Stoica Razvan Cosmin

Group 30424

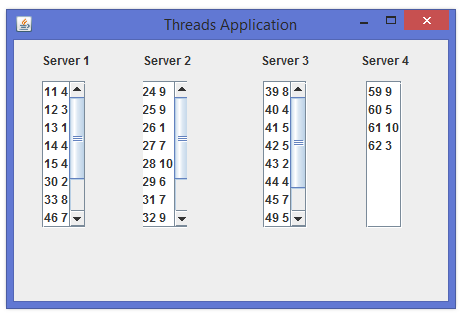
**Threads Application**

1. **Homework objective**

Design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients’ waiting time, using multithreading.

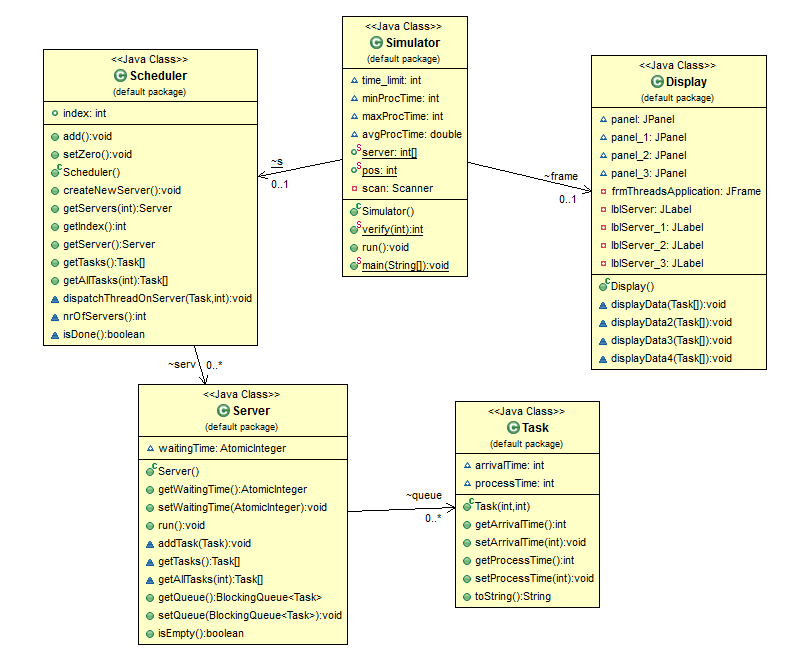
1. **Problem analysis**

The program should be simple to use and have clear input/output display. The servers should be turned on only if a threshold is passed in order to save resources. The input will be given from the console and the display of the tasks will be on the graphic interface. Were each task number as long as its waiting time will be displayed in a queue to the corresponding server it was assigned. Once a task is completed it will be removed from the queue. In order to work with threads we will need some special types of data like a blocking queue and volatile integers so we don’t have errors when computing.



1. **Design**

For the design of the project I’ve tried settle on the given templated in the class. So I have 5 classes: Scheduler, Simulator, Task, Server and Display.



UML DIAGRAM

In the class Task we have 2 integers arrivalTime and processTime as well as getters and setters for them. We also have an override at toString in order to display the data when using it in the simulator.

In the Server class, which implements the Runnable interface. We have a blocking queue of tasks, name queue and an atomic integers waitingTime (is atomic because of the operations done by the threads on it). We have getters and setters for the waitingTime and the blocking queue. We have the run method which is given by the the runnable interface. In the run interface we take an element from the queue and we put the server to sleep the amount of time which is given by the processTime. We also have an addTask method which adds a task to the server. And a getTasks method which takes all the tasks from the queue and puts them into an array and returns it.

In the Scheduler class we have an arraylist of servers of size 4. The servers will be created in the constructor of the class or in the createNewServer method. We also have an integer index which we synchronize when with an add synchronized method when we increase, and we set it to zero with a setZero synchronized mehod. This two methods will be used in the constructors when creating a new server. We have getter which returns the server at a given id. We have a getTasks method which puts the tasks in the server. We have the dispatchThreadOnServer method which adds a task on a specified server. We have a boolean method which checks if there are no servers and a nrOfServers method which returns the number of servers. This method will be used in the Simulation class when creating new servers.

In the Simulator class which implements the Runnable interface we have a time\_limit which is basically the number of tasks, we have an minProcTime and a maxProcTime which will be used as an interval of time given to the task using the random method. We also have a avgProcTime integer which return the average time it takes for a task to complete. We will have an Schedular and a display object. We also have a volatile static integer array which will remember the number of tasks on every server, this will be used inside the run method and here we will check for the server with the smallest amount of tasks and we will add a new task on it, at a position pos, which is also a volatile integer. As a threshold we use the time\_limit integer divided by 6. Here we also read the time\_limit from the standard input (the console of the application). The display on the graphical interface is done by the frame object, which will put every server inside a JPanel which is done by displayData methods.

In the Display class, we have the graphical interface of the application, here we have 4 JPanels in which we put the tasks lists of each server using the displayData methods. We use a groupLayout in order to position each server. We also have a label for every server Server 1,2,3,4.

As for the used packages we have java.util.Scanner and java.util.concurrent.atomic.AtomicInteger for the Simulator class. java.util.ArrayList for the Schedular class, which is used in order to create the list of servers. For the server class we have java.util.concurrent.BlockingQueue and java.util.concurrent.LinkedBlockingQueue for the Task queue, we also have java.util.concurrent.atomic.AtomicInteger for the atomic integer waiting time. As for the Display class we have all the usual java swing packages.

1. **Implementation and testing**

The operations work correctly if they are written in the expected format, if not the program will throw an exception (the user will see a pop-up message that will warn him/her if he did something wrong). I tested each class before using the user interface, and I tried to print every possible outcome to make sure that the application doesn’t break or throws unexpected exceptions.

1. **Results**

All the given operations work correctly and the application displays error messages in case the user inputs something wrong. I tried to have an optimal distribution on the servers before I settled on the final algorithm.

1. **Conclusion and further developments**

As a conclusion the project needs a lot of improvements in order to be optimal and bug free, and as for further developments I think that having a better graphic interface in which the user could do the things that are done from the console of the program would be a good improvement. Also displaying a peak hour for the simulation interval would make the application look more professional.