CONCSYS - ASSIGNMENT 1

Upload your solution on ILIAS as a ZIP or TAR.GZ including all your files (sources, text file with explanations, ...). If you have questions, please address them during the lab or by email to yaroslav.hayduk@unine.ch and maria.carpen-amarie@unine.ch.

Deadline: 9:00 AM, March 22, 2016

Exercise 1

1. Write a Java program that spawns multiple threads. All threads access a shared counter (initialized to 0) in a loop (of *i* iterations). All threads read the counter to a local variable (on the stack), modify the variable (as follows) and then store it back to the counter. The threads are split in two groups with respect to the modification of the variable: threads in first group will increment the variable, threads in second group will decrement it. We define *n*, number of threads in first group, and *m*, number of threads in the second group. *n*, *m* and *i* will be given as runtime arguments to the program. When all threads finish, the program prints the value of the shared counter and the duration of the execution. (Check the value for equal number of incrementing and decrementing threads, i.e. n = m, and for i = 100 000 iterations). Save the source to Ex1NoSync.java.

2. Modify this program using the keyword *synchronized* to protect the counter. The counter must be 0 at the end of the execution. Save your code to Ex1Sync.java.

3. Modify the first program to use java.util.concurrent.locks.ReentrantLock class. The counter must be 0 at the end of the execution. Save your code to Ex1ReentrantLock.java.

4. Run the 3 programs with equal sets of 1, 2, 4, and 8 threads and report the results in a textual table in a file Ex1.txt. Please report run time in ms, compute the speedup (Ex1NoSync as reference) and your machine specification (processor specs and operating system - e.g: the result of cat /proc/cpuinfo for Linux OS).

Exercise 2: Producer-Consumer problem

Suppose we have two types of threads: Producers and Consumers, sharing a circular buffer. Each Producer deposits data at a suitable position in the buffer, denoted by a variable *in* (i.e, the next position available in the buffer) and advances the variable *in*, while each Consumer retrieves the data item at the position denoted by the variable *out* and advances this variable. A producer cannot deposit its data if the buffer is full. Similarly, a consumer cannot retrieve any data if the buffer is empty.

Write a program that can correctly coordinate the producers and consumers and their depositing and retrieving activities. For simplicity, the number of Producer threads is equal to the number of Consumer threads and denoted by *t* (program argument). The buffer will have *n* integer elements (program argument).

Hints: Define a CircularBuffer class with two principal methods: produce() and consume(). Use Java semaphores for synchronization. In method main, start the required number of threads, which will share a CircularBuffer object.

Some tips and notes

- Use System.nanoTime() before and after the execution to measure the run time (try to make the measurements as fine grained/accurate as possible)
- Java API documentation : https://docs.oracle.com/javase/8/docs/api/.
- The names of the files are just suggestions for better organizing the solutions. You can obviously split your code in multiple files as you think it is fit, but in this case is recommended to add a note to the submitted assignment stating what and where was solved.