Project report: Stage 1
COMP3100 Distributed Systems, S2, 2022
SID: 46170596, Name: Alfred Consiglio

1 Introduction

This project's goal is to build a distributed system simulator that supports job assignment and task optimization to improve processing speed of the distributed system. By the time of completion, this simulator will have multiple algorithms to choose from to optimize the job scheduling and assignments. These algorithms will optimize the task que with different goals in mind, such a fastest processing speed or lower costs.

Due to the complexity of the project, the project is divided in two stage. Stage 1 is a ”vanilla” or minimal version of the system that will be created. The goal is to enable communication between the server and client in the emulator and to also find the best server with the simplest, lowest cost algorithm to distribute tasks.

2 Overview

The simulation project is split between two main components. the client side simulator MyClient.class and the server simulator ds-server. Other than the client and server, there is also a XML configuration file that holds the jobs to be completed and server information in the simulated distribution system. The client will schedule the jobs and make decisions about which servers receive what jobs.

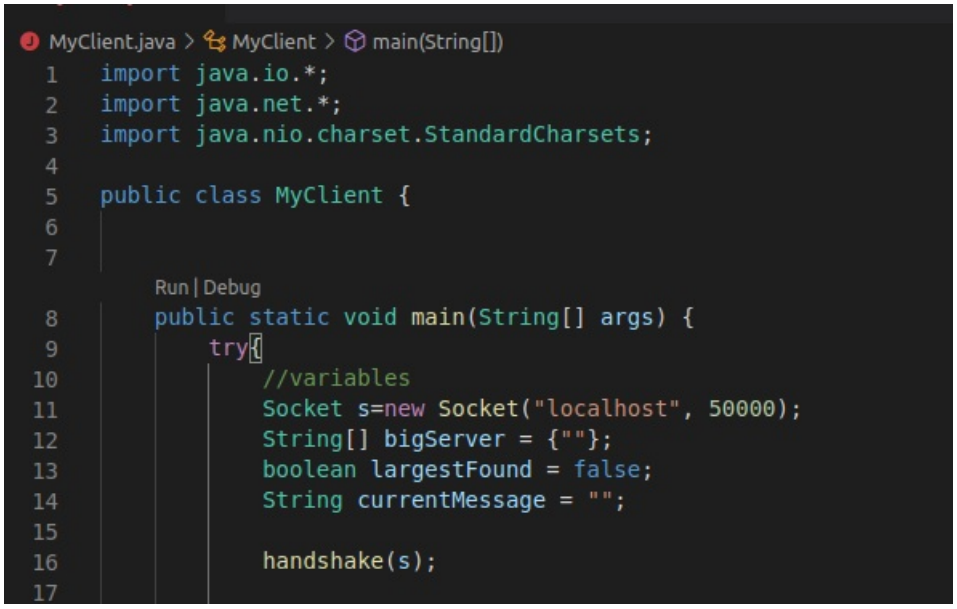
For stage one I have only implemented a ”Largest Server” algorithm. The client only compares servers once as it gets it from the .xml file. The largest server is determined by the highest cores.

3 Design

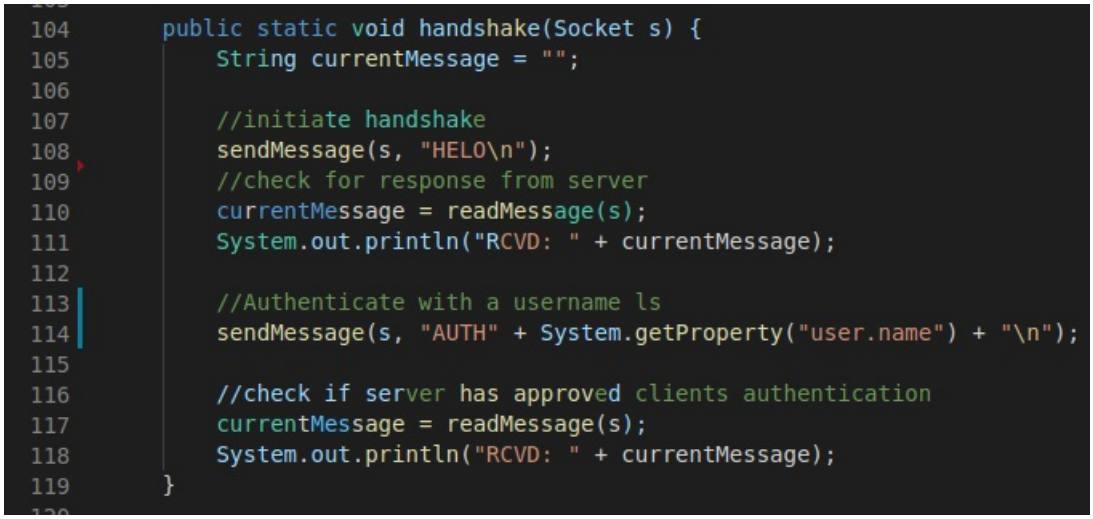
Regard the client as a single class or file instead of making multiple files. The client is split into having to achieve a few main goals. A handshake where the client sends a hello to the server and identifies its self with a username and authenticates. To achieve this a Readmessage and Sendmessage methods need to be created for the client to communicate with the server. After this hand shake the server sends data from the .xml where the client reads the data and identifies the largest server by the cores field. This server is then given the jobs to complete.

