Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Parallel Programming
Assignment 10: Advanced synchronization mechanisms
Spring Semester 2020

Assigned on: **27.04.2020** Due by: **04.05.2020**

Overview

In this exercise we will try to implement Lock conditions and try to familiarize ourselves with Java's Monitor implementation. In the second problem we will explore Semaphores and Barriers. In the example code we will see how they work and afterwards will implement our own versions of them.

Monitors, Conditions and Bridges

You are in charge of a bridge. A structural engineer tells you, that there is a problem and the bridge can currently carry only a very limited load. Thus you came up with the following regulation: Only either 3 cars or one truck may be on the bridge at each moment. To see how this impacts traffic you run a Java simulation.

a) In `BridgeMonitor` you are given a code skeleton for this. Finish the implementation using a Java object as monitor. All Java objects allow you to use `wait()`, `notify()` and `notifyAll()`.

b) In `BridgeCondition` you are given a similar code skeleton. Finish this implementation using a Lock and it's condition interface.

Hint: Both classes `BridgeMonitor` and `BridgeCondition` have a main method that starts a run with test data.

Semaphores and Databases

A Database application allows - for performance reasons - only 10 concurrent users. In the lecture you have seen the semaphore as a tool to enforce such conditions. The Database program also uses a backup algorithm, that works as follows:

- For all currently logged in users run Part A of the algorithm locally.
- Once all users finished Part A, run Part B for all users locally.
- Once all users finished Part B locally, run the last part of the algorithm on the Database server.

As a tool to enforce such conditions as needed here you have seen Barriers in the lecture.

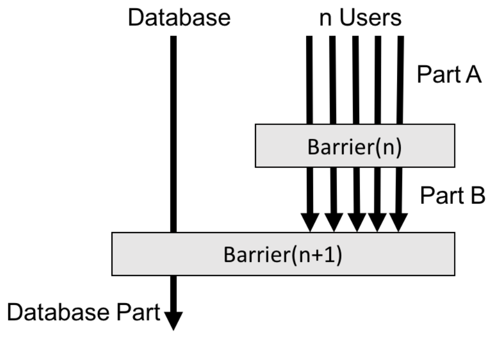

Figure 1: Backup Algorithm

In `DatabaseJava` you can see a reference implementation of this. It uses Java's `Semaphore` and `CyclicBarrier`. Your task is to implement `MySemaphore` and `MyBarrier` as drop-in replacements for those classes.

a) Implement your own semaphore in `MySemaphore`. Your implementation should be based on a Monitor.

b) Implement your own barrier in `MyBarrier`. Your implementation should be based on a Monitor. Compare this implementation with the implementation using Semaphores from the Lecture slides. You will find that Monitors make it a lot easier.

c) Explain the different ways to implement Semaphores (those shown in Lecture and Exercise session) with their pros and cons.

You can test both implementations by running the main methods in `DatabaseJava` and `DatabaseMyCustom`. If you have implemented `MySemaphore` correctly they should behave very similarly.

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 **assignment10**.
  - 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 **assignment10**.
  - 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 **assignment10**.
  - 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-PPROG20/`*<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-PPROG20/`*<nethz-username>*