CONCSYS - ASSIGNMENT 2

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, on the forum or by email to maria.carpen-amarie@unine.ch.

Deadline: 9:00 AM, April 4, 2017

Exercise 1

Case 1: Write a program (Ex1) that invokes n threads using Peterson's generalized algorithm in order to protect a shared counter. Each thread increments the counter. In addition, each thread will count its own accesses to the shared counter separately in one field of an array of size n (each thread will have associated one element of the array). Define the shared counter as volatile. The threads will stop when the counter reaches 300000. The program prints the final value of the counter and the number of accesses per thread (the values from the mentioned array). Peterson's algorithm for n threads is described in the lecture's notes (the Filter algorithm) and also in the Concurrency lab page.
Hints: Use a lower counter limit (e.g. 1000) in case of long execution delays. For the level and victim array `AtomicInteger` is the recommended type to use (using volatile for simple integer arrays will not make the array elements volatile, but only the reference)

Case 2: Change Ex1, by making the shared variables non-volatile.
You should not use other Java synchronization mechanisms in your solution besides declaring variables as Atomic types or volatile when/if you think this is needed. For each of the cases above, run the program in the specified form on a single-processor (Check tips at the end of this document) and on a multi-processor using Sun Fire T2000 with 4 and 8 threads. Write in a Ex1.txt file your statistics and the interpretation of the results in a short paragraph. The statistics data should include: case, counter value, lowest number of accesses in the critical section for a thread, highest number of accesses in the critical section for a thread, number of threads and the execution time.
Hints: Shared variables here refer to the ones not included in the Lock implementation (not the level and victim arrays) - essentially the shared variable will be the counter incremented by the threads. The purpose is to observe any difference in the time taken to modify "normal" variables and volatile ones.

Exercise 2

Why is the generalized Peterson algorithm unfair? Try to describe an example that shows the lack of fairness. Do you have any idea of how the algorithm can be enhanced to provide fairness? Give your answer and explain your reasoning in a file named Ex2.txt.

Hint: Consider 3 threads (or more) and exemplify a situation when one of them is not guaranteed to get into the critical section.

Exercise 3

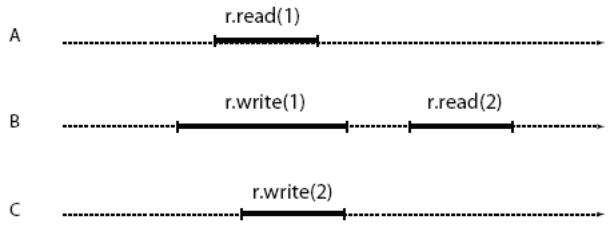

Figure 1: First execution history

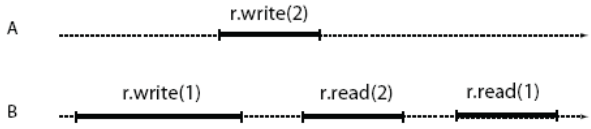

Figure 2: Second execution history

For each of the histories shown in Figures 1 and 2, are they (1)sequentially consistent, or (2)linearizable? Justify your answer in a file named Ex3.txt.

Tips

This function permits to use only one processor on Solaris OS (Sun Fire T2000):

```
public static void setSolarisAffinity() {
  try {
        // retrieve process
       id String pid_name =
       java.lang.management.ManagementFactory.getRuntimeMXBean().getName();

        String [] pid_array = pid_name.split("@");
        int pid = Integer.parseInt( pid_array[0] );
        // random processor
        int processor = new java.util.Random().nextInt( 32 );
        // Set process affinity to one processor (on Solaris)
        Process p = Runtime.getRuntime().exec("/usr/sbin/pbind -b ");

        p.waitFor();
      }
catch (Exception err) {
        err.printStackTrace();
}
}
```