4 Implementation

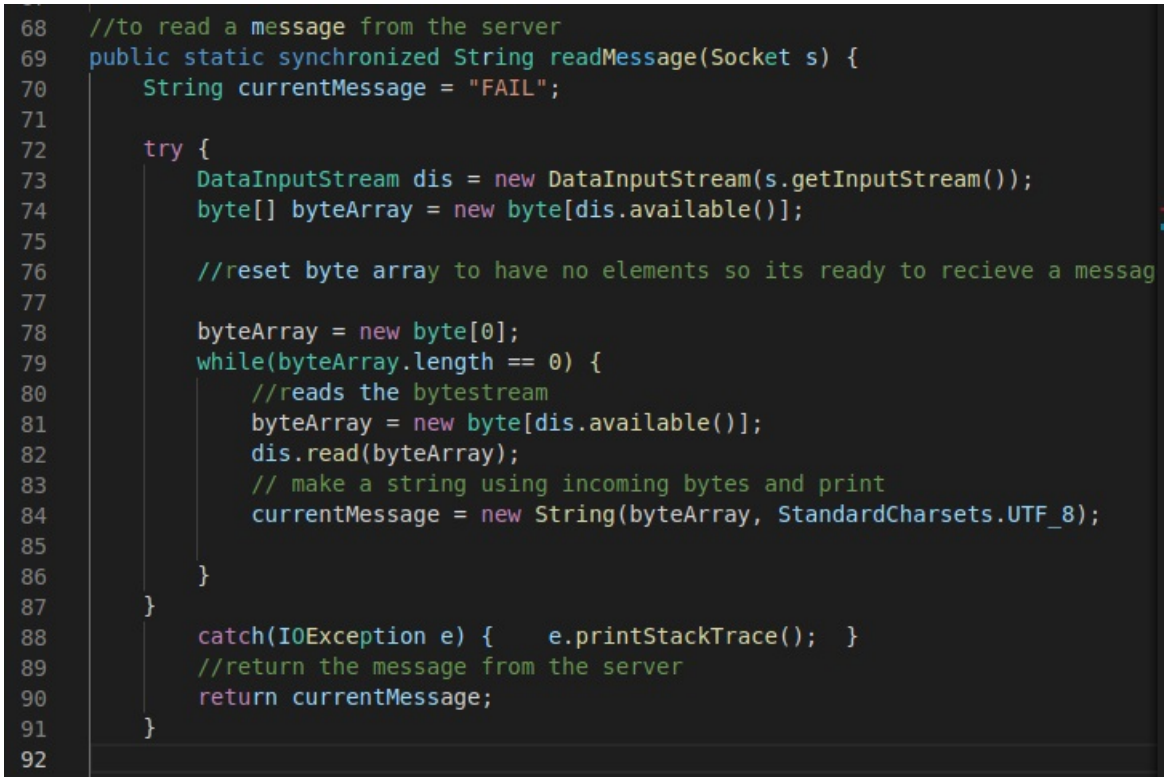
lines 1-16 Imports, variables declared and starting handshake with the socket.



lines 104-119 Handshake relies on sendMessage and readMessage methods to be able to complete the handshake. client sends HELO the server responds, the client sends AUTH + user. the server then either approves are responds with an error.



