Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Parallel Programming
Assignment 8: More on synchronization
Spring Semester 2017

Assigned on: **11.04.2017** Due by: **24.04.2017**

Overview

In this exercise, we go over lock implementations and the usage of atomic operations for locking. This is to have a better understanding on when and why they are an interesting option over other locking mechanisms.

First, we will look at a theoretical exercise for locking to acquire a deeper understanding of the problems that might arise.

Then, we review the usage of atomic operations for synchronizing the access to shared data to multiple threads while they are executing concurrently.

And finally, some questions must be answered regarding the usage of optimistic concurrency control and the usage of locking (pessimistic concurrency control).

Getting Prepared

- Download the ZIP file named `assignment8.zip` on the course website.
- Import the project in Eclipse: Click on *File* in the top-menu, then select *Import*. In the dialog, select *Existing Projects into Workspace* under the *General* directory, then click on Next. In the new dialog, select the radiobox in front of *Select archive file* to import a ZIP file. Then, click Browse on the right side of the text-box to select the ZIP file you just downloaded from the website (`assignment8.zip`). After that, you should see assignment8 as a project under *Projects*. Click Finish.
- If you have done everything correctly, you should now have a project named assignment8 in your *Package Explorer*.

Analyzing locks

The code in `assignment8.livelock` package mimicks the behavior of a couple that are having dinner together, but they only have a single spoon. As they are so much in love, they want their significant one to start eating before if they are hungry. The current implementation "offers" mutual exclusion over the shared resource (the spoon). In this section, we need you to analyze the code within `assignment8.livelock` package and answer the following questions in your report:

a) Prove or disprove that the current implementation provides mutual exclusion. HINT: Similarly to what has been done in the lecture, you should first identify the important instructions of code, and then do the state space diagram for finding out if the current implementation provides mutual exclusion or not.

b) What is the problem with the current implementation? Argue how it can be improved.

Atomic (Read Modify Write) Operations

In this section, we will analyze how we can use atomic operations to perform concurrency control (often referred to as *optimistic* concurrency control) and the cost of using them when having high data contention. The sample code for generating pseudorandom numbers is based on a linear congruential generation which is explained below.

A linear congruential generator (LCG) is an algorithm that yields a sequence of pseudo-randomized numbers calculated by using a recurrence to generate a series of integer values based on an initial seed value. The method represents one of the oldest and best-known pseudorandom number generator algorithms (java.util.Random works this way).

Thus, whenever a caller wants to obtain the next value of the sequence, the pesudoramdon generator must:

a) Get the current seed.

b) Generate the next seed using the recurrence equation.

c) Save the updated seed value.

d) Return the value derived from the computed next seed value.

The first three steps have to be done atomically because if different threads try to get the next number of the sequence concurrently, these threads must not get the same number. There are different options for ensuring atomicity. One option is to use a lock, but if many threads are using the pseudorandom generator at the same time, this could result in thread contention and therefore in low performance.

The sample code is simple and it is only for showing the cost of using atomic operations. In the code, we create several threads(`RandomWorker.class`) in which every thread computes the summation of a million random numbers. The purpose of this is to show data contention and cost of atomic operations when generating pseudonumbers with a LCG algorithm.

This task has two parts. In the first part you must complete the lock based implementation of the pseudorandom generator i.e. completing the class `LockedRandom` with the appropriate locking mechanism. In the second part, you have to answer the following questions on your report:

a) Given the code in the AtomicRandom.java class, analyze it and argue why or why not a single atomic operation for storing the seed is enough in this case. (HINT. The `compareAndSet` method tries to set a new value if stored value is the same as the one we read).

b) Execute the main program for different number of threads. Describe and explain the behavior of the results. For example, explain why do atomic operations become more expensive every time? Propose a possible solution if needed.

c) Describe in which cases optimistic concurrency control (usage of atomic operations) is a viable solution.

d) (Optional) Try to improve the performance of the setup using atomic operations.

Submission

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

Once you have completed the skeleton, commit it to SVN in a directory named `assignment8` by following the steps described below. The questions that require written answers should all be recorded in a single file named `report.pdf` and placed in the base directory of your project (i.e., in folder `assignment8`).

- **Check-in your project for the first time**
  - Right click your created project called **assignment8**.
  - In the menu go to **Team**, then click **Share Project**.
  - In the dialog that now appears, select **SVN** as a repository type, then click **Next**.
  - In case you have submitted Exercise 1, choose **Use existing repository location** and select the pre-defined URL in the dialog that should look like this
    https://svn.inf.ethz.ch/svn/vechev/pprog17/students/NETHZ_USERNAME
    Click Finish. Otherwise follow the steps in Exercise 1 to set-up a repository location.
- **Commit changes in your project**
  - Now that your project is connected with the SVN server, you need to make sure that every time you change your code or your report, at the end you submit it to the SVN server as well.
  - Right click your project called **assignment8**.
  - In the menu go to **Team**, then click **Commit**.
  - In the Comment field, enter a comment that summarizes your changes.
  - Then, click on **Ok**.