Distributed Algorithms

Fall 2019

Shared Memory
9th exercise session, 25/11/2019

Matteo Monti <matteo.monti@epfl.ch>
Athanasios Xygkis <athanasios.xygkis@epfl.ch>

Exercise 1 - Majority voting

Explain why every process needs to maintain a copy of the register value in the "Majority voting"[(1)] algorithm.

(1) [ABD95, slides 24 and following]

Exercise 1 (Solution)

Notice that the algorithm also needs to maintain a copy of the register value at all processes, even if we assume only one reader.

- Assume that some process *q* does not maintain a copy.
- Assume, furthermore, that the writer updates the value of the register: it can do so by only accessing a majority of the processes.

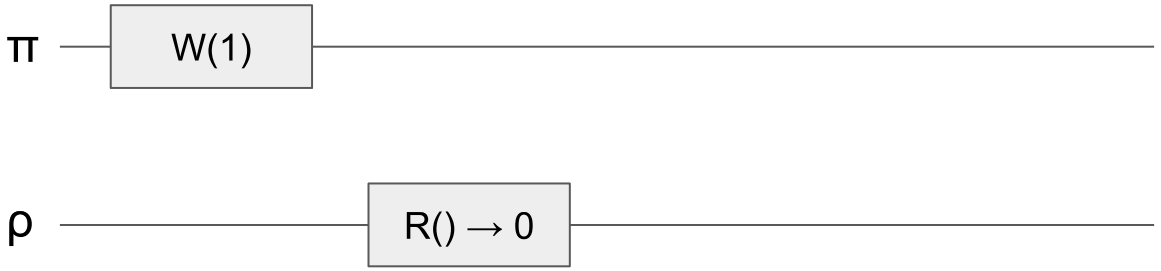
- If *q* is in that majority, then the writer would have stored the value in a majority of the processes minus one. It might happen that all processes in that majority, except for *q*, crash. However, the set of remaining processes plus *q* also constitutes a majority. A subsequent read in this majority might not return the last value written.

Exercise 2 - Unsafe execution

Consider a system with two processes, *π* and *ρ*. Give a register execution such that each process performs at most two operations and the execution is **unsafe**.

Exercise 2 (Solution)

Assume that the register is initialized to 0.



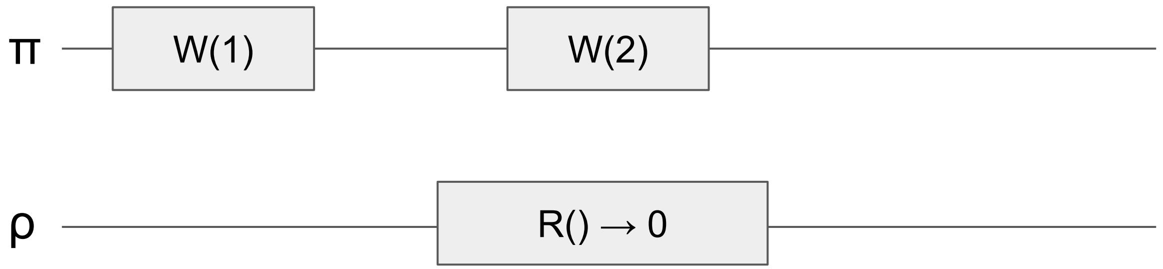
This execution is not safe, because the Read happens after (instead of concurrently to) the Write. Therefore this read should return the last value written.

Exercise 3 - Safe execution

Consider a system with two processes, π and ρ. Give a register execution such that each process performs at most two operations and the execution is **safe** but not **regular**.

Exercise 3 (Solution)

Assume that the register is initialized to 0.



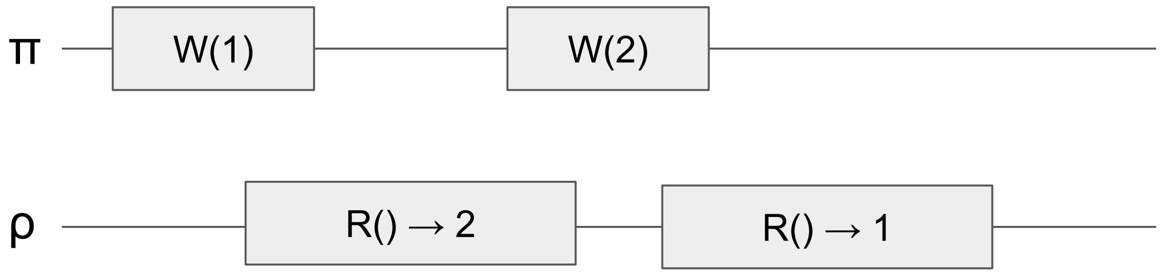
This execution is safe, because when a Read is concurrent to a Write, the Read is allowed to return any value from the universe of values of the register. If the execution was regular, the Read should have returned either the last written value (i.e. 1) or the concurrent written value (i.e. 2).

Exercise 4 - Regular execution

Consider a system with two processes, *π* and *ρ*. Give a register execution such that each process performs at most two operations and the execution is **regular** but not **atomic**.

Exercise 4 (Solution)

Assume that the register is initialized to 0.



This execution is regular, because every read returns either the last value written or the concurrent value written. The execution is not atomic because the second read returns a value that is older that the value returned by the first read.
Said differently, since the first read returns 2, it means that its linearization point is after the linearization point of the second write. Therefore, no matter where we place the linearization point of the second read, it will always be after the linearization point of the second write. This means that the second write has to return 2!

Exercise 5 - Timestamps

Explain why a timestamp is needed in the “Majority voting”(1) algorithm, but not in the “Read-one, write-all”(2) algorithm.

(1) [ABD95, slides 24 and following]
(2) [Slides 16 and following]

Exercise 5 (Solution)

The timestamp is needed precisely because we do not make use of a perfect failure detector.

Without the use of any timestamp, a reader *q* would not have any means of comparing different values from any read majority:

- In particular, if process *p* first writes a value *v* and subsequently writes a value *w*, but does not access the same majority in both cases, then *q* (which is supposed to return *w*) might have no information about which value is the latest. The timestamp is used to distinguish the values and to help the reader with determining the latest written value.
- Such a timestamp is not needed in the “Read-Impose Write-All” algorithm because the writer always accesses all processes that did not crash. The writer can do so because it relies on a perfect failure detector. It is not possible that a reader can obtain different values from the processes, as in the Majority Voting algorithm.