THE UNIVERSITY OF TEXAS AT ARLINGTON

Name :- Avinash Mahala(1002079433)
& Amit Munna Gupta(1002066302)
Semester :- Summer 2023
Course :- Distributed Systems(CSE-5306)

Remote Procedure Call Project Report

Contents

1. Assumptions We have made:-

1. Uploading a File Goes to "**server_uploads**" directory{Act as a Server Directory}.
2. Downloading a File Saves the File from the Server to The Client in "**client_downloads**" directory{Acts as a Client Directory}.

2. Description of How Code Works

Here We have 9 functionalities:-

1. UPLOAD: Upload a file to the file server.
2. DOWNLOAD: Download a file from the file server.
3. DELETE: Delete a file from the file server.
4. RENAME: Rename a file on the file server.
5. ADD: Perform addition of two numbers using the computation server.
6. SORT: Sort an array of records using the computation server.
7. START AUTO SYNC: Start automatic synchronization at regular intervals.
8. STOP AUTO SYNC: Stop automatic synchronization.
9. EXIT: Exit the program.

WorkFlow 1: UPLOAD A FILE:-

Description: The "Upload a File" workflow allows users to upload a file from the client-side program to the server.

Steps:

1. User selects the "UPLOAD" option from the client-side program's menu.
2. User is prompted to enter the file path of the file they want to upload.
3. The client-side program reads the file from the specified path.
4. The program extracts the file name and reads the file data.
5. The file data is encoded using base64 encoding to ensure proper transmission.
6. The client-side program establishes a connection with the server-side program using Pyro4.
7. The client-side program invokes the "upload" method on the server-side program, passing the file name and encoded file data as parameters.
8. The server-side program receives the file name and encoded data.
9. The server-side program decodes the file data from base64.
10. The program saves the file on the server, either by creating a new file or overwriting an existing one with the same name.
11. The server-side program returns a success or failure indication to the client-side program.
12. The client-side program displays an appropriate message to the user based on the server's response.
13. If the upload is successful, the user is informed that the file has been uploaded successfully.
14. If there is an error during the upload process, the user is notified of the failure.

Key Considerations:

- **Error handling:** The client-side program should handle any exceptions that occur during file reading, encoding, or network communication.
- **File validation:** The client-side program can perform checks to ensure the file exists and is accessible before initiating the upload process.
- **File overwrite:** If a file with the same name already exists on the server, the user should be informed and given the option to confirm the overwrite or choose a different name.
- **Progress indication:** For large files, it might be beneficial to provide progress indicators to the user to track the upload progress.
- **Network stability:** Both the client-side and server-side programs should handle network disconnections or failures gracefully and provide appropriate error messages to the user.

Overall, the "Upload a File" workflow enhances collaboration by allowing users to share files with other clients connected to the server. It facilitates file transfer and storage in a distributed system environment.

WorkFlow 2: DOWNLOAD A FILE:-

Description: The "Download a File" workflow enables users to download a file from the server to their local machine using the client-side program. The user can retrieve files that have been previously uploaded to the server.

Steps:

1. User selects the "DOWNLOAD" option from the client-side program's menu.
2. User is prompted to enter the name of the file they want to download.
3. The client-side program establishes a connection with the server-side program using Pyro4.
4. The client-side program invokes the "download" method on the server-side program, passing the file name as a parameter.
5. The server-side program receives the request to download the file.
6. The server-side program checks if the requested file exists on the server.
7. If the file exists, the server-side program reads the file data.
8. The file data is encoded using base64 encoding to ensure proper transmission.
9. The server-side program sends the encoded file data back to the client-side program.
10. The client-side program receives the file data.
11. The client-side program decodes the file data from base64.
12. The program saves the file on the client's local machine, either by creating a new file or overwriting an existing one with the same name.
13. The client-side program displays an appropriate message to the user based on the success or failure of the download process.
14. If the download is successful, the user is informed that the file has been downloaded successfully.
15. If the requested file is not found on the server or there is an error during the download process, the user is notified of the failure.

