Concurrent Distributed Systems
Final Homework

SANTA CLAUS WORKSHOP PLAN

STUDENT: DRAGHICI ROBERT-CRISTIAN

GROUP: CEN 3.2A

YEAR: III

Technical report

1 Application design

1.1 Architectural overview :

The application architecture is described as follows :

- **Workshop** - Santa's Workshop
  - *nrFactories* - number of existing factories
  - *factories* - all existing factories
  - *spawners* - all existing elf spawner threads
  - *nrTotalElves* - total number of existing elves
  - *elvesCounterLock* - a lock for the number of existing elves
  - *reindeers* - all existing reindeers
  - *giftQueue* - means of gift transfer between Santa and reindeers
  - *elfRetireSemaphore* - a semaphore for elf retirement
  - *elfRetire* - a thread for elf retirement
  - *getElvesCounterLock* - returns the elves counter lock
  - *createFactories* - creates all factories, elf spawners, reindeers and starts their execution
- **Elf** - implements a thread that acts like an elf
  - *number* - the elf's number/name (identification)
  - *X* - the elf's current X coordinate in the factory matrix
  - *Y* - the elf's current Y coordinate in the factory matrix
  - *gift* - the current gift created by the elf
  - *factory* - the factory that contains the elf
  - *run* - the elf will execute the following actions in an infinite loop :
    - creates a new gift
    - moves in the factory
    - sleeps 30 milliseconds
    - tries to retire from the factory
  - *changePosition* - changes the elf's current coordinates

  - *getNumber* - returns the elf's number/name
  - *getX* - returns the elf's current X coordinate
  - *getY* - returns the elf's current Y coordinate
  - *getGift* - returns the elf's current gift
  - *stopWork* - makes the elf sleep between 10 and 50 milliseconds
  - *reportPosition* - prints the elf's current position in the factory matrix

- **ElfSpawner** - implements a thread used to spawn an elf in a certain factory
  - *factory* - the factory for which the thread will spawn elves
  - *run* - the thread will do the following actions in an infinite loop :
    - sleeps between 500 and 1000 milliseconds
    - spawns an elf
  - *spawnAnElf* - the thread will execute the following actions :
    - gets the factory matrix lock and locks it
    - creates a new elf if possible (number of existing elves in factory smaller than factory size / 2)
    - gets the factory elves counter lock and locks it so that the elves' total number can't be modified
    - adds the elf to the factory and grows the number of total elves
    - unlocks the factory elves counter lock
    - unlocks the factory matrix lock

- **ElfRetirement** - implements a thread used to retire a random elf
  - *run* - the thread will execute the following actions in an infinite loop:
    - releases a permit of the retirement semaphore so that an elf can retire
    - sleeps 50 milliseconds

- **Reindeer** - implements a thread that acts like a reindeer
  - *number* - the reindeer's number/name (identification)
  - *factories* - all the existing factories in the workshop
  - *giftQueue* - the means of gift transfer
  - *run* - the thread will execute the following actions in a infinite loop :
    - gets a gift from a factory
    - gives the gift to Santa via giftQueue
    - sleeps between 10 and 30 milliseconds
  - *giveGiftToSanta* - gives a gift to Santa (puts it in the gift queue)

  - *getGiftFromFactory* - chooses a random factory from the existent ones and takes a gift from there

- **GiftTransfer** - a concurrent queue used as a means of transfer between Santa and his reindeers
  - *head* - the head of the queue
  - *tail* - the tail of the queue
  - *gifts* - the numbers of gifts to be transferred
  - *receiveGift* - method used by Santa to get a gift from the reindeers
  - *giveGift* - method used by a reindeer to give a gift to Santa

- **SantaClaus** - implements a thread that will act like Santa Claus
  - *giftQueue* - the means of gift transfer
  - *run* - Santa will receive gifts endlessly

- **Planner** - the starting point of the application
  - *main* :
    * creates the gift transfer queue
    * creates Santa
    * creates the workshop
    * the workshop starts to create factories
    * Santa starts receiving gifts from reindeers

- **ToyFactory** - implements a thread that will act like a factory
  - *number* - the factory's number
  - *N* - the factory matrix size
  - *elves* - the existing elves in the factory
  - *gifts* - the existing gifts in the factory
  - *factoryLock* - a lock for accessing the factory matrix
  - *elvesListLock* - a lock for accessing the elves list
  - *reindeerSemaphore* - a semaphore for maximum reindeers allowed in the factory (10)
  - *giftsLock* - a lock for accessing the gifts list
  - *getFactoryLock* - returns the factory matrix lock
  - *nrExistingElves* - returns the number of existing elves in the factory
  - *getN* - returns the factory matrix size
  - *getNumber* - returns the factory number
  - *run* - the thread will execute the following actions :

