ESIR SPP – TP1 & 2 (non noté)

Exercise 1:

In Java write a multithreaded program that manipulates one shared long variable so that:

- 5 threads increment this variable 100,000 times;
- 15 other threads read this variable 100,000 times, and write out what they have read onto the console every 20,000 iterations (with their ID as prefix).

1 – Write a first version of your program without any synchronisation. What do you observe? Why do you think this is the case?

2 – Make your program thread-safe with a normal re-entrant lock. Measure the execution time taken by your program.

3 – Write a second version of your program, in which you replace the normal lock by a read/write lock. Measure the execution time of your program. What do you observe? Why do you think this is the case?

4 – We will now artificially increase the cost of the integer operations performed on the shared variable (both increment and read), by using `Thread.sleep(..)`, and reassess the effect of a read/write lock over a re-entrant lock. To this aim, write two new versions of your program which:

- only use 1000 iterations of each loop. Set the console printouts to occur every 200 iterations.
- artificially delay each operation on the shared variable by 1ms using `sleep(..)`.

One version should use standard re-entrant locks, the other read-write locks. Measure and compare the times obtain. How do you interpret them?

Exercise 2:

1 – Implement your own version of a read/write lock using non-reentrant locks (e.g. binary semaphores) in Java. You should have one Class called "MyRWLock" with four methods "lockRead()", "lockWrite()", "unlockRead()" and "unlockWrite()".

2 – Why do you need non-rentrant locks?

3 – Use this implementation in the code of exercise 2. Measure the execution time of your program. Do you observe any difference?

4 – (Optional) Merge both unlock operations into one single method.

Exercise 3:

In Java write a multithreaded program that manipulates one instance of ArrayList<E> containing Long integers (class Long) so that:

- 10 threads insert their ID into the list in random positions 10,000 times;
- 10 other threads delete random elements of the list 10,000 times;
- one observer thread prints out the size of the list every 0.05s (50ms).

1 – Write a first version of your program without any synchronisation. What do you observe? Why do you think this is the case?

2 – Make your program thread-safe using a monitor. Do you need to use signal and wait?

3 – Measure the execution time of your program with 3, 7, 11, … up to 43 threads. Draw a chart of your measurements. What do you observe?