Project title

**Basic scheduler with inbuilt job dispatcher within a client side simulator**

Group members

Jonathan Cawood (45887454)

Carlos Larino ()

Introduction (1/2 page)

The aim of this project is to develop a job scheduler for a distrusted system .With specific relation to Stage 1 requiring the design and implementation of a “vanilla’ version of a client-side simulator that includes a basic scheduling function with a simple job dispatcher. The simple job dispatcher upon successful implementation will send all jobs to the first one of largest server type. The largest server type will need to be determined in order for this process to work. The role of a job dispatcher within a distributed system is key for ensuring the efficient use of computer systems including distributed systems which will be highlighted within this project.

System overview (1/2 page)![Diagram

Description automatically generated with medium confidence]()

Design (1 page)

Implementation (2 pages)

Jonathan Cawood worked on the implementation of initial communication to the server specifically dealing with the implementation of “HELO”, “AUTH”, “REDY”, “GETS All” requests and replies. Whilst also implementing the data handling of the servers from the server side as in the creation and inputting of each server into a created array list. Additionally, the implementation of public functions to reduce redundancy within the code was implemented.

An Array List was used to store all the information of each server using the created Server class. The Server class was created to be able to import server properties into the class and be able to locate and keep track of the imputed servers, such as the ID, State, Start Time, Cores, Memory, Disk. Therefore, enabling the use of a for loop to loop through and determine which server had the highest amounts of Cores with the use of an if statement to compare the core count with the highest core count so far, allowing for the highest to be determined.

The public function Msg sender was constructed to reduce redundancy within the code, as the sending of the message from the client to the server was repeated with only the message to be sent altered. Therefore, an overall function could be constructed that was passed the message as a string to then convert the string to bytes then flush the inputted message to the server. Whilst also letting the client-side operator know that a message has been sent. This feature allows for easier debugging in the future. The addition of this function allowed for the minimisation of code and allowed for the workflow to be easily identifiable.

Likewise, the introduction of the Msg converter and Msg receiver are to reduce the amount of redundant code within the file, with the Msg receiver reading the input stream and writing it to a byte array, to then convert to a string which is then printed on the client side, likewise with the Msg converter reading and storing the input stream, converting the byte array into a string to then return the string to the call.

The three functions were designed and developed to reduce the reptation of code whilst allowing for the ability to read the code sequentially and understand the process undertaken, therefore was designed specifically to streamline code.

References