Answers to Homework #3

Due Date: February 27, 2001

1. (20 points) Show that in Lamport’s algorithm the critical section is accessed according to the increasing order of timestamps. (text, problem 6.7, p. 149)

Answer: Recall that two basic assumptions of Lamport’s algorithm (or any other distributed mutual exclusion algorithm, for that matter) is that messages sent from process p to process q arrive in the order they are sent, and if a message is sent then it will arrive (i.e., no messages are lost).

Proof by contradiction. Suppose process p₁ issues a request to enter the critical section at time t₁, p₂ issues a similar request at time t₂ with t₁ < t₂, and p₂ enters first. This means that p₂’s request is at the head of its queue. As the queues are ordered by timestamp, this means p₁’s request has not arrived. If p₂ enters, though, it also received a message from p₁ with a timestamp higher than t₂. This implies that p₁’s request has a timestamp higher than t₂ (which is false as t₁ < t₂) or p₂ never received p₁’s request. The latter is possible only if either p₁’s request was lost, or messages from p₁ to p₂ arrive out of order. Both these contradict the above basic assumptions. Hence p₂ cannot enter the critical section first, proving the claim.

2. (20 points) Show that in the Ricart-Agrawala algorithm, the critical section is accessed according to the increasing order of timestamps. (text, problem 6.5, part 1, p. 149)

Answer: Proof by contradiction. Suppose process p₁ issues a request to enter the critical section at time t₁, p₂ issues a similar request at time t₂ with t₁ < t₂, and p₂ enters first. This means that p₂ has received reply messages from all other processes including p₁. But p₁ will send such a message only if it is neither requesting nor executing the critical section (which is false) or if p₂’s request’s timestamp is smaller than that of p₁’s request (which is also false). Hence p₁ will not send a reply to p₂’s request, and so p₂ cannot enter the critical section first. This contradicts hypothesis, proving the claim.

3. (30 points) On p. 145, the text discusses the greedy strategy for Raymond’s tree-based algorithm, and notes that it can cause starvation. Please give an example of the application of this algorithm to a situation in which the greedy strategy causes starvation, but the regular algorithm does not.

Answer: There are two answers to this question, depending on how one views “site.”

If there are multiple processes at each site, the processes can generate a stream of requests to enter the critical section. As the greedy nature of the algorithm requires the site to honor requests generated at that site first, the token stays at the site and any other site with a request to enter the critical section starves.

If there is a single process at each site, starvation will not occur. Observe that, after the process finishes executing in the critical section, the token will be forwarded as indicated by the holder variable. Given this observation, the proof showing no starvation in both the greedy and non-greedy cases are the same.

Extra Credit

4. (30 points) Does Maekawa’s algorithm access the critical section according to the increasing order of timestamps? Either show that it does or provide a counterexample. (text, problem 6.5, part 2, p. 149)

Answer: The claim is false. Consider the following situation, with three sites:

\[ R_1 = \{ S_1, S_2 \} \]
\[ R_2 = \{ S_2, S_3 \} \]
\[ R_3 = \{ S_1, S_3 \} \]

These satisfy the conditions for Maekawa’s algorithm.

Let the clocks at sites 1, 2, and 3 be C₁ = 10, C₂ = 20, and C₃ = 30, respectively. Then:

- S₂ sends REQUEST(2, 20) to S₂ and S₃
- S₂ receives REQUEST(2, 20) from S₂
- S₂ sends REPLY(2, 21) to S₂
- S₂ receives REPLY(2, 21) from S₂
- S₃ sends REQUEST(3, 30) to S₁ and S₃
- S₃ receives REQUEST(3, 30) from S₃
- S₃ sends REPLY(3, 31) to S₃
$S_3$ receives $\text{REPLY}(3, 31)$ from $S_3$
$S_1$ receives $\text{REQUEST}(3, 30)$ from $S_3$
$S_1$ sends $\text{REPLY}(1, 31)$ to $S_3$
$S_3$ receives $\text{REPLY}(1, 31)$ from $S_1$

At this point, $S_3$ enters the critical section even though its request has a timestamp greater than that of $S_2$.

This works because Maekawa's algorithm sends a $\text{REPLY}$ to the first message that a process receives. If a later request comes with a lower timestamp, either a $\text{FAILED}$ message is sent or the $\text{REPLY}$ is held.