Simple Job Scheduler for all Jobs to Largest Server

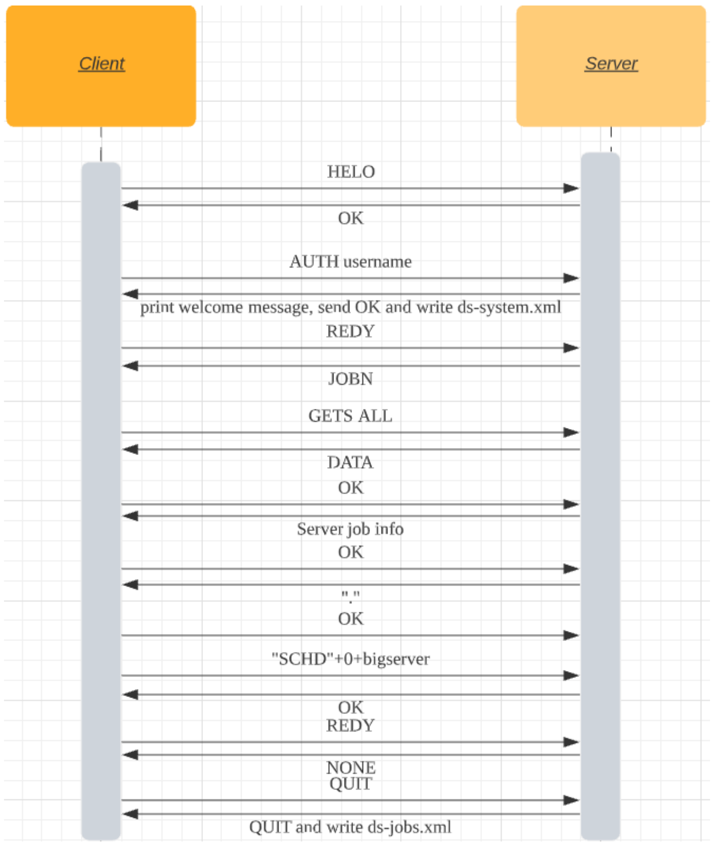
Akul Pandi: 44927835	Wenming Zhao: 45803080	Agastya Kapur:45348332

Introduction

Scheduling is an essential process that lets resource sharing within a number of activities by determining the execution order with the available resources. [2]. The main focus of stage 1 is for the group to design and implement a plain/vanilla version of client-side simulator that acts as a simple job dispatcher including basic scheduling functionalities. By the end of the project the goal is to implement a client which is able to gather and understand the information from the host of servers and allocate the job to a server within the available server. To reach the end goal, the group is required to understand and formulate an approach to execute the tasks keeping in mind the consideration and constraints of the systems.

System Overview

The system completes the work through the interaction between server and client. First, the server and client will handshake to confirm the connection, and then the system will create a DS- system.xml File to record system information. Next, after the client queries the server status information, the server will send the job information to the client. After scheduling job and confirming there are no other jobs, the connection between the client and the server will be interrupted. The specific operation is shown in the figure below.



Design

Philosophy

The philosophy of the group was to research and understand the job scheduling system. With that understanding, we were able to design a philosophy by breaking the project into parts. Agastya Kapur has taken the lead in developing the code by breaking the tasks into smaller tasks and tackling them one by one. With the understanding of the research done, Agastya was able to develop and execute the program through our philosophy of research and divide and conquer.

Considerations

One of the main considerations with developing the program is the knowledge and time that is taken to develop and test the program. When developing the program, it is important to understand the functionalities of the client and server and the interaction that is happening. We were required to consider the time it would take to develop the program and then time it would take to test and fix any problems that may occur.

Constraints

Some of the constraints that were faced while developing the solution for the program is trying to have a working job schedule and the resolution of that problem. To reach the goal of the project, Agastya was required to break the tasks into subtasks to tackle one by one. In doing so a lot of errors have arisen such as code not working correctly for the rest of the group. This is later fixed with the use of debugging and through trial and error. The use of trial and error and debugging is a time-consuming process but with the use of the system overview diagram we were able to achieve the goal.

Client-Side Simulator

The Client-side simulator refers to the action that takes place from the user or client computer. [3]. In this stage the client-side server consists of the functionalities of sending a message, receiving a message and parsing XML.

Server-side Simulator

A server-side simulation is a program that offers requested services from one or more clients. This was done through the implementation of socket programming and is used to communicate with each other. [3]. To have this functionality working we used Java APIs which ensured the connection and transfer information. The server-side simulator waits for the client side to make a connection request and when the connection is accepted, the server will receive a new socket that is then assigned to a different port.

