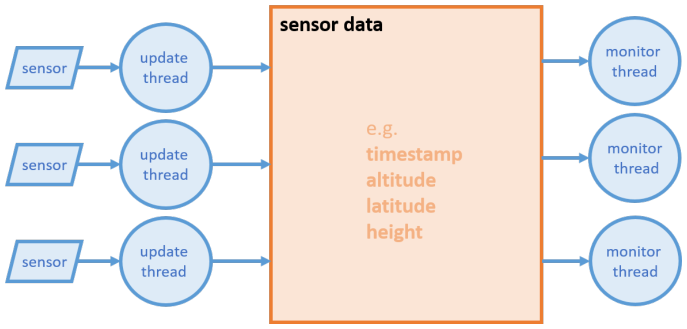
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Parallel Programming
Assignment 11: Advanced synchronization mechanisms
Spring Semester 2019

Assigned on: **13.05.2019** Due by: **20.05.2019**

Lock Free Sensor System

This exercise is about the lock-free implementation of a data record. We consider the following scenario: Assume you have a number of sensors that deliver complex data. It is considered clear from the beginning that retreiving the data from a sensor and in particular writing the data to some data structure cannot happen atomically. In this exercise we consider a sensor that delivers floating point data and an integer timestamp. You may want to think of a GPS sensor delivering, say, altitude, longitude and height together with the current time in milliseconds. In order to keep the system reliable and responsive, redundant data sources are installed, i.e. we have a lot of sensors that provide the same kind of data.



Now, we assume we have a lot of concurrent reader threads that want to monitor the sensor data while we have a moderate number of writer threads. What we consider most important is that the reader threads do always read a consistent data set. Secondly, we require that only the newest sensor data are written to the sensor data object. This is where the timestamp becomes important.

Implementation

Implement two versions of the sensor data class:

a) One blocking version based on a readers-writers lock (*LockedSensors.java*).

b) A lock-free version (*LockFreeSensors.java*)

Hints:

a) Before you implement the readers-writers lock based version, start with a simple locked version in order to understand. Then try a readers-writers lock but be aware that the Java-implementation does not give fairness guarantees. What can this imply? In any case, you have the code from the lecture slides presenting a fair RW-Lock implementation.

b) The lock-free implementation solutions does NOT rely on mechanisms such as Double-Compare-And-Swap. Also it does not rely on a lazy update mechanism. Somehow you have to make sure that with a single reference update you change all data at once. How?

Compare the efficiency of the two solutions. Experiment with different numbers of readers and writers and different timings in order to find out under which conditions which version is better. Start with the template that we provide.

Note: When testing your implementation using the provided Unit Tests make sure that you run the tests multiple times (i.e. 5-10 times). This increases the chance of finding an error in your implementation as the Unit Tests can succeed even with an invalid implementation due to randomness of scheduling.

Questions

a) Is your lock-free algorithm also wait-free? Please explain your answer.

b) Think about the solution you have provided: can you generalize this? Imagine, for example, the scenario of an expression tree that is updated by (a small amount of) writer threads sporadically but is used for evaluation by a huge number of reader threads. Can you use the same kind of mechanism?

Submission

In order to receive feedback for your exercises, you need to submit your code to the Git repository. You will find detailed instructions on how to install and set-up Eclipse for use with Git in Exercise 1.

Once you have completed the skeleton, commit it to Git by following the steps described below. For the questions that require written answers, please write them on paper and bring them to the next exercise session where the solutions will be discussed.

- **Check-in your project for the first time**
  - Right click your created project called **assignment11**.
  - In the menu go to **Team**, then click **Share Project**.
  - You should see a dialog ”Configure Git Repository”. Here, next to the Repository input field click on **Create...**
  - Select a root git directory or your projects that you have created in Execise 1. Note for all your assignments you should use the same directory.
  - click **Finish**.
- **Commit changes in your project**
  - Now that your project is connected to your git repository, you need to make sure that every time you change your code or your report, at the end you commit your changes and send (push) them to the git server.
  - Right click your project called **assignment11**.
  - In the menu go to **Team**, then click **Commit..**.

- In the Comment field, enter a comment that summarizes your changes.
- In the Files list, select all the files that you changed and want them to be committed. This typically includes all the Java files but not necessarily all the files (e.g., you dont have to commit setting files of our eclipse installation).
- Then, click on **Commit** to store the changes locally or **Commit and Push** to also upload them to the server. Note that in order to submit your solution you need to **both** commit and push your changes to the server.

- **Push changes to the git server**
  - Right click your project called **assignment11**.
  - In the menu go to **Team**, then click **Push Branch 'master'**. Note if this is not your fist push you can also use **Push to Upstream** to speed up the process.
  - A new dialog appears, now fill in for the URL field:
    `https://gitlab.inf.ethz.ch/COURSE-PPROG19/<nethz-username>.git`
  - Click **Next**
  - Keep the default values and click **Next**
  - An authentication dialog should appear. Fill in your nethz username and password and click **OK**.
  - Click **Finish** to confirm your changes. Note that eclipse might ask for authentication again.

- **Browse your repository online**
  - you can access and browse the files in your repository online on GitLab at:
    `https://gitlab.inf.ethz.ch/COURSE-PPROG19/<nethz-username>`