lines 68-91 readMessage the input is split into a byte array and each individual byte is a char, the chars are converted into a string to be returned from the method as a string



```java
//to read a message from the server
public static synchronized String readMessage(Socket s) {
    String currentMessage = "FAIL";

    try {
        DataInputStream dis = new DataInputStream(s.getInputStream());
        byte[] byteArray = new byte[dis.available()];

        //reset byte array to have no elements so its ready to recieve a messag

        byteArray = new byte[0];
        while(byteArray.length == 0) {
            //reads the bytestream
            byteArray = new byte[dis.available()];
            dis.read(byteArray);
            // make a string using incoming bytes and print
            currentMessage = new String(byteArray, StandardCharsets.UTF_8);

        }
    }
        catch(IOException e) {    e.printStackTrace();  }
        //return the message from the server
        return currentMessage;
    }

```

sendMessage inputs socket + string(message to send) converts string to a byte array and sends the array to the server.

```
93  //send a message to the server
94  public static synchronized void sendMessage(Socket s, String currentMessage){
95      try{
96          //converts string to a byte array and sends the array to the server
97          DataOutputStream dout = new DataOutputStream(s.getOutputStream());
98          byte[] byteArray = currentMessage.getBytes();
99          dout.write(byteArray);
100         dout.flush();
101     }catch (IOException e) { e.printStackTrace(); }
102 }
103
```

continuing into main lines 19-45 After connection is establish(handshake) the client sends the message REDY to get the jobs. if the message back from the server contains JOBN instead of an error. the client asks for servers to run the job. the message is saved and the client responds OK largest server method is run to find the server with the most cores.

```
19  //While there are jobs to do
20  while (!currentMessage.contains("NONE")){
21      //client tell the server that it is ready and reads
22
23      sendMessage(s, "REDY\n");
24      currentMessage = readMessage(s);
25      if (currentMessage.contains("JOBN")) {
26          String[] JOBNSplit = currentMessage.split(" ");
27
28          //asks for servers available to run a job
29          sendMessage(s, "GETS Avail " + JOBNSplit[4] + " " + JOBNSplit[5] + " "
30          //Reads the message saying what data is being sent and responds
31          currentMessage = readMessage(s);
32          sendMessage(s, "OK\n");
33
34          //reads the servers data and responds
35          currentMessage = readMessage(s);
36          sendMessage(s,"OK\n");
37
38          //checksif the biggest server is found
39          if(largestFound == false) {
40              bigServer = findLargestServer(currentMessage);
41              largestFound = true;
42          }
43
44          //reads from the server
45          currentMessage = readMessage(s);
46
```

lines 121-148 Findlargestserver method takes the message from the server containing the list of servers as an input. each server has the cores field examined and if that server is the largest it is saved. After cycling through all the servers the largest server is saved as a string, currentServer and returned.

```java
//find the biggest server
public static String[] findLargestServer(String currentMessage){
    int mostCores = 0;
    String[] currentServer = {""};

    //All the servers in the string split into an array
    String[] servers = currentMessage.split("\n");

    //searches for server with the most cores
    for(int i = 0; i < servers.length; i++) {
        currentServer = servers[i].split(" ");
        int cores = Integer.valueOf(currentServer[4]);

        if(cores > mostCores){
            mostCores = cores;
        }

    }
    //finds and returns the server with the most cores
    for (int i = 0; i < servers.length; i++) {
        currentServer = servers[i].split(" ");
        int cores = Integer.valueOf(currentServer[4]);
        if(cores == mostCores){
            return currentServer;
        }
    }
    return currentServer;
}
```

returning to the main and finishing off lines47-65 The current job is sent to the biggest found server. after sending the job the next job is read. if the current message contains DATA the client responds to the server with OK. When there are no more jobs this exits the while loop and the client sends the server QUIT and closes the data stream to end the connection and closing the program.

```java

                //Schedule the current job to biggest server
                sendMessage(s, "SCHD " + JOBNSplit[2] + " " + bigServer[0] + " " + bigServer[


                //reads next job
                currentMessage = readMessage(s);

                System.out.println("SCHD: " + currentMessage);
            }
            else if (currentMessage.contains("DATA")){
                sendMessage(s, "OK\n");
            }
        }
        //sends quit to the server to gracefully end connection
        sendMessage(s, "QUIT\n");
        s.close();
        }catch(Exception e){System.out.println(e);
        }
}
```

5 Github url

https://github.com/Lu-nux/comp3100A1