

Programming Project: A Multithreaded Controller for the "Green Elevator"

Objectives

The main objectives of this assignment can be summarized as follows:

- To understand how to use threads (Pthreads, Java threads) in concurrent computing;
- To understand how to implement a distributed multithreaded application in a distributed environment;
- To implement a multithreaded application.

Task

Your group will design, implement, and evaluate a controller(s) for elevators simulated by the Elevators application written in Java (see links below).

The controller must accept actions events from the elevators (button pressings, position changes) and react on the events by sending control commands to elevators' motors, doors and scales (level indicators). In order to be able to control elevators in "real-time", the controller should be implemented as a multithreaded application. The velocity of elevators can be controlled "by-hand" with a special control slider in the Elevators GUI, so the timing requirements (deadlines) can be really hard.

Quality of a control algorithm (quality of service)

A control algorithm should provide fairness in servicing visitors which call

controller algorithm should provide fairness in servicing visitors which can and use elevators. In order to achieve acceptable quality of service which can be estimated as the servicing time (waiting time plus moving time), you should try to achieve the following in your controller algorithm and its implementation:

- Minimize servicing time that is a sum of two components:
 1. Waiting time that a visitor spends waiting for an elevator after he/she presses a button on a floor,
 2. Moving time that a visitor spends traveling in the elevator (including door open/close time, entering and leaving the elevator).


Is it true that minimizing waiting time and minimizing moving time are aims that are in contradiction to each other? For example, in order to decrease waiting time, you may want to stop the elevator when it passes a floor where a button has been pressed if a direction associated with the button corresponds to the direction of the moving elevator. But in this case, the moving time of visitors in the elevator increases (while the waiting time decreases!). Based on this observation, one can conclude that improving waiting time or, as an alternative, improving traveling time does not give any gain in servicing visitors. Is it true?

- Avoid starvation of elevators ("dancing" of an elevator between floors without actual service).
- Minimize power consumption of elevators (do not move elevators too much without need). Take into account service_time/power tradeoff.

See also the section "Testing an Elevators' controller" below.

Implementation

The controller should be implemented as a multithreaded program. For example, the controller may contain one "master" thread (event listener) that gets input events (button pressings, position changes and velocity changes) from the elevators, and several "worker" threads - one "worker" per one elevator. The master thread gets and parses input events, and delivers the events to corresponding workers. The master can also make a decision which one of the workers should be used to deliver the events to the next visitor.

decision which master thread (i.e. which elevator) should process a request (floor button pressing) based on current states of elevators and the request. To make a decision the master can use a "cost" function calculated for each elevator that indicates a "cost" of processing request by a particular elevator. The functions can be calculated by a master thread or by elevator threads on master's request or by special threads (in parallel). The master and worker threads may communicate via shared data structures (objects), e.g. request queues, buffers, flags, etc. You may use (concurrent) collections classes from the [java.util](https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html) 

(<https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html>) and [java.concurrent.util](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html) 
(<https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html>) packages.

- The controller can be implemented either in C with Pthreads library, or in C using the OpenMP API, or in Java using multi-threading related classes and concurrent utilities.
- Implementing in C, you can control the elevator via TCP sockets. You can use the specially developed API (C-functions) for controlling Elevators via TCP sockets in C. The API hides TCP communication. Find links to the API under "Download" below.
- Using Java for implementation, you can control elevators either via TCP sockets or by means of Java RMI.


Report

The report should be structured as described on the Programming Project page. Specifically, the report on this project should include the following sections:

- Description of an algorithm used for selection of an elevator to service a request (button pressing)
- Description of implementation
- Explanation of synchronization mechanisms used in implementation (which mechanism, where and why).

Elevators: Documentation and Download

Documentation

- The description of [the primary class of the Elevators application](https://kth.instructure.com/courses/17204/files/2294893/download?wrap=1) (<https://kth.instructure.com/courses/17204/files/2294893/download?wrap=1>)
 - (<https://kth.instructure.com/courses/17204/files/2294893/download>) The description includes instructions how to run the application and how the elevators can be controlled.
- The complete documentation of the Elevators application in HTML can be found in [the zip file with Elevator application](https://kth.instructure.com/courses/17204/files/2294911/download?wrap=1) (<https://kth.instructure.com/courses/17204/files/2294911/download?wrap=1>)
 (<https://kth.instructure.com/courses/17204/files/2294911/download?wrap=1>) under the doc directory.

Download

- [A zip file with the Elevator application](https://kth.instructure.com/courses/17204/files/2294911/download?wrap=1) (<https://kth.instructure.com/courses/17204/files/2294911/download?wrap=1>) (including complete documentation, classes and jar file with all classes)
- An API (C-functions) for controlling Elevators via TCP sockets in C:
 - [for Linux](https://kth.instructure.com/courses/17204/files/2294810/download?wrap=1) (<https://kth.instructure.com/courses/17204/files/2294810/download?wrap=1>)
 - [for Windows](https://kth.instructure.com/courses/17204/files/2294892/download?wrap=1) (<https://kth.instructure.com/courses/17204/files/2294892/download?wrap=1>)

The API hides TCP communication. To learn how to use the API, look to the README file and the example code in test-hwAPI.c

Testing an Elevators' controller

In order to test your controller, you should execute the following tests (use cases) described below and compare the behavior of the Green Elevator (GE) with that what you expect from your controller algorithm.

- Before executing a test, make sure that all elevators are at the BV floor .

- "N up" ("N down") means press button up (or down) at the Nth floor.
- "Panel M" means press the panel button M of an elevator.

Tests (Use cases)

1) Run GE with 1 elevator, 6 floors:

```
java -classpath classes elevator.Elevators -number 1 -top 5 -tcp
```

1. Guaranty of service: Press 3 up, panel 5, and then 3 down.
2. Test of starvation: Press 3 up. When the elevator is in between the 2nd and the 3rd floor, press 2 up, 3 down, 2 down.
3. Quality of service: Press 4 up, 3 up, 2 up
4. Quality of service: Press 2 up, 3 up, 4 up

2) Run GE with 2 elevator, 6 floors:

```
java -classpath classes elevator.Elevators -number 2 -top 5 -tcp
```

1. Press 3 up, 3 down
2. Press 3 up. When the moving elevator is in between 1st and 2nd floor, press 1 up, 2 down.

3) Run GE with 3 to 5 elevators, 6 floors:

```
java -classpath classes elevator.Elevators -number 3 -top 5 -tcp
```

- Replay all scripts above. Remember to compare with what you expected from the algorithm.