Key Considerations:

- **Error handling:** The client-side program should handle any exceptions that occur during network communication, file writing, or decoding.
- **File validation:** The client-side program can perform checks to ensure the file name is valid and exists on the server before initiating the download process.
- **File overwrite:** If a file with the same name already exists on the client's local machine, the user should be informed and given the option to confirm the overwrite or choose a different name.
- **Progress indication:** For large files, it might be beneficial to provide progress indicators to the user to track the download progress.
- **Network stability:** Both the client-side and server-side programs should handle network disconnections or failures gracefully and provide appropriate error messages to the user.

Overall, the "Download a File" workflow allows users to retrieve files from the server, enabling easy access to shared files and promoting collaboration in a distributed system environment.

WorkFlow 3: DELETE A File From Server:-

Description: The "Delete a File" workflow allows users to remove a file from the server using the client-side program. This workflow provides users with the ability to manage the files stored on the server and delete unnecessary or outdated files.

Steps:

1. User selects the "DELETE" option from the client-side program's menu.
2. User is prompted to enter the name of the file they want to delete.
3. The client-side program establishes a connection with the server-side program using Pyro4.
4. The client-side program invokes the "delete" method on the server-side program, passing the file name as a parameter.
5. The server-side program receives the request to delete the file.
6. The server-side program checks if the requested file exists on the server.
7. If the file exists, the server-side program removes the file from the server's storage.
8. The server-side program sends a success message back to the client-side program.
9. The client-side program receives the success message from the server.
10. The client-side program displays an appropriate message to the user based on the success or failure of the delete process.
11. If the file is successfully deleted, the user is notified that the file has been removed from the server.
12. If the requested file is not found on the server or there is an error during the delete process, the user is notified of the failure.

Key Considerations:

- **Error handling:** The client-side program should handle any exceptions that occur during network communication or file deletion.
- **Confirmation prompt:** Before deleting the file, the client-side program can display a confirmation prompt to ensure the user intends to delete the file.
- **File validation:** The client-side program can perform checks to ensure the file name is valid and exists on the server before initiating the delete process.
- **Access control:** Depending on the system's requirements, access control mechanisms can be implemented to restrict file deletion to authorized users only.
- **Data integrity:** The server-side program should handle cases where the file is currently being accessed or modified by another user to prevent data corruption.
- **Logging:** It can be beneficial to log file deletion actions for auditing purposes or troubleshooting.

The "Delete a File" workflow provides users with control over their stored files, allowing them to remove unwanted files and manage storage space efficiently. It promotes file organization and maintenance within the server-side program, ensuring a clean and up-to-date file repository.

WorkFlow 4: RENAME a File in the Server:-

Description: The "Rename a File" workflow enables users to rename an existing file stored on the server using the client-side program. This workflow provides users with the flexibility to update file names according to their requirements, enhancing file organization and management.

Steps:

1. User selects the "RENAME" option from the client-side program's menu.
2. User is prompted to enter the current name of the file they want to rename.
3. User is prompted to enter the new name for the file.
4. The client-side program establishes a connection with the server-side program using Pyro4.
5. The client-side program invokes the "rename" method on the server-side program, passing the current and new file names as parameters.
6. The server-side program receives the request to rename the file.
7. The server-side program checks if the current file name exists on the server.
8. If the file exists, the server-side program renames the file to the new name.
9. The server-side program sends a success message back to the client-side program.
10. The client-side program receives the success message from the server.
11. The client-side program displays an appropriate message to the user based on the success or failure of the rename process.
12. If the file is successfully renamed, the user is notified that the file has been renamed.
13. If the requested file is not found on the server or there is an error during the rename process, the user is notified of the failure.

