MINISTRY OF EDUCATION AND RESEARCH
TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FUNDAMENTAL PROGRAMMING TECHNIQUES

ASSIGNMENT 2

QUEUES MANAGEMENT APPLICATION USING THREADS AND SYNCHRONIZATION MECHANISMS

1. Requirements

Design and implement a queues management application which assigns clients to queues such that the waiting time is minimized.

Queues are commonly used to model real world domains. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue-based systems is interested in minimizing the time amount their "clients" are waiting in queues before they are served. One way to minimize the waiting time is to add more servers, i.e., more queues in the system (each queue is considered as having an associated processor) but this approach increases the costs of the service supplier.

The queues management application should simulate (by defining a simulation time $t_{simulation}$) a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues. All clients are generated when the simulation is started, and are characterized by three parameters: ID (a number between 1 and N), $t_{arrival}$ (simulation time when they are ready to enter the queue) and $t_{service}$ (time interval or duration needed to serve the client; i.e. waiting time when the client is in front of the queue). The application tracks the total time spent by every client in the queues and computes the average waiting time. Each client is added to the queue with the minimum waiting time when its $t_{arrival}$ time is greater than or equal to the simulation time ($t_{arrival} \geq t_{simulation}$).

The following data should be considered as **input data** for the application that should be inserted by the user in the application's user interface:

- Number of clients (N);
- Number of queues (Q);
- Simulation interval ($t_{simulation}^{MAX}$);
- Minimum and maximum arrival time ($t_{arrival}^{MIN} \leq t_{arrival} \leq t_{arrival}^{MAX}$);
- Minimum and maximum service time ($t_{service}^{MIN} \leq t_{service} \leq t_{service}^{MAX}$);

1.1 Example

Consider the following input data for the application:

- N=4 clients
- Q = 2 queues
- $t_{simulation}^{MAX} = 60$, a 60 second simulation interval
- [2, 30] - the bounds for the client parameters, respectively a minimum and maximum arrival time, meaning that clients will go to the queues from second 2 up to second 30.
- [2, 4] - the bounds for the service time, meaning that a client has a minimum time to wait in front of the queue of 2 seconds and a maximum time of 4 seconds.

Using this input data, a set of 4 clients are generated random, each client *i* being defined by the following tuple: (ID^i, $t_{arrival}^i$, $t_{service}^i$), with the following constraints:

- $1 \leq ID^i \leq N$

- $t_{arrival}^{MIN} \leq t_{arrival}^{i} \leq t_{arrival}^{MAX}$
- $t_{service}^{MIN} \leq t_{service}^{i} \leq t_{service}^{MAX}$

A number of Q threads will be launched to process in parallel the clients. Another thread will be launched to hold the simulation time $t_{simulation}$ and distribute each client i to the queue with the smallest waiting time when $t_{arrival}^{i} \geq t_{simulation}$

The log of events contains the status of the pool of waiting clients and the queues as the simulation time $t_{simulation}$ goes from 0 to $t_{simulation}^{MAX}$. An example of data displayed in the log of events is given in the table below:

Log of events	Explanation
Time 0 Waiting clients: (1,2,2); (2,3,3); (3,4,3); (4,10,2) Queue 1: closed Queue 2: closed	At time $t_{simulation} = 0$, a number of 4 clients are generated. Client with ID = 1 has an arrival time equal to 2, meaning that it will be ready to go to a queue when $t_{simulation} \geq 2$. Furthermore, it has a service time equal to 2, meaning that is needs to stay 2 timesteps in the front of the queue. The same rules apply for the next 3 clients. The two queues are closed since there are no clients available.
Time 1 Waiting clients: (1,2,2); (2,3,3); (3,4,3); (4,10,2) Queue 1: closed Queue 2: closed	At time $t_{simulation} = 1$, none of the clients can be sent to the queues because none of them has the arrival time greater or equal to 2. The two queues are closed since there are not clients available.
Time 2 Waiting clients: (2,3,3); (3,4,3); (4,10,2) Queue 1: (1,2,2); Queue 2: closed	Queue 1 opens and the client with ID =1 is sent to the first queue since $t_{arrival}^{1} \geq t_{simulation} = 2$. Other clients are still waiting. Queue 2 is closed.
Time 3 Waiting clients: (3,4,3); (4,10,2) Queue 1: (1,2,1); Queue 2: (2,3,3);	Queue 2 opens at time $t_{simulation} = 3$, client with ID = 2 is sent to it since $t_{arrival}^{2} \geq t_{simulation} = 3$, and the waiting time at the second queue (0) is smaller than the waiting time at the first queue (1), where a client is still processed. The client from queue 1 has its service time decreased to 1 (coloured in yellow) because it is being processed. Other clients are still waiting.
Time 4 Waiting clients: (4,10,2) Queue 1: (3,4,3); Queue 2: (2,3,2);	At time $t_{simulation} = 4$, client with ID = 3 is sent to the first queue since $t_{arrival}^{3} \geq t_{simulation} = 4$. Furthermore, client with ID =1 was eliminated from the queue because its service time has dropped to 0 (it was 1 at the previous iteration and was decreased with one at the simulation step) The client from queue 2 has its service time decreased to 2 (coloured in yellow) because it is being processed. The final client is still waiting.
…	
Average waiting time: 2.5	The simulation is finished when there are no more clients in the waiting queue or at the service queues or $t_{simulation} > t_{simulation}^{MAX}$

	The average waiting time is computed and appended to the log of events.

2. Deliverables

- A **documentation** written in the template provided on the laboratory Web site
- **Source files** – will be uploaded on the personal ***gitlab*** account created according to the instructions in the **Laboratory Resources** document, and following the steps:
  - Create a private repository on ***gitlab*** named according to the following template *PT2024_Group_FirstName_LastName_Assignment_2*
  - Push the source code and the documentation (**push the code not an archive with the code**)
  - Share the repository with the user **utcn_dsrl**

3. Evaluation

The assignment will be graded as follows:

Requirement	Grading
Minimum to pass • Object-oriented programming design • Random Client Generator • Multithreading: one thread per queue • Appropriate synchronized data structures to assure thread safety • Log of events displayed in a .txt file (see the example in Section 1.1) • Implement classes with maximum 300 lines (except the UI classes) and methods with maximum 30 lines • Use the Java naming conventions • Good quality documentation addressing all sections from the documentation structure.	5 p
Strategy pattern and the two strategies (shortest time, shortest queue) for allocating clients to queues	1
Graphical user interface for: (1) simulation setup, and (2) displaying the real-time queue evolution.	2 p
Display of simulation results (average waiting time, average service time, peak hour for the simulation interval) in the graphical user interface/.txt file corresponding to the log events	1 p
Run the application on the input data sets listed in the table below* and include the generated logs of events in your documentation/repository.	1 p

*For the application testing use the input data sets from the table below:

Test 1	Test 2	Test 3
N = 4 Q = 2 $t_{simulation}^{MAX}$ = 60 seconds $[t_{arrival}^{MIN}, t_{arrival}^{MAX}]$ = [2, 30] $[t_{service}^{MIN}, t_{service}^{MAX}]$= [2, 4]	N = 50 Q = 5 $t_{simulation}^{MAX}$ = 60 seconds $[t_{arrival}^{MIN}, t_{arrival}^{MAX}]$ = [2, 40] $[t_{service}^{MIN}, t_{service}^{MAX}]$= [1, 7]	N = 1000 Q = 20 $t_{simulation}^{MAX}$ = 200 seconds $[t_{arrival}^{MIN}, t_{arrival}^{MAX}]$ = [10, 100] $[t_{service}^{MIN}, t_{service}^{MAX}]$= [3, 9]

4. Bibliography

- http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html
- http://www.tutorialspoint.com/java/util/timer_schedule_period.htm
- http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html