DT271C - Seminar 2 - Thread synchronization

Introduction

The purpose of this Seminar is to study some classic synchronization. For each task you will be given some code that serves as a starting point. You should then modify the code to solve the given tasks.

Read the complete document before you start with the specific tasks.

Good luck!

1 Preparations

Before starting with the programming, you are expected to do the following tasks. You need to understand the objective of the seminar and do the following preparations:

1. Read about the Reader-Writer problem. In the textbook you can find a description in 7.1.2. Make sure that you understand the different starvation problems that can occur. Also understand the difference between the first and second reader-writer problem.
2. Read about the Dining Philosophers problem. In the textbook you can find a description in 7.1.3.
3. Study the monitor concept, and how it can be realized in Java using synchronized, wait and notify.

2 Programming tasks

In this seminar we have three major tasks. The first and second tasks are about the reader-writer problem. In exercise 2 we solved the first reader-writer problem with semaphores. Here we will use synchronized instead. **All modifications that you need to do for task 1 and 2, can be done in the class RWLock**.

The third task is the Dining Philosophers problem which is useful for studying deadlock. **All modifications that you need to do can be done in the class Table**.

2.1 Task 1

- Compile the code for ReaderWriter that is given in Seminar2.zip. How can you see that we have an inconsistent state? Try to formulate why we have this problem?
- First solve the first reader-writer problem. In the first reader writer problem a reader should wait only if a writer has locked the data. A writer must wait if readers or writers are active. **Use Java synchronized and wait/notify to solve the problem**.Add suitable variables to be able to detect when a thread should wait.

  How can you identify the starvation problem?

2.2 Task 2

- To solve the problem with starving writers in the first reader-writer problem a new condition for the reader might be added. Here the following additional is added: A new reader are not allowed to start if there are writers waiting for the data. This is defined as the second reader-writer problem.

  The motivation for the second reader-writer problem is that it is better to give a writer higher priority because then the readers will read the freshest value.

  Add variables to be able to detect waiting writers and modify your code.
- Now we will have the problem with reader starvation. Try to identify this in your code.

  A solution is to modify the conditions in the reader-writer problem to get a starvation free solution. Instead of modifying your synchronization code with additional condition but we will use a totally different approach

  The Java library contains a solution to the reader writer problem and provides a reader-writer primitive. The documentation can be found at: https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReentrantLock.html

  Start over with the original code given from Canvas. Instead of implementing your own solution use the ReentrantReadWriteLock from the library. Set fairness to true to have a starvation free solution.

2.3 Task 3

- Compile and run the code for DiningPhilosophers that is given in Seminar2.zip. Can you detect in the written output that too many chopsticks are used?
- Implement a version where a philosopher **waits if a chopstick is already used by another philosopher**. (You can use synchronized or Semaphores.) In this task each philosopher should first pick up the left and then the right chopstick.

Run the code a couple of times until you observe **deadlock**. (If it is hard to get a deadlock, modify the times in sleep.

- Implement a **deadlock free implementation**. Hint! Which of the four conditions for a deadlock do you think is the simplest to solve? How can you describe this in Java code?

3 Final observations

Before you present your seminar, make sure that you have the answer to the questions given in the preparations. Also write down the following for each task to discuss during the seminar.

- How did you implement the task?
- Why did you solve it in the way you did it?
- What difference in behaviour did you notice?