Concurrent and Distributed Systems

Vaccine Distribution Plan

STUDENT: Ciocîrlan Marius-Cosmin

GROUP: C.EN. 3.1B

YEAR: III

Technical report

1 Application design

1.1 Architectural overview :

The application architecture is described as follows :

- **Production robot** - implements a thread that acts like a production robot, it creates vaccines. The robots don't stop until they finish the task. Every time they create a vaccine, they lock the other robots movement and move to a random location in the vicinity of their current location. If they can't move (that location is occupied) they sleep for $10 \leq a \leq 50$ (milliseconds)
  - *factoryId* - the factory that contains the robot
  - *position* - the robot's current X and Y coordinate in the factory matrix
  - *robotMapPosition* - the position of each robot mapped to the respective robot
  - *vaccineQueue* - a queue of the position we find the vaccine
  - *run* - the elf will execute the following actions in an infinite loop :
    - creates a new vaccine
    - moves in the factory
    - sleeps 30 milliseconds
  - *move* - moves a robot in the factory :
    - locks the factory lock
    - tries to move in any direction or stops working if surrounded
    - moving in a direction means changing position in matrix, creting a vaccine, modifying the robot's current position
    - unlocks the factory matrix lock
- **ProductionRobotSpawner** - implements a thread, used to spawn a robot to every factory. Whenever this thread spawns a robot, it gets the lock to that factory and it blocks every robot movements until it spwans that robot in a valid location
  - *factory* - an array with every existing factories
  - *run* - the thread will do the following actions in an infinite loop :

        * sleeps between 500 and 1000 milliseconds
        * spawns an elf

- **VaccineQueue** - a concurrent queue, shared between packing and production robots, used to transfer the vaccine from one to another
    - *pop* - method used by the packing robots to get a vaccine and send it to HQ
    - *push* - method used by the production robots to create a vaccine
- **Main** - the starting point of the application
    - creates the vaccine transfer queue
    - creates the factories
    - creates the producion robot spawner
    - creates the packing robots
- **PackingRobots** - implements a thread that acts like a packing robot. This robot gets the vaccine and send it to HQ, there can only be 10 robots that transfer at once, but only one HQ
    - *numberOfFacilities* - total number of factories
    - *vaccineQueue* - the means of vaccine transfer
    - *run* - the thread will execute the following actions in a infinite loop :
        * gets a vaccine from a factory
        * gives the vaccine to HQ via vaccineQueue
        * sleeps between 10 and 30 milliseconds
- **Factory** - implements a thread that will act like a factory
    - *id* - the factory's identification number
    - *factoryDimension* - the factory matrix size
    - *elves* - the existing elves in the factory
    - *vaccineQueue* - the factory's queue where the production robots will create vaccines
    - *factoryLock* - a lock for accessing the factory matrix
    - *getFactoryLock* - returns the factory lock
    - *addProductionRobotToFactory* - add a robot to a random valid location
    - *run* - asks all the existing robots for their current position :
        * locks the factory lock
        * all elves in the elves list report their current position
        * unlocks the factory lock
        * sleeps for 3000 milliseconds

1.2 Implementation decisions :

Methods of synchronization :

- *Common*

  For factories' correct functionality there were used 1 locks :

  - A lock for limiting the access to the factory matrix used when an elf moves in the factory (two elves can't move at the same time in the factory or there can exist position mistakes), or when asking elves for their position (elves can't move while reporting their position).

  For the packing robots synchronization there was used a semaphore with 10 permits (maximum 10 robots can read the queues to send a vaccine to HQ)

  A synchronized queue to limit the access for the the packing and production robots, a packing robot can not get a vaccine while production robots produce vaccines.

2 Observations :

- Since production robots are very fast (they only rest 30 milliseconds between creating gifts), packing robots also need to be fast in getting gifts from the factories: they will have a sleeping time between 10 and 30 milliseconds.
- Multiple robots can send vaccines to HQ, so it needs to be fast
- The vaccine queue can be accessed by only one packing robot
- The factories will also be working threads since they ask the robots to report their positions every 3 seconds.
- All factory members that were accessed/modified by multiple parties were synchronized : the queue and the production robots.

References

[1] *All the laboratories*

[2] http://www.objectaid.com/ *UML Explorer for Eclipse.*

[3] https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Semaphore.html *Semaphores in Java.*

[4] https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReentrantLock.html *Locks in Java.*

[5] LaTeXproject site, https://www.overleaf.com