    - asks all existing elves for their position
    - sleeps for 3000 milliseconds
- *moveElf* - moves an elf in the factory :
    - locks the factory matrix lock
    - tries to move in any direction or stops working if surrounded
    - moving in a direction means changing position in matrix, creting a gift, modifying the elf's current position and asking all elves for their positions in the factory
    - unlocks the factory matrix lock
- *canMoveUp* - checks whether an elf can move up
- *canMoveDown* - checks whether an elf can move down
- *canMoveRight* - checks whether an elf can move right
- *canMoveLeft* - checks whether an elf can move left
- *addElf* - adds a newly created elf in the factory :
    - locks the elves list lock
    - if elf position not taken already adds the elf to the elves list, asks elf to report its current position and unlocks the elves list lock
- *askElvesForPosition* - asks all the existing elves for their current position
    - locks the factory matrix lock
    - locks the elves list lock
    - locks the gifts list lock
    - all elves in the elves list report their current position
    - unlocks the factory matrix lock
    - unlocks the elves list lock
    - unlocks the gifts list lock
- *getGift* - method used by a reindeer to get a gift from the factory
    - acquires a reindeer permit
    - locks the gifts list lock
    - gets a gift from the gift list
    - unlocks the gifts list lock
    - releases a reindeer permit
- *createGift* - adds a gift to the gift list
    - locks the gifts list lock
    - puts the gift in the gift list
    - unlocks the gifts list lock
- *retireElf* - retires an elf from the factory
    - locks the elves list lock

* lock the factory list lock
* removes the elf from the factory list and matrix
* unlocks the elves list lock
* unlock the factory list lock

1.2 Implementation decisions :

The Planner is the highest level of the architecture and it also contains the main method.
In the workshop, we compute random numbers for the number of factories (between 2 and 5), for factory dimension(between 100 and 500) and the number of reindeers(between 8 and 15). We create, start and join the factories, reindeers, elf spawners, the semaphore and method for elf retirement.

Methods of synchronization :

- *Common tasks*

  For factories' correct functionality I used 3 locks:

  - A lock for limiting the access to the elves list used when a new elf is added in the factory (2 elves can't be added at the same time) or when asking elves for their position (the elves' list can't be modified then).
  - 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).
  - A lock for limiting the access to the gifts list used when asking elves for position (a reindeer can't get a gift while the factory is asking elves to report their positions), when a reindeer gets a gift from the factory (2 reindeers can't read the same gift), and when creating a new gift in the factory (modifying the gift list).

  For reindeer factory entrance synchronization I used a semaphore with 10 permits for every factory (because maximum 10 reindeers can access the factory at the same time), acquired when

a reindeer gets a gift from the factory.

Since I gave each elf a number which identifies them globally, and not by factory, I used a lock for accessing total elves counter so that 2 elves can't have the same number.

I used a concurrent queue instead of a TCP connection through a server for transferring gifts from reindeers to Santa, because I found it to be simpler, synchronizing the methods of adding a gift in the queue and removing a gift from the queue.

- *Extra tasks:*

  Retiring an elf:

  For retiring an elf there was introduced a new thread that will release a retiring permit every 50 milliseconds. Each elf will move in the factory then try to acquire a permit to retire from the factory. Retiring from the factory means that an elf will be removed from the factory matrix and from the list of existing elves in the factory.

  Sleeping elves - semaphores:

  When reaching the main diagonal, en elf will try to acquire a semaphore to modify the counter for elves awaiting at the barrier then wait while the counter is smaller than N.
  (Note : the maximum elves in a factory was modified from N/2 to N, so that all positions on the main diagonal can be occupied)

  Sleeping elves - cyclic barrier:

  When reaching the main diagonal, en elf will await at the barrier until N elves have reached it, then continue its normal actions (move in the factory).

Sleeping elves - own cyclic barrier:

For my own cyclic barrier implementation, the await method will use a counter lock to modify the counter for elves awaiting at the barrier then wait while the counter is smaller than N.

2 Observations :

- Reindeers will have a sleeping time between 10 and 30 milliseconds, less than elves, because they need to be at least as fast in getting gifts, as elves produce them.
- Multiple reindeers can give gifts to Santa so he needs to manage them fast(Santa gets no rest between receiving gifts).
- If we don't want an elf to immediately retire after creating a gift (as it happens with the implemented code), we should give the retirement thread a longer sleeping time between releasing retire permits.
- The gift list of a factory can be accessed by only one reindeer at a time, so the reindeer semaphore acts as an access queue to get the gift list lock.
- The toy factories will also be working threads since they ask elves to report their positions every 3 seconds.
- We synchronized all factory members that can be accessed or modified: the matrix, the elves list and the gifts list.

References

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

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

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

[4] LaTeXproject site, http://latex-project.org/

[5] *Laboratories from the course*