Exercise Session 9
Shared Memory

xx

December 3, 2012

Problem 1

Explain why every process needs to maintain a copy of the register value in the "Majority Voting" algorithm (Either Algorithm 4.2 in the book, or ABD95 algorithm from slides).

Solution

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 only by 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. But the set of remaining processes plus q also constitute a majority. A subsequent read in this majority might not return the last value written.

Problem 2

Why does the *Read* also return the timestamp on the fail-silent algorithm?

Solution

The reader also needs read ids to differentiate replies to consecutive read queries. Suppose that a reader p emits a read request $r1$, gets a majority of replies, performs some local computation, then emits a read request $r2$, gets a majority of replies, performs local computation, then emits a read request $r3$. It might happen that p now gets late replies from $r1$ and late replies from $r2$; both are minorities in the case of $r1$ and $r2$, but together they can form a majority (e.g., replies from $p1, p2, p3$ are late replies to $r1$ and replies from $p4, p5, p6$ are late for $r2$). That means that p might gather a majority of stale replies and return old data for the read. The reader ids solve this issue by differentiating replies for $r1$, $r2$ and $r3$.

Problem 3

Explain why a timestamp is needed in the "Majority Voting" algorithm (Algorithm 4.2 in the book, or ABD95 algorithm from slides), but not in the "Read-One Write-All" algorithm (Algorithm 4.1, or the Algorithm from slide 21).

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 to compare 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.