CSE-5306-001
Distributed Systems
Project 1

Name – Ranjith Gopal Reddy
Student id – 1002028255
Name – Suresh Kavadi
Student id – 1002040703

Introduction,

This project is about the client-server architecture to handle file operations like UPLOAD, DELETE, DOWNLOAD, RENAME and 2 other operations which is ADD and SORT operations. The mentioned operations are implemented using Java RMI system. RMI provides for remote communication between programs written in the Java programming language between server and client.

Part 1 -

To implement multi-threaded file server that supports UPLOAD, DOWNLOAD, DELETE, and RENAME file operations. Here we have 2 files server.java and client.java.
As the name suggests server, this acts as a remote object and handles remote procedure calls requested by the client. On the client side we have a stub object which has the connection details to the server and stub is responsible for all the requests performed to the server. In this part, the server handles UPLOAD, DELETE, RENAME, DELETE file operations based on the client request.

To handle these operations smoothly, create 2 directories one for client side files and one for server side files,
Client directory - C:\CSE5306\Client Files
Server directory - C:\CSE5306\Server Files

Now let us look at how to perform these activities,
Steps:

1. Open CLI is source code directory
2. Compile and run server java code,
   javac server.java
   java server.java
3. Compile and run client java code,
   javac .\ClientFactory.java

javac .\Client.java

1. To upload file,
java .\Client.java upload <filename>.<ext>

2. To download file from server,
java .\Client.java download <filename>.<ext>
Optionally to see, what files are available on server for download, you can run
java java .\Client.java listfiles

3. To rename file on server,
java .\Client.java rename <old_filename>.<ext> <new_filename>.<ext>

4. To delete file on server,
java .\Client.java delete <filename>.<ext>

Learnings, Introduction to RMI and understanding of client-server architecture and implementation of this architecture.
Challenges faced,

- When setting up the server system, when tried to restart the server. Second time execution of server failed because there was no dedicated porty assigned initially.
- At first we have had dependency on rmiregistry, which we had to run on separate CLI and execute the server. We introduced createRegistry in code to fix this dependency.

Part-2 -
To make the file operations transparent between client and server, we have created a helper thread which always executes and finds if there is any new file created, modified and deleted on the client side and automatically makes these changes on the server. For this we have made use of Java **WatchService** API, which always monitors a particular directory and notifies for changes in the directory. Based on this thread we invoke respective file operation functions
Steps:

1. *Compile and run helper thread*
   javac .\Helper.java
   java .\Helper.java
2. *Now go to the client files directory and create or delete or modify a file. And you can notice that these changes are reflected on the server.*

Learnings, Introduction to WatchService APi and understanding of how a service can monitor the directory and can invoke respective functions.

Part-3 -

Implement add(I,j) and sort(array A), and demonstrate synchronous and asynchronous RPC calls. Add and sort functions are implemented as a part of server code, and we can invoke the add and subtract command from the client program. For example,

1. java .\Client.java adder 2 3
   5
2. java .\Client.java sorter 2,1,3,4
   [2, 1, 3, 4]
   [1, 2, 3, 4]

As this part speaks about synchronous and asynchronous communication, we have created 2 programs SynchronousClient.java and AsynchronousClient.java where we can run these files without any arguments to be passed, as we hard coded the inputs as (2,3) for addition and {5, -2, 23, 7, 87, -42, 509} for the array to be sorted. For both the functions there is a 10 secs delay introduced to demonstrate the working mechanism. For both the programs we have timestamps recorded, to have a clear idea about the execution.

1. SynchronousClient execution timestamps on Client side are below,

```
Jul 03, 2022 8:32:31 PM SynchronousClient main
INFO: Adder function is currently running in thread main | Timestamp is : 2022-07-03 20:32:31.8721033
5
Jul 03, 2022 8:32:42 PM SynchronousClient main
INFO: Sorter function is currently running in thread main | Timestamp is : 2022-07-03 20:32:42.0666721
[-42, -2, 5, 7, 23, 87, 509]
```

As we notice above that 10 secs after addition is executed, sorting function starts. Now let us look at Server side timestamps,

```
Jul 03, 2022 8:32:42 PM Server add
INFO: Adder function is currently running in thread RMI TCP Connection(16)-192.168.0.110 | Timestamp is : 2022-07-03 20:32:42.0647528
Jul 03, 2022 8:32:52 PM Server Sorter
INFO: Sorter function is currently running in thread RMI TCP Connection(16)-192.168.0.110 | Timestamp is : 2022-07-03 20:32:52.0709503
```

We can notice the same here as well. One important thing to note is, both the functions happened to run on the same thread.
main thread on client side
thread 16 on server side

2. AsynchronousClient execution timestamps on Client side are below,

```
Jul 03, 2022 8:47:41 PM AsynchronousClient lambda$0
INFO: Adder function is currently running in thread ForkJoinPool.commonPool-worker-1 | Timestamp is : 2022-07-03 20:47:41.7906037
Jul 03, 2022 8:47:41 PM AsynchronousClient lambda$1
INFO: Sorter function is currently running in thread ForkJoinPool.commonPool-worker-2 | Timestamp is : 2022-07-03 20:47:41.7906037
5
[-42, -2, 5, 7, 23, 87, 509]
```

   As we notice above, even though there is 10 secs delay introduced, both the function

executions happened to be in parallel as both the calls were asynchronous in nature.
Server-side timestamps,

```
Jul 03, 2022 8:47:51 PM Server Sorter
INFO: Sorter function is currently running in thread RMI TCP Connection(22)-192.168.0.110 | Timestamp is : 2022-07-03 20:47:51.9718392
Jul 03, 2022 8:47:51 PM Server add
INFO: Adder function is currently running in thread RMI TCP Connection(21)-192.168.0.110 | Timestamp is : 2022-07-03 20:47:51.9718392
```

On the Server side also the executions are asynchronous, which happened to start at 10:47:51 precisely. Again, if we look at the threads,
Client side -
Pool-worker-1 for add function
Pool-worker-2 for sort function
Server side -
Thread 22 for add function
Thread 21 for sort function

Asynchronous mechanism is implemented using inbuilt Java **CompletableFuture** class.

Learnings, Introduction to CompletableFuture class which speaks about the asynchronous mechanism implementation in Java. Along side also had an understanding of how Java evolved to implement asynchronous mechanism more productively. Right from Threads, Runnable, Callable, Futures to more evolved version CompletableFuture.

Distribution of Work,

Ranjith Gopal Reddy,

- Implemented file operations UPLOAD, DOWNLOAD file operations using RMI.
- Implemented WatchService API to monitor file changes in directory.
- Implemented Asynchronous mechanism for add and sort operations.

Suresh Kavadi,

- Implemented file operations RENAME, DELETE file operations using RMI.
- Implemented ClientFactory to manage all the operations.
- Implemented Synchronous mechanism for add and sort operations.

References -

WatchService API - WatchService (Java Platform SE 7) (oracle.com)

RMI system - Trail: RMI (The Java™ Tutorials) (oracle.com)

Completable Future(Asynchronous mechanism) - CompletableFuture (Java Platform SE 8) (oracle.com)