QUEUE BASED SYSTEM

Faculty of automation and computer science | TEHNICAL UNIVERSITY – CLUJ NAPOCA

GROUP 30423

Adrian - Radu Macocian

2018

# Assignment objective

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

Secondary Objectives:

* Design a runnable task class, so that each task can be executed separately
* Create a server class that such that each server will have its own thread
* Have the possibility to run all the actions at the same time
* Have 2 strategies of inserting tasks: based on the lowest queue, or based on the lowest waiting time
* Create a Scheduler class that adds tasks to the servers based on said strategy
* Create a simulation manager which creates the scheduler and which stores all the main input data
* Create the graphical user interface (Projection)
* Have a start frame which from where all the input data can be accessed
* Display all the occurring actions on the graphical user interface

# Assignment analysis, assumptions, use-cases, errors

The queue simulator should be able to create a given n number of random tasks and then add a task on a queue based on the given strategy and based on the current time of the system and the arrival time of task. The simulator must the show all those results in the GUI. All the servers (queues) should act independently and work concurrently with each other and with the tasks

The use-cases are easy to follow: once we run the application, we just have to write the input data in the text boxes assigned labeled for each of them. The program will signal if the input data is not correct, as in, the maximum time is smaller than the input time, not all the fields are completed or the strategy is not valid (the strategy can be “time” – insert the task at the queue with the smallest waiting time or “queue” – insert the task at the queue with the lowest number of queues). After inserting all the input data the user has to just press start and then the user interface will display all the running actions, with the user losing all the control over the program. Pressing the exit button will, naturally, close the application. The exceptional use cases are whenever the user tries to ignore an input field, or tries to pass invalid data to the application, in which case the program will signal the user that the input is invalid and not procced until valid input is given.

# Projection

As described above, in order to maintain an object orientated programming structure and the concurrency, I decided to create a class that describes the task, having an arrival time and a waiting period, and the run method which decrements the processing time by one each second. I also added an ID field for an easier following of the program. Then I created a server class that stores a list of tasks and the waiting time, which is the sum of all the processing times of the tasks in its list. It has the run method that that will run as long as the application is active, which will create a new thread for the head of the tasks list and decrease it’s waiting time while the task decreases its own processing time, when the processing time of a task is 0 the server will remove it from its list and then move on to the next task on the list and repeat the operation.

The graphical user interface is created with a the awt and swing packages. It uses a grid bag layout for the panel and it has a separate method for creating the main screen. Inside the window class, everything is done with JTextBoxes for the input and the ActionListner of the exit and start buttons. When the start button is pressed it will execute the run method which will reset the panel and then set it up for displaying the servers. For clarity reasons the user interface only displays the first 4 tasks of each server. It uses the setDisplay method to prepare the panel for the display and a displayData method which displays the actual state of the application. The latter method is accessed from the simulation manager every second making sure the data it’s showing is up to date.

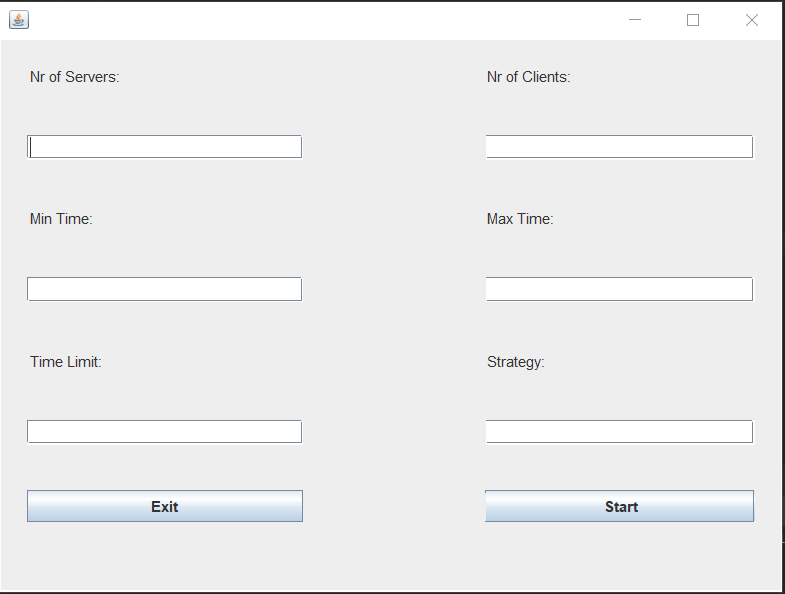
The control package contains the two classes: scheduler and simulation manager. The simulation manager creates the graphical user interface and then extracts the data from the text boxes, it then creates the tasks based on that data and passes them on to the scheduler together with the strategy when the tasks arrival time matches the current time. The scheduler holds a list of all the servers and whenever it receives a task, it will add it to one of the existing servers based on the given strategy.

