

CSE 6324-002: Advanced Topics in Software Engineering

Report

Concurrent Programming Project

Instructor

Jeff Lei

Team: 03

Mansi Patel	1001874295
Harshada Padwal	1002080814
Niral Shah	1002061390
Dishant Koli	1002067615

TABLE OF CONTENT

Sr No.	Title	Page No.
1.	Introduction	2
2.	Major Functionality	2
3.	Thread Management (Client Flow, Server Flow)	3
4.	Test Plan	7
5.	Test Results	8
6.	Challenges	9
7.	Future Scope	10
8.	Code Snippets	11
9.	Members Contributions	14
10.	Learnings	15

Introduction:

The purpose of this project is to develop a concurrent programming project using java a project which like the google drive/drop box/or any other cloud storage. The application allows multiple clients to communicate with a single server using TCP for connection establishment and UDP for file transfer. The project utilizes multiple threads to achieve concurrency and efficient file synchronization.

Major Functionalities:

- **File Uploads:** Clients can upload files to the server. The files are divided into packets of 4 bytes each by the FileReader thread. These packets are deposited into a buffer monitor. The PacketSender thread then withdraws the packets from the buffer monitor and sends them to the server's UDP port for storage.
- **File Delete:** Clients can delete files from the server. The ClientHandler thread on the server side handles each client's request to delete a file. Upon receiving a delete request, the server removes the corresponding file from storage.
- **Delta Sync:** The application supports delta synchronization, which means that only the changes or modifications made to a file are synchronized instead of transferring the entire file. This optimization minimizes data transfer and improves synchronization efficiency.
- **Sync Status:** The application provides sync status information to clients, indicating whether files are in sync or out of sync with the server. Clients can check the sync status to ensure the integrity of their files.

- **Error Handling:** Proper error handling mechanisms are implemented to handle exceptions and ensure the reliability of file synchronization. Error messages are displayed to clients in case of any errors during file transfer or synchronization.

Thread Management:

Client Thread:

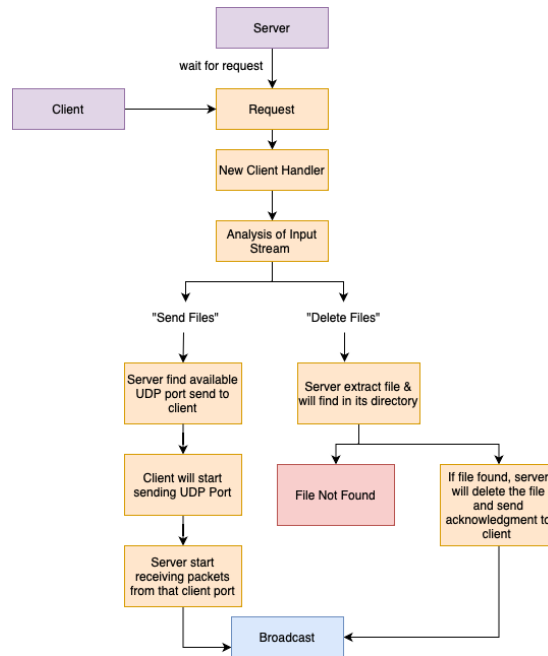
- **FileReader:** This thread reads files from the client's local storage and creates packets of size 4 bytes. The packets are then deposited into the buffer monitor.
- **PacketSender:** This thread withdraws packets from the buffer monitor and sends them to the server's UDP port for storage. FileReader and PacketSender threads synchronize their execution using the join() method.
- **BroadcastReceiver:** This thread listens for broadcast messages from the server, which contain updates or events related to file synchronization. It ensures that the client stays updated with the changes made by other clients.

Server Thread:

