
University of Kelaniya

# Peer-to-peer (P2P) file-sharing application using Socket Programming


| Student Number | Student Name |
|---|---|
| CS/2021/052 | Fernando P. T. V. A |
| CS/2021/029 | Fernando W. S. L |


CSCI 21023 - Data Communication and Networks

# The design and architecture of the application:

## Architecture Overview:

- **Server-Side (FileServer Class)**

**Role**: Acts as a central point for clients to connect, request, upload, or list files.

**Core Components**:

- **ServerSocket**: Listens for incoming client connections on a specified port (12345).
- **ClientHandler (Runnable)**: A dedicated thread for each connected client to handle multiple clients simultaneously. The server starts a new ClientHandler thread each time a client connects, ensuring multithreaded processing.
- **ClientSockets List**: Maintains a list of connected client sockets for future expansion (e.g., broadcasting, or multi-client messaging).

- **Client-Side (FileClient Class)**

**Role**: Connects to the server and allows users to send files, request files, list available files, or terminate the connection.

**Core Components**:

- **Socket**: Connects to the server's IP address and port.
- **DataOutputStream** and **DataInputStream**: Manage communication with the server for sending and receiving file data, requests, and responses.
- **BufferedReader**: Captures user input to interactively control the client's functionality.

## Key Design Considerations:

- **Multithreaded Server:** Each client connection spawns a new ClientHandler thread, which enables simultaneous file operations from multiple clients without blocking others.
- **Communication Protocol:** The communication between client and server follows a simple protocol where specific commands trigger corresponding actions on the server. Commands (send, get, list, and exit) are sent as UTF strings, with the server processing each command within the ClientHandler's run() method.

- **Error Handling:** Basic error handling is implemented for file transfer, server connection issues, and client command parsing. The server and client print messages for file operations and errors, assisting with debugging and monitoring during execution.

- **Expandability:** The design can be expanded to include features such as file deletion, user authentication, or encrypted file transfer for secure communication.

## Core Features and Their Interaction:

- **File Transfer (Send and Receive):**

The client initiates a file upload by sending the command "send". The server expects a file name and file size, then reads the file data from the client and saves it in the server's storage directory.

The client requests a specific file from the server using the command "get" followed by the file name. If the file exists, the server streams it to the client, which then saves it to its local storage.

- **File List Retrieval:**

The client sends the command "list" to retrieve available files on the server. The server responds with a list of file names in the directory.

- **Client-Server Communication**:

The client sends commands to the server via DataOutputStream and reads responses via DataInputStream. For file transfer, both the server and client use efficient buffered reading and writing (4096-byte chunks) for stable, memory-efficient performance.

## Challenges faced and how they were resolved:

1. Allowing multiple clients to connect to the server and perform file operations (upload, download, list) at the same time without interfering with each other's requests.

   **Solutions:**
   - Implemented a multithreaded server using the ClientHandler class, which extends Runnable.
   - Each time a client connects, the server starts a new ClientHandler thread to handle the client's requests independently.

2. Errors could occur during file operations, such as if a file doesn't exist on the server, network interruptions, or IO exceptions.

   **Solutions:**
   - Implemented error handling in critical sections.
   - Both the server and the client provide useful messages when working with files by pri nting information about the presence or absence of files.

3. Maintaining a clear and consistent communication protocol between the client and server to handle various commands (send, get, list, exit).

   **Solutions:**
   - The process is clearly defined and each command results in a specific response from t he server.
   - The server uses if statements to check the command received and performs the appropriate action.

4. Provide the client with a complete list of files and let the client know when the list transmissi on is complete.

   **Solutions:**
   - After listing all files, the server sends an "END" message to signal the end of the list.
   - The client reads each file name until it encounters "END", knowing that the list is complete.

# Instructions on how to run the application:

## Prerequisites:

1. Java Development Kit (JDK) installed.
2. Java IDE (e.g., IntelliJ, Eclipse) or command-line environment.

## Instructions:

1. Compile both the **FileServer.java** and **FileClient.java** files using java compiler.
2. Run the **FileServer.java** file. Once the file starts to run, the server will print "**File Server started...**".

```
"C:\Program Files\Java\jdk-1.8\bin\java.exe" ...
File Server started...
```

3. Run the **FileClient.java** file.
4. Once both files start to run in the environment, you will work on the client side through **FileClient.java** file.
5. There are a set of commands when you need to work on the client side.

    - `send`: Upload a file from the client to the server.
    - `get`: Download a specified file from the server.
    - `list`: Display the list of files available on the server.
    - `exit`: Terminate the connection with the server.

**To upload a file to the server:**

1. Type `send` and press **Enter**.
2. You will be prompted to enter the file path of the file you want to upload (e.g., `C:\path\to\file.txt`).
3. The client will upload the file in chunks to the server. If successful, it will display `File sent: <filename>`.

```
"C:\Program Files\Java\jdk-1.8\bin\java.exe" ...
Enter command (To send a file: Enter 'send'/ To retrieve a file: Enter 'get'/ To list available files: Enter 'list'/ To terminate the process: Enter 'exit'):
send
Enter file path to send:
C:\Users\thisa\Downloads\example02.txt
File sent: example02.txt
```

**To Download a File from the Server**:

1. Type `get` and press **Enter.**
2. Enter the exact name of the file you want to retrieve.
3. If the file exists on the server, it will be downloaded and saved in the client's directory with a `client_` prefix. If the file does not exist, an error message will display.

```
Enter command (To send a file: Enter 'send'/ To retrieve a file: Enter 'get'/ To list available files: Enter 'list'/ To terminate the process: Enter 'exit'):
get
Enter file name to request:
server_example02.txt
Received file: client_server_example02.txt
```

**To View the List of Available Files on the Server**:

1. Type `list` and press **Enter**.
2. The server will send the list of available files to the client, which will display each file name line by line.

```
Enter command (To send a file: Enter 'send'/ To retrieve a file: Enter 'get'/ To list available files: Enter 'list'/ To terminate the process: Enter 'exit'):
list
Files available:
.gitignore
client_server_example.txt
client_server_example02.txt
client_server_example03.txt
P2P_File_Sharing_Application.iml
server_        try {
server_Downloads
server_example.txt
server_example02.txt
server_example03.txt
server_send
```

**Exit the Application:**

1. Type `exit` and press **Enter**.
2. The client will terminate the connection to the server, and you can close the client's terminal or console.

```
Enter command (To send a file: Enter 'send'/ To retrieve a file: Enter 'get'/ To list available files: Enter 'list'/ To terminate the process: Enter 'exit'):
exit

Process finished with exit code 0
```