O imagine care conține captură de ecran

Descrierea a fost generată cu un grad foarte mare de încredere

# Implementation

The graphical user interface uses a grid bag layout for the only panel it uses. Its resolution is set with an absolute value on 640 x 480 pixels and the position is set for the center of the screen.



Each individual text box has its own label and specifies the input data for the respective field. The start button acts on a ActionListner. The text is taken from the text boxes and converted into integers except for the strategy which receives the text directly as a string. All the boxes must be filled in order for the application to run. If all the boxes are not filled then a new label is created between the buttons which states the error and how to fix it. The exit button is not necessary since we still have the utility buttons of the window but I added it for commodity reasons.

Each action done on the window has its own method, such as the startDisplay which is shown above, the setDisplay which fills the screen with n \* 6 labels and naming the first line as being the servers, n being the number of servers. And then the display method which sets the text for the labels as follow: the second line being the waiting time of each server and then the next 4 lines are the tasks of each server. Those lines might be empty if there is no tasks on that position.

# Testing

Since the main purpose of this application is to test and implement threads, there was no way in which to test them reliably using Junit tests, therefore I have done no testing for this application.

# Conclusions

The queue based system works as a server simulator that is capable of running a number of tasks simultaneously based on the given number of servers which “process” those tasks. As long as the input fields are correctly inserted processor can take care of the rest. It’s structured in an object oriented programming style extending the tasks into a list that becomes it’s one entity: a server. Each class has its own functionality and is vital to the program. The data is passed through them continuously so each of them has to do its job correctly. The whole graphical interface is done manually, and although I considered (and tried) creating the user interface without using a layout I came to the conclusion that the software would luck much cleaner and would be more responsive if I did use a layout. Since I already learned some of the basics of the GridBagLayout I used it for this application as well. No helping software was used, so now I’m even more capable of finding any bugs and fixing them a lot easier since I know exactly why I did something or why something is in a specific place.

The only problem I found with the program is that the servers waiting time and the tasks process time are decremented independently. Because of that I had to create a special check when the server removes a task so that it whenever the list is empty, the waiting time is reset to 0, in case some inconsistencies occurred. I could not think of a solution to this problem. The tasks list is still a synchronized list so whenever the scheduler tries to add a new task and the server tries to remove one, the entire process is thread safe. I choose the synchronized list over the blockingqueue because I needed access to all the elements (at least the first 4) from the list in order to display them. Since the blockginqueue only allows the user to remove, or to check the head of the queue, I found no way of using a blockingqueue effectively without having some sort of auxiliary list from which to get the non-head tasks.

I feel like I’ve learned a lot about threading while working on this assignment. I knew vaguely how threads should work but I had no idea on how to use them and could barely design a software without even considering using synchronized objects. This assignment thought me a lot about concurrency since I didn’t understand it before and I was avoiding it until I had to really use it just now. In this assignment I’ve also learned how to update a GUI, and make sure its data is always up to date, while executing it through and outside class. Threading is still confusing for me, but I can now handle the basics and I’m pretty sure that, if handled right, threading can have a huge positive impact on a given software.

One of the other big improvements in my coding abilities following this assignment is my ability to use thread safe, or synchronized entities. I’ve never had to use them before and tried mostly using the same entities as for a one thread application. Being forced to write a program so reliant on threads and thread safety the I’ve learned how the AtomicIntegers, BlockingQueues and SyncronizedList work. And to my surprise this was not even that hard to learn once you understand the basic principles of the threads and concurrency.

Possible improvements:

One of the main problems I see with this program is the display limit to 4 tasks per server. Finding a way of showing all the tasks without sacrificing the clarity of the program would be a big improvement. This would solve some of the confusion which may be created by seeing a waiting time on a server way bigger than the processing time of it’s tasks. This is not necessarily interfering with the way the program works but it is slightly inconvenient and disturbing. The graphical interface could be made more colorful and it could be created only using one of the awt or swing libraries. It is advised not to combine them, since they could have some unforeseen interactions between each other that could mess with the interface.

One of the biggest possible improvements is changing the way the server updates its waiting time to be directly affected by the tasks processing time. Right now, the waiting time is being decremented at the same time as the task is being decremented, but both of them are being worked on simultaneously and independently, so in order to avoid some serious errors there is an error correction which resets the waiting timer to 0 any time the task list of the server is empty.

Another improvement could be made by changing the way the input error is shown on screen, so that it doesn’t disturb the whole layout for the error to show appear on the panel.

# Bibliography:

Most of the project was done by myself with the help of the laboratory professor, who gave us some ideas on how to use threads and on how to make our application as thread safe as possible. The only two external information sources were used for the creation of the graphical user interface and for creating the division of polynomials:

<https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>

<https://en.wikipedia.org/wiki/Polynomial_long_division>

For a debugging and problem solving there were a lot of searches on stack overflow but I can’t pin point exactly to which url’s I’ve used:

<https://stackoverflow.com/>