_SIZE).decode()
        if response == "OK":
            file_path = os.path.join(CLIENT_DIRECTORY, requested_file)
            with open(file_path, 'wb') as f:
                while chunk := client_socket.recv(BUFFER_SIZE):
                    f.write(chunk)
        else:
            print(response)

    client_socket.close()
```

Listing 2: Client File Request

4 Responsibilities

- **Server:** Handles client connections, provides the list of files, and sends requested files.
- **Client:** Lists available files, requests files for download, and manages local storage.