Implementation

Overview of implementation

The implementation of the client software was done in java. The Ubuntu virtual machine was used which runs on the linux operating system. The manner in which the solution was achieved was by breaking down the whole task into small semi-isolated chunks which each fulfilled a certain purpose in the program. Then, a solution was determined for each chunk and then they would be combined into a coherent program. Another integral technique utilised in ensuring the correct solution would be achieved was using debugging statements by simply printing outputs to trace the progress and functionality of the program.

The software libraries used are java.net.*, java.io.*, java.nio.charset.StandardCharsets and java.util.ArrayList. Java.net.* consists of the socket class which creates a stream socket and allows a connection between specified port numbers and specified IP addresses. Java.io.* assists with using input and output buffer streams in addition to error handling and assisting with thread synchronization and sleeping if required. Java.nio.charset.StandardCharsets is used to import the UTF_8 eight bit transformation format to convert input and output messages into a format readable to extract information to either continue the progression of the program or to debug or manipulate data. Java.util.ArrayList is used to import the arraylist class which is used to store the server information which gets sent to convert it to an appropriate format to read and extract information like the serverId, serverName and the number of cores it has or other such server specific data.

The data structures used were classes, byte arrays and arraylists. The classes and arraylists were used to create server classes from the data sent by the server and assign them to an arraylist where they could be analysed. The byte array was used in order to create a means to input a stream of bytes to the server and receive data from the server.

Components

The primary components of the program were the handshake, reading server information and job scheduling and handling. The coding of all these sections was done by Agastya Kapur.

The handshake consisted of two parts, sending and receiving messages over a socket following a protocol defined in the ds-sim_user-guide document. The data sent over the sockets and the input/output streams are treated as buffered streams to make reading them more efficient for memory usage.

Reading the server information was performed by using the “GETS All” message sent to the server to which the response was information of all the servers available which the server reads from ds-system.xml. The format and order of the server information is known by the examples and information in the user guide, and thus, an ArrayList of servers was constructed, where each attribute, separated by a space (“ ") was assigned to a variable. First, the biggest server was determined by iterating through the server and its size was used to then find the first server in the serverlist with the corresponding size which would be the largest server which would be used in the “AllToLargest” implementation.

The job scheduling and handling was done by a simple algorithm, where the response type from the server was determined by reading the first four characters of the response (e.g. JCPL, JOBN etc.). If the response was of type JCPL, then the client would respond with REDY, if the response would be JOBN, then the a job would be scheduled with SCHD followed by the jobID number which would then be iterated by 1 to be prepared for the next job. If a message is received that could be ERR or otherwise, a REDY message is sent to request a job. The final condition is that if the message received is NONE, which would mean that there are no more jobs to be performed, the loop is broken and the QUIT message is sent which terminates the program.

Process of design and development

The process involved solving small programming chunks. The progression of functionality is as follows. First, the transmission of messages between server and client over sockets was understood and tested [1] and then after transmission and receiving of messages was done with buffered streams, the handshake protocol was fulfilled by sending and receiving messages by conforming to the protocol given by the user-guide.

Following the handshake, the choice was between parsing the ds-system.xml file for the server information or reading the server contents using GETS All. Using GETS All the get the server contents was used. The servers were put into an ArrayList which would then be used to iterate through and find the largest server.

Initially, a simple condition was used to find the largest server by which the server with more cores than the previous largest server would be assigned the largest server, however the condition did not stop a server with equal number of cores to be selected, thus a server with equal number of cores, but appearing further down in the serverlist would be selected as the largest server, which was not intended. So, to select the first appearing server with the most cores, another iteration was made through the server list where the first server with the largest core number would be returned and this server would be assigned all the jobs.

The final step was handling the job scheduling algorithm, based on the user-guide conditions of how to respond to certain server messages, it was trivial to make a loop that runs while the server to send messages that begin with JCPL or JOBN or a message that indicated no more jobs to schedule NONE which would break the loop.

A technique used consistently throughout the development was debugging using print statements to ensure that the correct output was being received from the server and being processed the correct way.

References

[1] Sonoo Jaiswal, *javatpoint.com* https://www.javatpoint.com/socket-programming [Date Accessed 17th April 2021]

[2] Bittencourt, L., Goldman, A., Madeira, E., da Fonseca, N. and Sakellariou, R., 2018. Scheduling in distributed systems: A cloud computing perspective. Computer Science Review, 30, pp.31-54.

[3] cloudflare.com. 2020. What Do Client-Side And Server-Side Mean? | Client Side Vs. Server Side.[online] Available at: <https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/>[Accessed 7 April 2020].

[4] Choon Lee, Y., Ki Kim, Y., King, J., 2021. ds-sim: A Distributed Systems Simulator User Guide.

Git repository

https://github.com/AgastyaK/Comp3100Group33