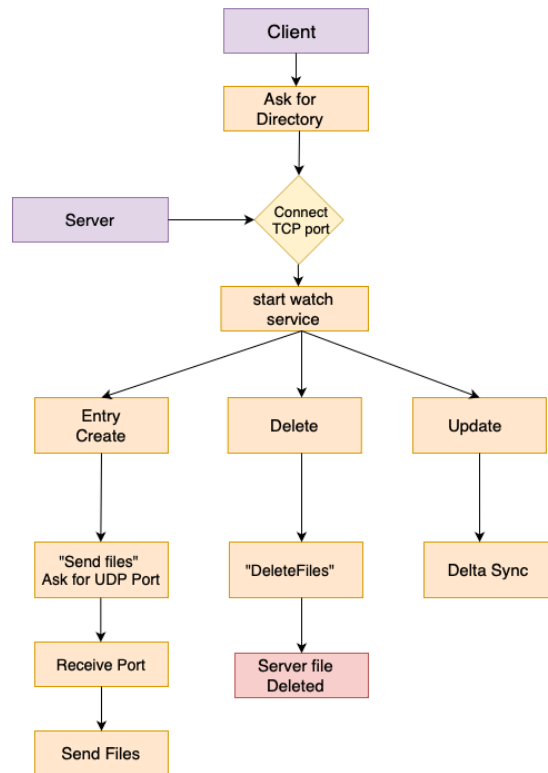
- **ClientHandler:** This thread handles each client's requests on the server side. It processes file upload and delete requests, communicates with the client's PacketSender thread, and manages synchronization.
- **PacketReceiver:** This thread receives packets sent by the client's PacketSender thread and deposits them into the buffer monitor.
- **FileWriter:** This thread withdraws packets from the buffer monitor and writes the packet contents to the corresponding file on the server. PacketReceiver and FileWriter threads synchronize their execution using the join() method.
- **BroadcastSender:** This thread broadcasts events related to file creation or deletion to other clients, ensuring synchronization across multiple clients.

- **BroadcastReceiver:** This thread listens for broadcast messages from other clients, which contain updates or events related to file synchronization. It ensures that the server stays updated with the changes made by other clients.

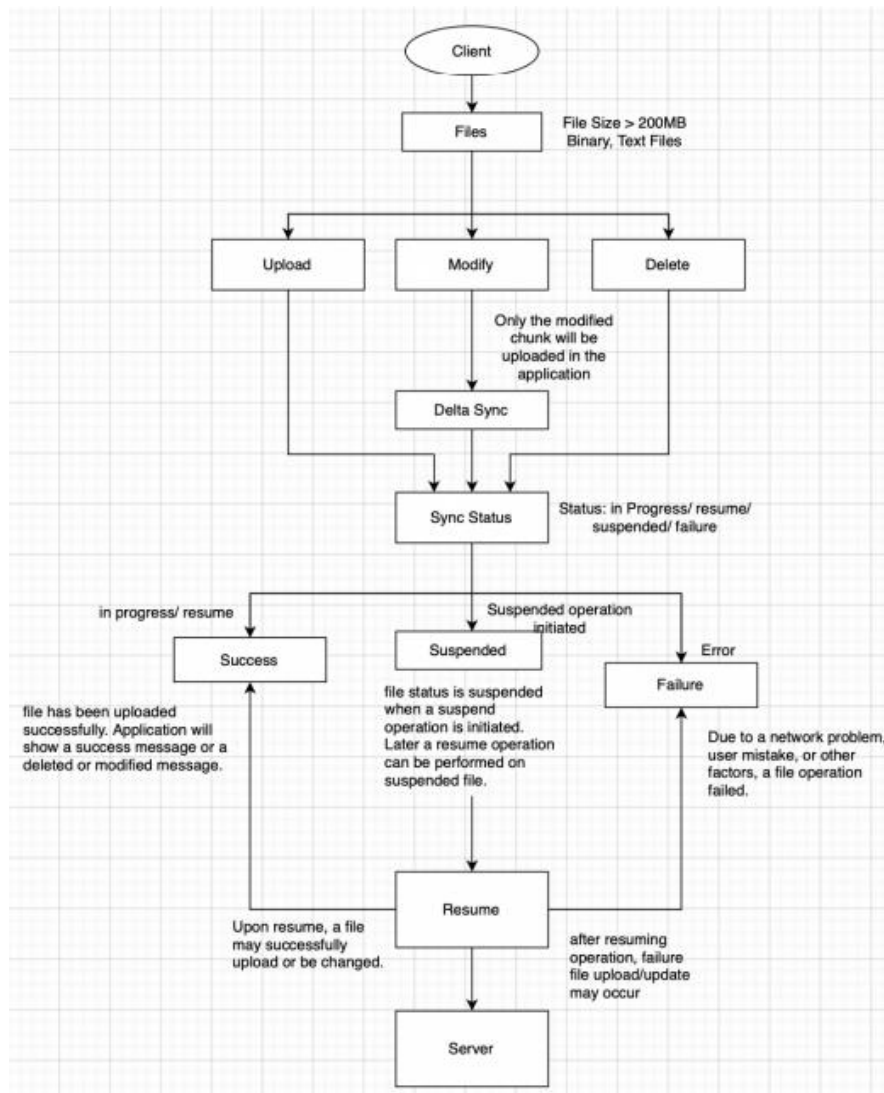
Client Flow:



