Homework on Mutual exclusion for Algorithmic approach to distributed computing

Gabriele Genovese

October 2024

1 Exercise 1: the Suzuki-Kasami algorithm

Let's prove that Suzuki-Kasami algorithm verifies the three properties of mutual exclusion.

The three properties of mutual exclusion are:
1. **Safety**: only one process can be in the critical section at any given time;
2. **Liveness** or freedom of deadlock: at least one process is eligible to enter critical section;
3. **Fairness** or absence of starvation: every process trying to enter in a critical section must eventually succeed.

1.1 Proof of safety

Only the process that holds the token can enter the critical section. Since only one token exists in the system, only one process can hold the token at any given time, ensuring that no two processes can be in the critical section simultaneously.

1.2 Proof of liveness

The algorithm guarantees that if a process requests the critical section and there is no failure, it will eventually get the token. Once a process has the token, it will execute in finite time its critical section and then pass the token to the next process that has requested it.

1.3 Proof of fairness

The algorithm maintains a request number array and processes requests in the order they are received. A process is allowed to enter the critical section when its request number is higher by one than what has already been processed. Since each process will eventually get the token in order of their request, this prevents starvation. If a process k that never requested the token do the request (then, $\text{req}[k] = 1$), the following expression should be false:

$$k \notin Q \land \text{req}[k] = \text{last}[k] + 1$$

This is impossible because $\text{last}[k]$ is 0. Thus, the condition is true and it the process will be added to the queue of the requesting processes. Using the FIFO serving policy, the process will eventualy enter in the critical section.

2 Exercise 2: the quorum-based algorithm

Let's prove that the quorum-based algorithm satisfies safety but does not satisfy liveness.

2.1 Safety:

The quorum-based algorithm guarantees safety thanks to the intersection between nodes. By definition, any two quorums Q_1 and Q_2 must intersect at least at one node, say N.

- If process P_1 has obtained permission from quorum Q_1 to enter the critical section, this means that node N, which is part of both quorums, has already granted permission to P_1.
- Since N cannot grant permission to another process simultaneously (because it's already committed to P_1), it will not grant permission to process P_2 in quorum Q_2 until P_1 releases the critical section.

Thus, process P_2 cannot obtain permission to enter the critical section until P_1 completes its execution and releases the token.

2.2 Liveness:

The quorum-based algorithm does not satisfy liveness due to the possibility of deadlock.

Suppose two processes N_1 and N_2 send a REQUEST message at the same time to enter the critical section. Both N_1 and N_2 share two common quorum members, say P_1 and P_2. If P_1 grants permission (sends an ACK message) to N_1 and P_2 grants permission to N_2, both processes N_1 and N_2 will be waiting for the other quorum member's permission to complete the set. N_1 is waiting for an ACK from P_2, and N_2 is waiting for an ACK from P_1. Since neither process can receive the final permission, they will be stuck in a cycle, waiting forever.