HÁSKÓLI ÍSLANDS
Iðnaðarverkfræði-, vélaverkfræði- og tölvunarfræðideild
TÖL401G: Stýrikerfi / Operating Systems · Vormisseri 2022

Assignments 11–12 · To be solved until 01.3.2022, 13:00

Assignment 11

1. Implement a Java program that creates two new threads that are in a race condition. Use the problem from slide 6-6 and 6-7 as example, however you do not use a spooler directory, i.e. just increment a shared variable `in` by copying it into a variable `next_free_slot` that is local to each thread (comparable to a CPU register – in fact the Java compiler/virtual machine likely use a CPU register for that local variable), incrementing that variable by 1, and writing it back to `in`.

   `in` shall be initialised with 0. Each of the two threads shall increment the `in` in a loop using the same number of iterations (the number is passed in as command line parameter) and terminate after that number of incrementations. Use the `join()` method to wait for the termination of the two threads. After that, the value of `in` is printed out.

   As starting point, use the files from `tol401g_assignment11.zip` that you find in Canvas

   - File `Assignment11.java` contains the main method that reads the number of iterations to be executed by *each* thread and prints the result out – **do not change that file**!
   - Use file `MyAssignment11.java` to add your implementation there (it gets called by file `Assignment11.java`). Feel free to change that file except:
     (a) Do not change the name of this class (adding `extends` is OK) nor put it into another package.
     (b) Do not modify the name and input and output parameter of the main method.
   - You are allowed to add further classes.

   Hints:

   - As each thread is executed by (or within) two *different* objects (created by you via `new`), it is not sufficient to define `in` as ordinary field (outside of Java, fields are known as attributes) within the two thread objects. The object-oriented concept of encapsulation requires to define the shared variable either as `static` so that it is shared by all instances of that class or to introduce a separate class (e.g. named `Counter` with, e.g., a method `increment()`) or a public `in` variable which are instantiated by you and passed to both thread instances.
   - Depending on how threads are scheduled on your machine, the data type `int` may not be large enough to provoke a race condition, hence you better use the data type `long` for your counters and number of iterations variables.
   - You need to define your variable that is shared by the threads as `volatile` (see slide 6-41), e.g. `volatile long in`.

2. Run your program: is the printed value of `in` the expected value? How would you see that a race condition occurred?

3. What command line parameter value demonstrates typically at *each* run the race condition on your computer? What is the name and version of your OS (e.g. "MS Windows 10")?

Assignment 12

1. Implement Peterson's algorithm in Java to protect the critical section from Assignment 11. (The hints for assignment 11 apply as well for this assignment!)

   Comparable to Assignment 11, use as starting point the files from `tol401g_assignment12.zip` that you find in Canvas:

   - File `Assignment12.java` contains the main method that reads the number of iterations to be executed by *each* thread and prints the result out – **do not change that file**!
   - Use file `MyAssignment12.java` to add your implementation there (it gets called by file `Assignment12.java`). Feel free to change that file except:
     (a) Do not change the name of this class (adding `extends` is OK) nor put it into another package.
     (b) Do not modify the name and input and output parameter of the main method.
   - You are allowed to add further classes.

2. Run your program to convince yourself that your implementation solves the critical section problem! (Like in assignment 11, you should experiment with different command line parameters for the number of iterations.) No need to answer anything for this question – just convince yourself and observe changes in run time due to busy waiting.

General comments concerning Java assignment submission

For Java programming assignments, a special programming assignment Gradescope mode is used, hence **do not submit any PDF**, only the Java source code (`*.java` files plus a readme file) of your programming solutions – the compiled byte code (`*.class` files) is not required (Gradescope will compile your code).

1. Use **zip format** to create a single file archive of the multiple source code files! Use the same structure as in the `tol401g_assignment11.zip` and `tol401g_assignment12.zip` files, i.e., no sub-directories, no Java package structure.

2. For Assignment 11 parts 2 and 3, provide inside the zip file a short text file **readme.txt** that contains answers to the questions in part 2 and 3.

3. If Gradescope was able to compile the source from your uploaded zip file, it will run same basic sanity checks against your solution. **Failing already compilation or all or the majority of these checks means that something is wrong with your submission and you are likely to score 0 points** in the manual grading: Typically,

compilation has failed because your zip file does not adhere to the structure described above. To fix these problems, you are can resubmit as often as you like within the deadline.

4. For assignment 11, the Java source code (part 1) and the **readme.txt** (for parts 2 and 3) need to be submitted

5. For assignment 12, just upload the source code (part 1). You do not really need to answer part 2: rather convince yourself whether your solution works or not.

Preparation

Read the first part of chapter 6, sections 6.1 to 6.4 (up to slide 6-52). (Remaining sections of chapter 6 will be covered the week after.)

The human brain is not used to handle things happening in parallel, but parallel programming and the necessary synchronisation of parallel processes/threads are needed to make use of today's multi core architectures. Therefore, this chapter is important and that synchronisation is a chosen functional candidate for the final exam.
TL;DR: difficult, make sure to understand this chapter: read slides, watch videos, play the computer game as instructed on the slides, ask during class and on Canvas/Ed-discussion.