Server Flow:



High Level Design:



Test Plan:

Following are the elaborated test cases for major use case like Sync file, Delta sync, Sync status, Error handling:

Test Case ID	Test Case Description	Precondition	Expected Output
TC1 - File Sync (File Size)	The system should support. Large file (>200mb) as well as text and binary file	The file can be large. (>200MB)	Displays as “File is sent”
TC2 - Sync status (Upload Status)	The file will either sync or fail to sync after being processed. The system should display whether the file is synced or not.	The file should be processed.	Display as “File is in sync” or “File is out of sync”
TC3 - Error Handling (File Status)	The system shall maintain the file's status in case of network failure or client intervention	File should be in syncing process	There should be no data loss.
TC4 - File Transfer	The system shall resume file transfer from the block where it stopped in case of a network failure or client intervention.	System should be in the file transfer process.	File is successfully transferred.
TC5 - File Deletion	The system shall successfully delete the file from the server if the deletion is not halted by client action or a network issue.	The file should be present in the system.	Display as “File is successfully deleted”

TC6 - Reflect Changes	The system shall reflect any changes made to a file at the client end in the file uploaded to the server.	Upload the file.	The changes are reflected at the client side.
TC7 - Show Error messages	The system shall show an error message	An error occurs.	An error pops up whenever it occurs.
TC8 - Delta Sync	Compress the data that needs to be sync.	Upload a large file	The file is modified at compressed size.

Test Results:

The application was tested extensively to ensure proper functionality and synchronization of files across multiple clients. Various scenarios, including file uploads, file deletion, delta sync, sync status checks, and broadcast message handling, were tested to validate the accuracy, efficiency, and reliability of the synchronization process.

Challenges:

During the development of this project, we encountered several challenges, including:

- **Concurrent programming:** Managing multiple threads and ensuring proper synchronization was a complex task.
- **Port management:** Assigning UDP ports for multiple file uploads while avoiding conflicts required careful handling.
- **Syncing clients via the server:** Ensuring that all clients receive updates and stay in sync with the server posed a challenge.
- **Broadcasting changes:** Broadcasting file creation or deletion events to all connected clients while maintaining efficiency was challenging.
- **Delta Sync:** Implementing an optimized mechanism to synchronize only the modified parts of a file instead of transferring the entire file presented a challenge.
- **Handling undefined errors:** Dealing with unforeseen errors and exceptions during file synchronization required thorough error handling and debugging.
- **Handling small/large files:** Efficiently handling file synchronization for both small and large files, while maintaining performance, was a challenge.
- **Time constraint:** Completing the project within the given time frame was a challenge, requiring effective time management and prioritization.

