5306 - Distributed Systems

Project -1 Report

Submitted by

Harshilkumar Patel -**1001717222**

Sreenidhi Timmaiahgari - **1001721679**

Assignment 1:

The problem given was to implement a single-threaded file server that supports the four operations i.e. **upload, download, rename and delete**. We used a message-oriented communication protocol i.e. UDP to establish a connection between a client and a server residing on a same machine and communicated using a different port.

Implementation of the Program:

Following is the step-by-step process of how we solved the problems:

1. We made a simple server that created a socket on localhost port 8080 and listened to any incoming connections.

2. We made a simple client which connected to the localhost server on port 8080.

3.The client sends a request through the port once the server receives the request, it fulfills the request using the same port.

4.We have created the following file-based operations:

a. Upload: The client reads the file locally and send it to the server through the socket connection. The server then implements the changes at its end

b. Download: After receiving download command, the server reads the file locally and sends the date to the client through the socket connection. The client then implements the changes at its end

c. Delete: Upon receiving the delete command, the server removes the file from its end

d. Rename: When the user enters the rename command at client-end, the new name is then sent to the server through the socket, which the server then implements at its end

5. After the client connects to the server, the server creates a handler thread that handles the command sent by the client, one all the packets are received the server again goes to listening mode.

Issues Encountered:

1. Unknown file size: Since the protocol used is independent of the connection, the server cannot know until which packet is the file streamed. To resolve this, we broke down the file using the buffer size and calculated the number of packets required to complete the transfer and inform the server beforehand. The server is then aware of the number of packets to output to the file. Similarly, for server-to-client transfer.
2. Missing files: If the file is missing for either DELETE or RENAME operation, the error for the same is conveyed to the client.

Assignment 3:

The problem given was to implement a computational server using synchronous RPC, we used TCP protocol to establish a connection between a client and a server and enable callable methods on the server side.

Implementation of the Program:

1. The assignment contains individual client and server objects to initialize the operations and a base class 'stub' to handle the encoding and decoding of the messages.

2. On the client side, once the connection is initiated, an RPC interface lets the client call any exposed methods defined on the server. So, this is a true implementation on the RPC model.

3. The server-side implementation contains a base class to define all the exposed methods i.e. add, sort, calculate pi, matrix multiplication and besides that also enable feature to encode and decode messages.

3. The server exposes the following methods:

a. *add* (i , j): returns the addition of the two inputs.

b. *matrix_multiply*(Matrix A, Matrix B, Matrix C): returns a matrix multiplication of the input. The input could be of form file -> "input.txt" or console input.

c. *calculate_pi* (): returns the pi value.

d. *sort* ([] arrayA): returns a sorted list of the input list.

Issues Encountered:

1. Exposing methods: the methods on the server had to be callable from the client. To allow this, the client stub had to accept anonymous method calls from the client object and convey this to the server. To tackle this, we used the language feature "getattr" from python, that allowed this core RPC feature.

2. Encoding and decoding messages: the method name and parameters could not be sent as blank string since; the parameter type was unknown, and the solution had to be seamless for any kind of parameters. To tackle this, we enabled packing messages in a key-value pair format, so that the type of parameter is received as expected on the server side.

3. Replying error messages: just like receiving messages in key-value pair format, the error messages were sent back just like the result.

Assignment 4:

The task was to run the server and client program in Assignment 3 in separate containers and pass messages across the host.

Implementation of the Program:

The server container runs on 8080 port and the client on 8081 port. We exposed these ports to the container and let them run on the host network. So, they expose the ports on the host network just like how the scripts would if they were running outside the docker. This allows a seamless pass of the messages between the two containers and hence the program.

The project includes bash scripts to automate the task and can be run on the terminal once you navigate to the project directory. The instructions for the same has been mentioned in the tagged README.md in the project repository.

Issues Encountered:

1.Exposing ports: Even after exposing the right ports to the container, the containers could not discover each other until the ports were exposed to the host network.

2.Allowing console input into the client container: the client container requires a direct console input, for which we used the docker command parameter “-it”.

Key Learnings

In this project, we learnt about client server model and its complexities, the connection techniques available between a client and a server. We also learnt on a better way to enable remote method call using the basic message passing which is the core of RPC model. Besides this, we also learnt the models for docker and its benefits.