Key Considerations:

- **Error handling:** The client-side program should handle any exceptions that occur during network communication or file renaming.
- **File validation:** The client-side program can perform checks to ensure the current file name is valid and exists on the server before initiating the rename process.
- **New name validation:** The client-side program can validate the new file name provided by the user to ensure it adheres to any naming conventions or restrictions.
- **Duplicate names:** The server-side program should handle cases where the new file name conflicts with an existing file on the server.
- **File references:** If the renamed file is referenced by other files or systems, appropriate updates or notifications should be implemented to reflect the new file name.
- **Logging:** It can be beneficial to log file renaming actions for auditing purposes or troubleshooting.

The "Rename a File" workflow empowers users to update file names in a server-side program, enhancing file management capabilities and improving the organization of stored files. It offers a user-friendly approach to modify file names, promoting better file identification and retrieval.

WorkFlow 5: ADD 2 Numbers:-

Description: The "Add 2 Numbers" workflow allows users to perform addition operations using the client-side program. This workflow enables users to input two numbers and receive the sum as the result, leveraging the computation capabilities of the server-side program.

Steps:

1. User selects the "ADD" option from the client-side program's menu.
2. User is prompted to enter the first number.
3. User inputs the first number.
4. User is prompted to enter the second number.
5. User inputs the second number.
6. The client-side program establishes a connection with the server-side program using Pyro4.
7. The client-side program invokes the "add" method on the server-side program, passing the two numbers as parameters.
8. The server-side program receives the request to perform the addition operation.
9. The server-side program adds the two numbers together.
10. The server-side program sends the result back to the client-side program.
11. The client-side program receives the result from the server.
12. The client-side program displays the result to the user.

Key Considerations:

- **Input validation:** The client-side program can validate user inputs to ensure they are valid numbers.
- **Error handling:** The client-side program should handle any exceptions that occur during network communication or the addition operation.
- **Data types:** Consider the data types of the numbers being added and ensure compatibility and accurate results.
- **Integer vs. floating-point:** Depending on the requirements, determine whether the addition should be performed on integers or floating-point numbers.
- **Overflow/Underflow:** Take into account potential issues with integer overflow or underflow when dealing with large or small numbers.

The "Add 2 Numbers" workflow provides users with a simple and convenient way to perform addition operations using the client-side program. By utilizing the computational capabilities of the server-side program, users can quickly obtain the sum of two numbers, saving time and effort in manual calculations.

WorkFlow 6: SORT a Given Array of Records:-

Description: The "Sort a Given Array of Records" workflow allows users to sort an array of records using the client-side program. This workflow leverages the computation capabilities of the server-side program to efficiently sort the array and provide the sorted result to the user.

Steps:

1. User selects the "SORT" option from the client-side program's menu.
2. User is prompted to enter the array of records.
3. User inputs the array of records, separating each record with a comma.
4. The client-side program establishes a connection with the server-side program using Pyro4.
5. The client-side program invokes the "sort" method on the server-side program, passing the array of records as a parameter.
6. The server-side program receives the request to perform the sorting operation.
7. The server-side program sorts the array of records using an appropriate sorting algorithm (e.g., quicksort, mergesort, etc.).
8. The server-side program sends the sorted result back to the client-side program.
9. The client-side program receives the sorted result from the server.
10. The client-side program displays the sorted array of records to the user.

Key Considerations:

- **Input validation:** The client-side program can validate user inputs to ensure they are in the correct format and meet any requirements.
- **Error handling:** The client-side program should handle any exceptions that occur during network communication or the sorting operation.
- **Sorting algorithm:** Choose an appropriate sorting algorithm based on the requirements and characteristics of the records in the array.

- **Record structure:** Define the structure of the records and ensure consistency in the client-side and server-side programs.
- **Sorting order:** Determine the desired sorting order (e.g., ascending or descending) and implement the sorting algorithm accordingly.