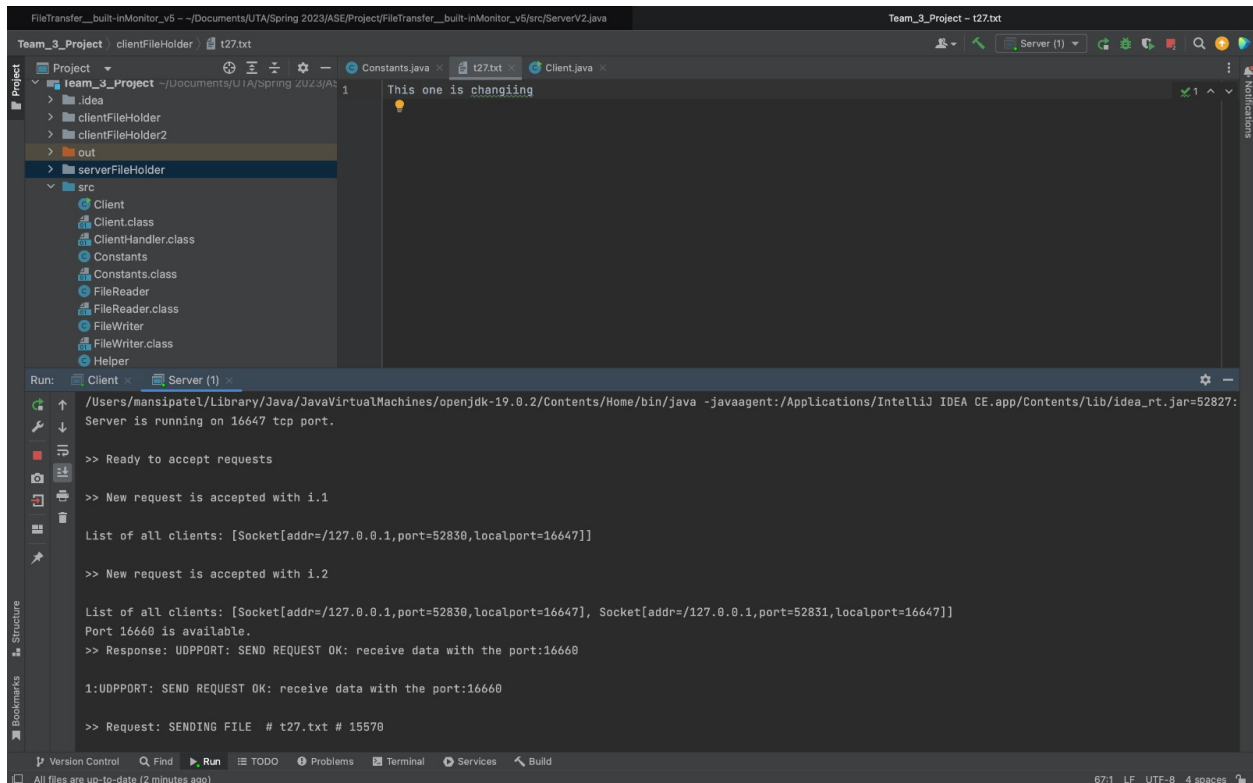
Future Scope:

In the future, the project can be enhanced in the following ways:

- **Design UI:**
Provide a intuitive and user-friendly experience for accessing and managing files in the cloud.
- **Decrease response time and latency:**
Optimize the application's performance to reduce response time and latency during file synchronization, providing a faster and smoother user experience.
- **Authentication:**
Implement a robust authentication system to ensure secure access to files and prevent unauthorized access to sensitive data.

By addressing these areas, the cloud-based Java application can be further improved and expanded to meet the evolving needs of users for efficient file synchronization and management.

Code Snippets:



The screenshot shows the IntelliJ IDEA interface. The top toolbar includes icons for Run, Debug, and other IDE functions. The main editor area displays the file `t27.txt` with the content `This one is changing`. The left sidebar shows the project structure for `Team_3_Project`, with the `src` directory expanded, listing files like `Client`, `Client.class`, `ClientHandler.class`, `Constants`, `Constants.class`, `FileReader`, `FileReader.class`, `FileWriter`, `FileWriter.class`, and `Helper`. The bottom panel shows the Run configuration for `Server (1)` and the output console. The console output indicates the server is running on port 16647, accepts requests, and sends responses.

```
Run: Client x Server (1) x
/Users/mansipatel/Library/Java/JavaVirtualMachines/openjdk-19.0.2/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=52827:
Server is running on 16647 tcp port.

>> Ready to accept requests
>> New request is accepted with i.1
List of all clients: [Socket[addr=/127.0.0.1,port=52830,localport=16647]]
>> New request is accepted with i.2
List of all clients: [Socket[addr=/127.0.0.1,port=52830,localport=16647], Socket[addr=/127.0.0.1,port=52831,localport=16647]]
Port 16660 is available.
>> Response: UDPPORT: SEND REQUEST OK: receive data with the port:16660
1:UDPPORT: SEND REQUEST OK: receive data with the port:16660
>> Request: SENDING FILE # t27.txt # 15570
```

Upload for server

```
FileTransfer_built-inMonitor_v5 - ~/Documents/UTA/Spring 2023/ASE/Project/FileTransfer_built-inMonitor_v5/src/ServerV2.java
Team_3_Project - t27.txt
Project: Team_3_Project
  .idea
  clientFileHolder
  clientFileHolder2
  out
  serverFileHolder
  src
    Client
    Client.class
    ClientHandler.class
    Constants
    Constants.class
    FileReader
    FileReader.class
    FileWriter
    FileWriter.class
    Helper
Run: Client x Server (1) x
/Users/mansipatel/Library/Java/JavaVirtualMachines/openjdk-19.0.2/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=52824:
Enter Client Number: 1
Connected with Server
Multicast Receiver Started.....
0:1
1:UDPPORT
2: SEND REQUEST OK
3: receive data with the port
4:16660
[Ljava.lang.String;@1b2c6ec2
16660
Port 15570 is available.
>> Response: SENDING FILE # t27.txt # 15570

>> Begin to send packets

>> Begin to read a file
```

