Readers/Writers Lock

In the course we gave a naive solution to the readers/writers problem using π-calculus primitives:

$$\begin{array}{rcl} RW_0 & = & sr.RW_1 + sw.ew.RW_0 \\ RW_n & = & sr.RW_{n+1} + er.RW_{n-1} \quad (n > 0) \end{array}$$

Unfortunately this solution can lead to writer starvation if there are continuously new readers coming because in this case we never leave the state RW_n ($n > 0$).

The goal of this exercise is to explore two ways of modifying the naive solution to solve the writer starvation problem, implement these new solutions in Scala and compare them.

Solution 1

The first solution is the one explained in the course which uses an extra queue process Q for forbidding new readers to read as soon as a writer emitted its will to write. We remind here the description given in the course:

$$\begin{array}{rcl} RW_0' & = & sr'.RW_1 + sw'.ew.RW_0 \\ RW_n & = & sr'.RW_{n+1} + er.RW_{n-1} \quad (n > 0) \\ \\ Q & = & sr.\overline{sr'}.Q + ww.\overline{sw'}.sw.Q \end{array}$$

Download the file rwlock.scala *from the course page and write a class* ReadWriteLock4 *which implements this solution. You can copy and paste the class* ReadWriteLock3 *which is the naive solution and modify it.*

Solution 2

The first solution did not require to modify the naive read/write lock but only to append a queue for controlling access to it.

The idea is now to count the number of pending writers in extending the states of the naive solution with an extra index. From the first solution we keep the idea of making always a ww precede a sw. This will naturally lead to two kinds of states:

- $Read(m, n)$ representing a state where there are m reading readers and n pending writers.
- $Write(n)$ representing a state where there are 1 writing writer and $n - 1$ pending writers.

The system will be initially in the state $Read(0, 0)$.

Write a new class ReadWriteLock5 *inspired by the naive solution which implements this solution.*

Discussion

Are these two implementations equivalent ? Why ?