Answers to Homework #3

Due Date: February 27, 2001 **Points**: 70

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 p_1 issues a request to enter the critical section at time t_1, p_2 issues a similar request at time t_2 with $t_1 < t_2$, and p_2 enters first. This means that p_2's request is at the head of its queue. As the queues are ordered by timestamp, this means p_1's request has not arrived. If p_2 enters, though, it also received a message from p_1 with a timestamp higher than t_2. This implies that p_1's request has a timestamp higher than t_2 (which is false as $t_1 < t_2$) or p_2 never received p_1's request. The latter is possible only if either p_1's request was lost, or messages from p_1 to p_2 arrive out of order. Both these contradict the above basic assumptions. Hence p_2 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 p_1 issues a request to enter the critical section at time t_1, p_2 issues a similar request at time t_2 with $t_1 < t_2$, and p_2 enters first. This means that p_2 has received reply messages from all other processes including p_1. But p_1 will send such a message only if it is neither requesting nor executing the critical section (which is false) or if p_2's request's timestamp is smaller than that of p_1's request (which is also false). Hence p_1 will not send a reply to p_2's request, and so p_2 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 genetate 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 execuing in the critical section, the token will be forwarded as indicated by the *holdier* 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 $C_1 = 10$, $C_2 = 20$, and $C_3 = 30$, respectively. Then:
S_2 sends REQUEST(2, 20) to S_2 and S_3
S_2 receives REQUEST(2,20) from S_2
S_2 sends REPLY(2, 21) to S_2
S_2 receives REPLY(2, 21) from S_2
S_3 sends REQUEST(3, 30) to S_1 and S_3
S_3 receives REQUEST(3,30) from S_3
S_3 sends REPLY(3, 31) to S_3

S_3 receives REPLY(3, 31) from S_3
S_1 receives REQUEST(3, 30) from S_3
S_1 sends REPLY(1, 31) to S_3
S_3 receives 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 REPLY to the first message that a process receives. If a later request comes with a lower timestamp, either a FAILED message is sent or the REPLY is held.