Upload for client

```
FileTransfer_built-inMonitor_v5 - ~/Documents/UTA/Spring 2023/ASE/Project/FileTransfer_built-inMonitor_v5/src/ServerV2.java
Team_3_Project - t27.txt
Project: Team_3_Project
  .idea
  clientFileHolder
  clientFileHolder2
  out
  serverFileHolder
  src
    Client
    Client.class
    ClientHandler.class
    Constants
    Constants.class
    FileReader
    FileReader.class
    FileWriter
    FileWriter
Run: Client x Server (1) x
>> Receive the packet with index 0
Send an ACK packet for packet 0

>> Prepare to write the file t27.txt

>> Receive the packet with index 1
Send an ACK packet for packet 1

>> Write to a file the packet with index 1
>> Receive the packet with index -1
Send an ACK packet for packet -1

>> Finish receiving packets
>> Finish saving the file:t27.txt
>> Packet indices: [1]
Multicast Publisher Started.....
MulticastMessage:New_File_Uploaded:t27.txt:2
```

Multicast sending of server

The screenshot shows an IDE window for a project named 'Team_3_Project'. The project structure includes a 'serverFileHolder' directory with a 't27.txt' file. The terminal output shows the following sequence of events:

```
>> Write to a file the packet with index 1
>> Receive the packet with index -1
    Send an ACK packet for packet -1

>> Finish receiving packets
>> Finish saving the file:t27.txt
>> Packet indices: [1]
Multicast Publisher Started.....
MulticastMessage:New_File_Uploaded:t27.txt:2
File_name_of_modified_File:t27.txt:a2f44feec98ae29fdb2d0d5fc7a30a46:[8048533e64
t27.txt
dd919df1397be2189fee120a8c36cc7e
Needed modification
Start: 9
Modified Block: [111, 110, 101, 32, 105, 115, 32, 99, 104, 110, 97, 103, 105, 110, 103, 103, 32, 99, 104, 97, 110, 103, 105, 105, 110, 103]
Modification at server is completed
Multicast Publisher Started.....
```

Delta sync of server

Members Contributions:

- **Dishant Koli** - Contributed to the creation of the client-side of the project, which included creating the user interface and adding functionality for file synchronization. Designing a dependable file synchronization system and making sure the user interface was user-friendly and aesthetically pleasing. This needed proficiency in client-side development-related programming languages and technologies, as well as a sharp eye for design and user experience.
- **Niral Shah** - Contributions in the areas of network protocol implementation and delta synchronization implementation, in the client-side development of the project. This required putting in place systems to recognize and send just the portions of files that had changed since the last sync, as well as making sure that data was transferred between the client and server smoothly and effectively. Overall, this improved the client-side portion of the project's functionality and performance.
- **Harshada Padwal** - Worked on managing the file system and network connections in order to contribute to server-side development. Maintaining the server's file organization and making sure the network connections were reliable. Essential to the server's smooth operation made sure that the clients could quickly access the files and that the network connections were reliable and secure.
- **Mansi Patel** - Contributed significantly to the project's server-side development, concentrating on establishing the file transfer protocol and guaranteeing seamless file transfers between the client and server. In addition, was in charge of the critical duty of testing and debugging, making sure that any problems with the system were found and fixed right away, guaranteeing that the project was dependable and satisfied the needs of its customers.

Lessons Learned:

- **Concurrent Programming** - In order to improve system speed and boost efficiency, concurrent programming is a programming approach used to create programs that can run many tasks or processes concurrently. It entails the use of coding techniques and technologies that permit a number of threads or processes to execute concurrently while sharing resources and avoiding problems like deadlocks and race situations.
- **Port Management** - In order to maintain the safe and effective movement of freight and passengers into and out of ports, port management refers to the administration of ports and associated facilities, including maintenance, security, and operations.
- **Syncing clients via the server**
- **Broadcasting changes**
- **Delta Sync** - A file synchronization technique called delta sync, also referred to as delta synchronization, detects and sends only the changes made to a file—not the full file. By dramatically reducing the amount of data that needs to be sent across devices, this can speed up syncing, consume less bandwidth, and need less storage. Cloud storage services and other programs that need dependable and effective file synchronization frequently employ delta sync.
- **Working with multiple threads to achieve concurrency**