



Topic: P2P File Transfer App

Name: Mahbub Anam

ID: 2411731642

Course: CSE215

Section: 13 Faculty: MUO

Semester: Summer 24

Project Report: P2P File Transfer App

Abstract:

The Peer-to-Peer (P2P) File Transfer System is the solution for the effective and safe file exchange between the users in the network. It has both the server and the client part, where the server takes responsibility for receiving and storing files, reconstructing files and the client is responsible for splitting files, transmitting and handling errors. It provides for segmentation of files, dynamic pairing codes for client authentication and utilizes md5 algorithm for verifying data integrity. The system is designed for parallel or simultaneous transfers through multi-thread mode for increased capability. Developed in Java, it also applies key concepts like Object-Oriented Programming (OOP), multi-threading, network I/O, and file I/O.

Tools and Technologies :

Programming Language: Java

Frameworks: Java Swing for GUI development

Networking: Java Sockets for communication between peers

File Handling: File I/O for storing files.

Metadata : Google's Gson library for json file parsing & MD5 algorithm for hashing.

Tools: IntelliJ IDEA IDE for development.

System Design:

The system is designed using Object-Oriented Programming (OOP) principles to manage peer-to-peer file transfers efficiently. The following are the key components:

1. **FileServer:** The server-side component that listens for incoming client connections, verifies the pairing code, and handles file reception and reconstruction.
 2. **FileClient:** The client-side component that connects to the server, handles file splitting into smaller chunks, and sends the chunks to the server.
 3. **Utils:** A utility class that provides helper methods for generating random pairing codes and validating IP addresses.
 4. **FileServ:** A service class that handles the splitting and rebuilding of files, ensuring that files can be divided into multiple parts and reassembled correctly.
-

Project Structure:

Class Name	Attributes	Methods
FileServer	port: int serverSocket: ServerSocket CHUNK_SIZE: int stat: boolean pathToSave: String	start(): void stop(): void handleClient(Socket): void receiveFile(DataInputStream, DataOutputStream, File, long): void logMessage(String): void
FileClient	serverAddress: String serverPort: int code: int	setCode(int): void sendFiles(String[] filePaths, String fname, int fileParts): void sendFile(String filePath): void getPaths(String PathT, int segments): String[]
FileServ	Path: String segmentCount: int CHUNK_SIZE: int	split(): void rebuild(String fileName, String filePathToSave): void deleteS(String tPathofFile, int parts): void bytesToHex(byte[] bytes): String
Utils	None	randomCode(): int isValidIP(String ip): boolean

ReceiverGUI	pairCodeField: JTextField saveFolderField: JTextField console: JTextArea randomCode: int path_selected: String fileServer: FileServer serverThread: Thread startButton: JButton	createAndShowGUI(): void startServer(): void SetMsg(String msg): void
SenderGUI	console: JTextArea	createAndShowGUI(): void validateAndSend(String ip, String filePath, int threads, String pairCode): void log(String message): void
FileTrasnsfer	None	FileTrasnsfer(): void createAndShowGUI(): void main(String[] args): void

Relationships & workflow :

1. **FileServer Class** communicates with the **FileClient** to handle file transfers.
 2. **FileServer** uses the **FileServ** class for file operations like receiving and rebuilding files.
 3. **FileClient** works with the **FileServ** class to split large files and send them in parts.
 4. **FileServer** and **FileClient** both interact with **GUI classes** (FileTransfer, SenderGUI and ReceiverGUI) to provide a graphical interface for users.
 5. **FileServ** handles file operations like splitting, rebuilding, and deleting segments of files.
 6. **Utils** provides helper methods for the system, like generating random codes and validating IP addresses.
-

Highlights :

During project development I faced some difficulties & while solving them I learned some new topics too. I'm listing them here :

1. Rebuilding issues :

While writing the FileServ class (It's a short form for File Service) I encountered a weird bug which was a segmentation issue. Each file was missing around 1024 bytes of data from the beginning of the file. Changing the read/write size higher than 1024 solved it. Java often messes up if you read/write with a small amount of buffers.

2. EOF Handling :

As I was using multiple threads to send multiple files, I needed to tell the server to close the connection & start rebuilding when all the parts were sent. But in multi threading each thread doesn't run by order, it runs by memory usage. So, the end signal was sent before the file transfer was even completed. After doing some research I learned about Java latch, which makes thread wait until a thread is completed. So I implemented it & finally the problem was solved, Here's the implementation snapshot:

```
// ref : https://www.baeldung.com/java-countdown-latch
public void sendFiles(String[] filePaths, String fname, int fileParts) {
    // latch insertion with total file count as thread count
    CountDownLatch latch = new CountDownLatch(filePaths.length);
    for (String filePath : filePaths) {
        new Thread(() -> {
```

```

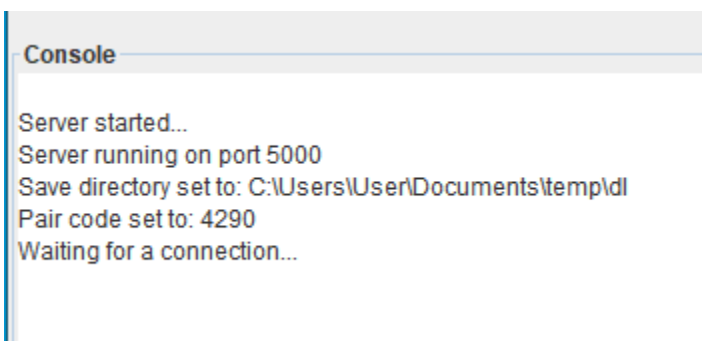
        try {
            sendFile(filePath);
        } finally {
            latch.countDown(); //after starting file thread add countdown
        }
    }).start();
}

// end thread that sends EOF to server as -9999
new Thread(() -> {
    try {
        latch.await(); //start when other threads are finished
        try {
            // rest of my code
        } catch (Exception e) {
            e.printStackTrace();
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}).start();
}

```

3. Printing Status message to GUI :

It took me some time to figure out that I can keep the console public and pass a log function to my backend classes to show on GUI.



The screenshot shows a 'Console' window with the following text:

```

Server started...
Server running on port 5000
Save directory set to: C:\Users\User\Documents\temp\dl
Pair code set to: 4290
Waiting for a connection...

```

It finally worked after some digging

TO-DO (Future development of this project) :

1. Implement encryption.
2. Developing an android application. (Under development)
3. Multi threading based file splitting.
4. Check available space before splitting.
5. Progressbar GUI.
6. More efficient error handling (Current one has some lackings)

All the updates can be found here : <https://github.com/Rhythm113/CSE215L.git>

Submitted by :

Mahbub Anam
2411731642
CSE215L.13