The "Sort a Given Array of Records" workflow provides users with a convenient way to sort an array of records using the client-side program. By leveraging the computational capabilities of the server-side program, users can quickly obtain the sorted result without having to implement the sorting logic themselves. This workflow is particularly useful when dealing with large datasets or complex sorting requirements.

WorkFlow 7: START AUTO SYNC:-

Description: The "Start Auto Sync" workflow enables users to initiate automatic synchronization between the client-side program and the server-side program. This feature ensures that any changes made on either side are automatically reflected in the other, providing a seamless and up-to-date file synchronization mechanism.

Steps:

1. User selects the "START AUTO SYNC" option from the client-side program's menu.
2. The client-side program checks if automatic synchronization is already in progress.
3. If automatic synchronization is already active, the client-side program displays a message indicating that the sync is already in progress.
4. If automatic synchronization is not currently active, the client-side program sets a flag to indicate that sync is in progress.
5. The client-side program spawns a new thread to perform the synchronization process.
6. The synchronization thread periodically checks for changes on both the client-side and server-side.
7. If any changes are detected on the client-side (new file, modified file), the synchronization thread uploads the updated file to the server-side program.
8. If any changes are detected on the server-side (new file, modified file), the synchronization thread downloads the updated file to the client-side program.
9. The synchronization thread continues to run in the background at a specified interval, repeating the synchronization process.
10. User can continue using other functionalities of the client-side program while automatic synchronization is in progress.

Key Considerations:

- **Synchronization interval:** Define the time interval at which the synchronization thread checks for changes and performs synchronization.
- **Conflict resolution:** Decide how conflicts between client-side and server-side changes will be resolved (e.g., latest modification timestamp, user confirmation, etc.).
- **Error handling:** Handle any errors or exceptions that may occur during the synchronization process.
- **Termination:** Provide an option for the user to stop the automatic synchronization if desired.
- **Efficient synchronization:** Implement mechanisms to optimize synchronization, such as checking file modification timestamps and only syncing modified files.

The "Start Auto Sync" workflow enhances the user experience by automating the synchronization process between the client-side and server-side programs. It ensures that any changes made on either side are promptly reflected in the other, minimizing the risk of data inconsistencies. Users can focus on their work without worrying about manually syncing files, improving productivity and collaboration in a multi-user environment.

WorkFlow 8: STOP AUTO SYNC:-

Description: The "Stop Auto Sync" workflow allows users to stop the automatic synchronization between the client-side program and the server-side program. This feature gives users the flexibility to control when the synchronization process should be halted.

Steps:

1. User selects the "STOP AUTO SYNC" option from the client-side program's menu.
2. The client-side program checks if automatic synchronization is currently in progress.
3. If automatic synchronization is active, the client-side program sets a flag to indicate that sync should be stopped.
4. The client-side program joins the synchronization thread, allowing it to finish its current iteration before terminating.
5. The synchronization thread terminates, and the synchronization process is stopped.
6. The client-side program displays a message confirming that the automatic synchronization has been stopped.
7. User can continue using other functionalities of the client-side program without automatic synchronization running in the background.

Key Considerations:

- **Thread termination:** Ensure that the synchronization thread is properly terminated to avoid any resource leaks or incomplete synchronization processes.
- **Confirmation message:** Display a message to the user confirming that the automatic synchronization has been successfully stopped.
- **Handling concurrent actions:** Consider how to handle user actions that involve file operations during the process of stopping auto sync (e.g., file upload, download, deletion).

- **Graceful termination:** Allow the synchronization thread to complete its current iteration before terminating to ensure data integrity and avoid potential synchronization conflicts.

The "Stop Auto Sync" workflow provides users with the ability to pause the automatic synchronization process when necessary. It gives users control over when and how the synchronization occurs, allowing them to focus on specific tasks without continuous synchronization interruptions. By stopping auto sync, users can tailor the synchronization process to their specific needs and preferences.

