Distributed Algorithms, Bonus Exam

xx

December 13, 2012

Name:
Sciper number:

Assume that we work in the fail-stop model and consider Algorithm 1.

Recall that in the fail-stop model we consider a set of n processes $\{p_1, p_2, \ldots, p_n\}$ and we suppose that any process may stop taking steps at any time and that all processes have access to a perfect failure detector. Moreover, when a process stops taking steps, it never takes any step again.

Observe that Algorithm 1 is similar to an Algorithm you saw in the course and that both solve non-uniform consensus in the fail-stop model.

Your task is to answer the following question:

Is it possible to obtain a uniform consensus algorithm by modifying Algorithm 1 as follows?

- Adding new variables to the algorithm and possibly initializing them in the $\langle cons \mid Init \rangle$ handler.
- Replacing the pseudo-code of the $\langle urb, Deliver \mid round, v \rangle$ handler (lines 18 to 22) by a new block of pseudo-code that may do any local computation but that must not trigger any event other than the *Decide* event.

To clarify, by local computation we mean variable assignment, "if" blocks, and "while" blocks. Your modifications should not contain any trigger event except for the *Decide* event.

Write a detailed explanation of your answer, focusing on both the Agreement and Termination properties.

```
Implements:
    Consensus (cons)

Uses:
    UniformReliableBroadcast (urb)
    PerfectFailureDetector (P)

upon event ⟨cons, Init⟩ do
    suspected ← ∅
    round ← 1
    currentProposal ← ⊥
    delivered ← [false]^N

upon event ⟨P, Crash | p⟩ do
    suspected ← suspected ∪ {p}

upon event ⟨cons, Propose | v⟩ do if currentProposal = ⊥ then
    end
    currentProposal ← v

upon event ⟨urb, Deliver | round, v⟩ do if round = i then
    end
    trigger ⟨cons, Decide | v⟩

currentProposal ← v
delivered[round] ← true

upon event delivered[round] = true or p_round ∈ suspected do
round ← round + 1

upon event round = i and currentProposal ≠ ⊥ do
trigger ⟨urb, Broadcast | p_i, currentProposal⟩
```

Algorithm 1: Non-Uniform consensus algorithm, code for process p_i

Solution

The resulting algorithm would not solve uniform consensus.

Agreement

First, let us consider agreement.

Suppose that a process decides before hearing from everybody else (either receiving a message or a crash notification). In other words, suppose that p_i decides after receiving a message or the crash notification from a set of processes S, where $S \subset \Pi$, $S \neq \Pi$. Suppose there is a correct process p_j such that p_i did not deliver any message from p_j before deciding.

Suppose now that p_i crashes, along with all processes in S. Suppose that p_j receives the crash notifications for p_i and all processes in S before it delivers the urb broadcasts. p_j cannot know whether the processes in S broadcast anything before crashing, therefore p_j must decide on a value without waiting for a deliver. Since p_j did not receive the value proposed by p_i or by the processes in S, p_j will decide on a different value. Therefore, p_i and p_j will decide differently, violating uniform agreement. This means that p_i cannot decide before hearing a message or a crash notification from every other process. (Basically, in the round-based algorithm, it means that processes must either use acknowledgments from every correct process or wait until the final round before deciding)

Termination

Now let us consider termination.

We proved previously that we need to wait for a message or crash notification from every process before deciding.

Consider now a scenario where there is a set S of correct processes and a faulty process p_i, $p_i \notin S$, where i is greater than the ids of all processes in S. Suppose that every process $p \in S$ receives a message from every other process in S, but not from p_i. This means that all processes are waiting to hear a message from p_i or to hear of p_i's crash before deciding. Since the *urbDeliver* function only accepts messages from a particular sender (the sender id must be equal to the local *round* variable) and all processes are waiting only for a message from p_i, it means that the local *round* of at least one process is equal to i (p_i's id). We denote that process by p_j.

Suppose that p_i crashes before broadcasting any message. Notice that the crash handler of the correct processes will add p_i to *suspected* and increment the local round number. Therefore, the local *round* of p_j will become $i + 1$. However, since in our scenario i is the highest id, there is no process p_{i+1}. This means that p_j will now wait forever for messages from a process that does not exist. The *urbDeliver* of p_j will never be triggered again, meaning that p_j will never decide and termination will be violated.

Elegant proof by one of your colleagues

Suppose that there is a single correct process, p_i. This means that p_i must decide when $round = i$, since otherwise it will never terminate (the urbDeliver code will only be triggered exactly once). However, if a process decides in round i and then crashes, it might violate agreement (similar proof to above).