I have neither given or received unauthorized assistance on this work
Signed: Gurvir, Abhishek Date: October 5, 2019

Name: Gurvir Singh Tarlok Singh Bhogal
Student ID: 1001769871

Name: Abhishek Shinde
Student ID: 1001754842

Distributed Systems
Project 1

---

1) Assignment 1:

Program Implementation:

We have used Java Programming language for the implementation of this program. We used the Socket function in java for communication between the client and the server. This program uses an inputstream and an outputstream for communication, the client provides various commands such as UPLOAD, DOWNLOAD, DELETE, and RENAME in the form of text to the server and the server performs various actions based on the command. For example: The client sends the UPLOAD command which is then received by the Server and the server sends a command acknowledgment by sending the OK message, then the client sends the file name to be uploaded to the Server, if the file is uploaded successfully the server prints a Successfully Saved message.

What have we learned:

- Socket communication between the Client and Server
- Sending and Receiving messages between Client and Server
- Sending and Receiving Files byte by byte using the input and output streams
- Working with Files in Java
- Deleting and Renaming previously created files using Java

Issues Encountered:

We have encountered various issues during sending and receiving of files by breaking them into bytes which is solved using a while loop to break the contents of the file in byte size chunks. We also faced challenges to check the availability of the file in the client side and sending appropriate error messages if any exception occurs.

Challenges:

The challenges faced were how to optimize the file sending and receiving through a socket communication and how to make the program easy to use by the client. We overcame these challenges by using minimum usage of the communication channel such as using minimal messages for acknowledgment saved memory space and the communication overhead. We also used easy to type and understandable commands for the communication.

2) Assignment 2:

Program Implementation:

We have used Java Programming language for the implementation of this program. We also used the Socket function in java for communication between the client and the server. This program uses an inputstream and an outputstream for communication, the client provides various commands such as UPLOAD, DOWNLOAD, DELETE, and RENAME in the form of text to the server and the server performs various actions based on the command. In this program we have implemented a ClientHandler class which extends the Thread class to implement the Multi-threading environment in the server side. The client programs remains the same as the Assignment 1. Using the thread class we can have multiple clients communicating with a single Server using the same port.

What have we learned:

- Creating Threads in Java
- Efficient Communication between Multiple Clients and a Single Server
- Exception handling for multiple clients

Issues Encountered:

In this assignment we had issues in creating multiple threads in the server side while keeping the client side the same. For creating multiple threads we used a ClientHandler class which keeps track of which client is connected and the process Id's of the connected client.

Challenges:

The challenges faced were how to serve each client efficiently while also accepting command from other clients and how to communicate and store commands of each client in the server side for processing. There were also cases of redundancies where multiple clients were trying to upload a single file or multiple clients were trying to download a single file.

Distributed Locking:

In a multi-thread programs multiple clients may try to access a single resource, in this case a single file to read or write into, this may cause errors for other clients if an incomplete read or an incomplete write is performed so it is necessary to implement locks on resources. A lock is simply an access restriction on a resource while an operation is performed which makes the resource unavailable during the course of critical write or reads. This ensures that only single client is using the resource at any given time.

3) Assignment 3:

Program Implementation:

This assignment was implemented in java. We created a client and a server file, and along with these, we made a stub file. This client-server architecture is intended to follow the Synchronous RPC Architecture. The stub file is used to convert the client parameters into a packed string message. The packed message is then sent to the server by serializing the message into a byte array. This byte array consists of our client parameters but it is encoded in Standard Charsets-ASCII values. The server then deserializes this message and retrieves the parameters for further computing and then returns the solution for the requested operation to the client using the Input/Output Streams.

What have we learned:

- Client and Server Socket Communication
- Data Serialization and De-serialization
- Synchronous RPC Working
- Packing and Unpacking Data using Stubs
- Client-Server Communication Medium, i.e. Input/Output Streams.

Issues Encountered:

In the course of this assignment, one challenge we faced was the transfer of parameters from the Client to the Server by packing the parameters into a message using a stub. After multiple sessions of brainstorming and following up on studies using the Java Documentation as reference, we thought of using serialization to transfer the said parameters from client to server.

4) Assignment 4:

Program Implementation:

This assignment was implemented in java. Our objective for this assignment was to reimplement the Client-Server computation server using Asynchronous and Deferred Synchronous RPCs. In Asynchronous RPC, we implemented it by having the client first send a request for computation for which the server immediately responds with an acknowledgement. On receiving this, the client resumes some other computation and it periodically keeps requesting the server to see if the computation has been completed. When the server finishes the computation, it updates the data stream with the result data. The client then receives the result data in its next periodic check.
For Deferred Synchronous RPC, we performed the implementation similar to Asynchronous RPC with respect to the Client Request and Server Acknowledgement. After receiving the server acknowledgement, the client continues with other execution. In Deferred Synchronous RPC, it is the server that interrupts the client execution to send the computation result when it is ready after computation.

What have we learned:

- Asynchronous RPC Working
- Deferred Synchronous Working

- Difference between Asynchronous, Deferred Synchronous and Synchronous RPC
- Client Interruption
- Client Periodic Request Mechanism

Issues Encountered:

The issues encountered in this assignment was returning the client back to execution after the acknowledgment message is received from the Server side and interrupting the clients executing to print the answer of the RPC called. We overcame this issue by using an availability check in the output stream, a loop continues the process of adding numbers until a return message containing the answer of the RPC is received by the Client.