WorkFlow 9: EXIT:-

Description: The "Exit" workflow allows users to gracefully terminate the client-side program and exit the application. This workflow ensures that all processes are properly closed, any ongoing synchronization is stopped, and the program resources are released.

Steps:

User selects the "EXIT" option from the client-side program's menu.
The client-side program initiates the exit process.
If automatic synchronization is active, the client-side program stops the automatic synchronization by following the steps outlined in Workflow 8: Stop Auto Sync.
The client-side program displays a confirmation message to the user, indicating that the program is exiting.
The client-side program terminates all running threads and releases any acquired resources.
The client-side program exits gracefully, and the application is closed.

Key Considerations:

- **Stopping automatic synchronization:** Check if automatic synchronization is active and stop it before exiting to ensure proper synchronization termination.
- **Confirmation message:** Display a message to the user confirming that the program is exiting to provide a clear indication of the program's termination.
- **Thread termination:** Ensure that all running threads are properly terminated to avoid any resource leaks or incomplete processes.
- **Resource release:** Release any acquired resources, such as network connections or file handles, to maintain system integrity and avoid resource conflicts.

The "Exit" workflow ensures that the client-side program is terminated properly, allowing users to close the application without any issues. By stopping any ongoing synchronization and releasing acquired resources, this workflow guarantees a clean exit and prevents any potential resource leaks or conflicts.

3. What We Have Learnt

Throughout the implementation of the client-side and server-side programs, We have gained valuable insights and learned several key lessons. Here are some of the main things We have learned:

Distributed Systems: Implementing a client-server architecture using Pyro4 library has provided me with a practical understanding of distributed systems. We have learned how to design and develop applications that utilize remote procedure calls (RPC) to communicate between client and server.

Pyro4 Library: Working with the Pyro4 library has given me hands-on experience with building distributed systems in Python. We have learned how to create Pyro objects, register them with a name server, and use proxies to invoke remote methods.

File Transfer: Implementing file upload and download functionality has enhanced our knowledge of handling file operations in Python. We have learned how to read and write files, encode and decode file data using base64, and transfer files between client and server.

Error Handling: Dealing with exceptions and error handling has been an important aspect of this project. We have learned how to catch and handle different types of exceptions to provide meaningful error messages and maintain the stability of the application.

Synchronization: Implementing the synchronization feature has provided me with insights into managing concurrent processes. We have learned how to schedule and perform periodic tasks, handle file changes, and synchronize data between the client and server.

Command-Line Interface (CLI): Building a command-line interface for the client-side program has improved our understanding of user interaction and input validation. We have learned how to present options to the user, handle user inputs, and provide appropriate feedback.

Troubleshooting and Debugging: During the development process, We encountered various issues and errors. This experience has taught me the importance of effective troubleshooting and debugging techniques. We have learned to use tools like traceback and print statements to identify and resolve issues in the code.

Overall, this project has deepened our understanding of distributed systems, file operations, error handling, synchronization, and command-line interfaces. It has provided us with practical experience in developing client-server applications and has enhanced our problem-solving skills in the context of distributed computing.

4. What Issues We Have Encountered

During the implementation of the client-side and server-side programs, We encountered several challenges and issues that required troubleshooting and resolution. Here are some of the main issues We encountered:

Pyro4 Configuration: Setting up the Pyro4 library and configuring the Pyro nameserver initially posed some challenges. We had to ensure that the Pyro4 library was properly installed and the nameserver was running correctly to establish communication between the client and server.

File Encoding and Decoding: Working with file data required careful handling of encoding and decoding operations using base64. Ensuring the correct conversion between binary data and text representation was crucial to successfully transfer files between the client and server.

Error Handling and Exception Management: Managing exceptions and error handling was an important aspect of the implementation. We had to anticipate and handle various types of errors, such as file not found, server not available, or invalid inputs from the user. Effectively capturing and conveying error messages to the user was crucial for a smooth user experience.

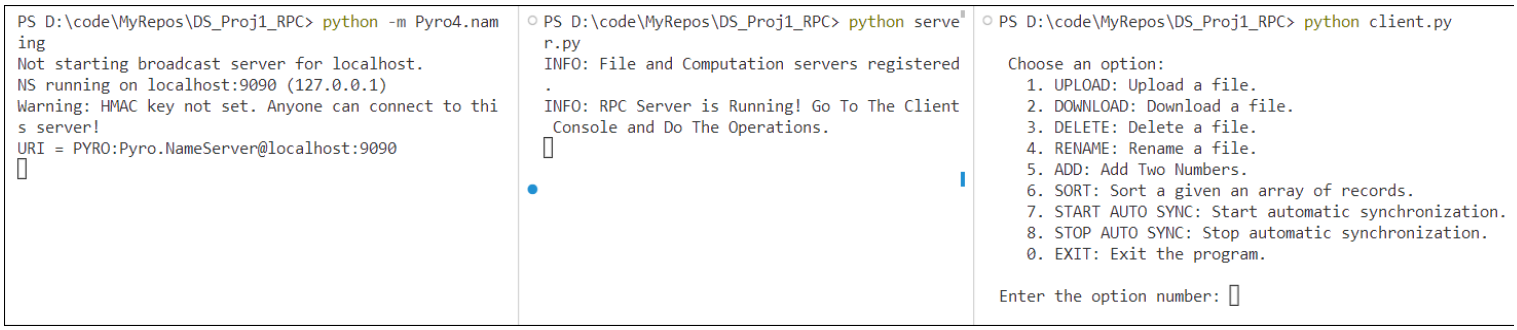
Synchronization Logic: Implementing the synchronization feature required careful consideration of file changes and synchronization intervals. We encountered challenges in detecting file modifications, managing synchronization threads, and ensuring that the synchronization process functioned correctly without any conflicts or data loss.

User Input Validation: Implementing a command-line interface required robust input validation to handle user inputs accurately. We faced challenges in validating user input for options, file paths, numbers, and arrays to prevent unexpected behavior or errors in the program.

Debugging and Testing: Throughout the development process, We encountered various bugs and inconsistencies. Debugging and testing the code required careful examination of the program's behavior, using print statements and debugging tools to identify and fix issues.

Overall, these challenges provided valuable learning opportunities in troubleshooting, debugging, and handling various aspects of distributed systems. By addressing these issues, We gained a deeper understanding of the intricacies involved in developing client-server applications and improved our problem-solving skills in the context of distributed computing.

5. Installation, Setup, and Execution Instructions



To install, set up, and execute the client-side and server-side programs, please follow the instructions below:

Prerequisites:

- Python 3.x installed on your system.
- Pyro4 library installed. You can install it using the following command:

```
pip install Pyro4
```

- colorama library installed. You can install it using the following command:

```
pip install colorama
```

In One Terminal Run The below Command before running server and client code(Make Sure Pyro4 is installed properly):-

```
Python -m Pyro4.naming
```

Client-Side:

Download the `client.py` file to your local machine.
Open a terminal or command prompt and navigate to the directory where the `client.py` file is located.
Execute the following command to start the client program:

```
python client.py
```

The client program will start running, and you will see a command-line interface (CLI) with a list of options.

Server-Side:

Download the `server.py` file to your local machine.
Open a terminal or command prompt and navigate to the directory where the `server.py` file is located.
Execute the following command to start the server program:

```
python server.py
```

The server program will start running and display the Pyro4 daemon URI, indicating that the server is ready to accept client connections.

Using the Client-Side Program:

Once the client program is running, you can interact with the server using the provided command-line interface (CLI). Here are the available options:

- UPLOAD:
  - Enter the option number `1`.
  - Provide the file path of the file you want to upload when prompted.
  - The file will be uploaded to the server.
- DOWNLOAD:
  - Enter the option number `2`.
  - Enter the name of the file you want to download when prompted.
  - The file will be downloaded from the server and saved in the local `client_downloads` directory.

- DELETE:
    - Enter the option number `3`.
    - Enter the name of the file you want to delete when prompted.
    - The file will be deleted from the server.
- RENAME:
    - Enter the option number `4`.
    - Enter the current name of the file you want to rename when prompted.
    - Enter the new name you want to assign to the file when prompted.
    - The file will be renamed on the server.
- ADD:
    - Enter the option number `5`.
    - Enter the two numbers you want to add when prompted.
    - The sum of the two numbers will be calculated on the server and displayed.
- SORT:
    - Enter the option number `6`.
    - Enter the array of numbers (comma-separated) that you want to sort when prompted.
    - The array will be sorted on the server, and the sorted result will be displayed.
- SYNC NOW:
    - Enter the option number `7`.
    - The client will perform an immediate synchronization with the server.
- START AUTO SYNC:
    - Enter the option number `8`.
    - The client will start automatic synchronization with the server at regular intervals.
- STOP AUTO SYNC:
    - Enter the option number `9`.
    - The client will stop automatic synchronization with the server.
- EXIT:
    - Enter the option number `0`.
    - The program will exit.

Follow the prompts and input the required information to perform the desired operations. The program will display relevant messages and outputs based on the chosen options.

Please note that the provided instructions assume that the client and server are running on the same machine. If the client and server are running on different machines, you will need to modify the code to specify the correct IP address or hostname of the server in the client program.

6. Pros or Cons of Our Approach

Based on the provided code and implementation, here are the potential pros and cons of our approach:

Pros:

Modularity: The code is structured into separate client and server programs, which promotes modularity and separation of concerns. This makes it easier to understand and maintain the codebase.
Use of Pyro4: Pyro4 is a powerful library for building distributed systems in Python. By leveraging Pyro4, we benefit from its features such as object serialization, remote method invocation, and name server integration, which simplify the development of distributed applications.
Synchronization: The implementation includes a synchronization feature that automatically syncs files between the client and server at regular intervals. This ensures data consistency and reduces the risk of data loss or discrepancies.
Error Handling: The code includes error handling and exception management, which helps in capturing and handling potential errors gracefully. This improves the overall robustness and reliability of the application.
Command-Line Interface: The client program provides a command-line interface (CLI) for interacting with the server. This allows users to perform various operations by providing input through the CLI, offering a convenient and user-friendly way to interact with the system.

Cons:

Lack of User Interface: The application relies solely on a command-line interface, which may not be the most intuitive or visually appealing for all users. Consider adding a graphical user interface (GUI) to enhance the user experience and make the application more accessible.
Limited Error Reporting: While the code includes error handling, the current implementation provides minimal error reporting to the user. Enhancing the error messages and providing more specific feedback can help users understand and resolve issues more effectively.
Lack of Authentication and Security: The provided code does not include any authentication or security measures. It is important to consider implementing secure authentication mechanisms to ensure authorized access and protect data integrity.
Scalability and Performance: The scalability and performance of the application may become a concern as the number of concurrent clients or the size of files increases. Consider optimizing the code and exploring distributed computing techniques to handle larger workloads efficiently.
Documentation and Comments: While the code is well-structured, it lacks detailed documentation and comments to explain the purpose and functionality of each component. Adding comprehensive documentation and comments can improve code maintainability and facilitate collaboration with other developers.

7. Who Worked on What In This Project ?

Avinash Mahala(1002079433) worked on the below components(for both clientSide & ServerSide):-

1. UPLOAD
2. DOWNLOAD
3. START AUTO SYNC
4. STOP AUTO SYNC

Amit Munna Gupta(1002066302) worked on the below components(for both clientSide & ServerSide):-

1. DELETE
2. RENAME
3. ADD
4. SORT