Homework 3

CS425/ECE428 Spring 2022

Due: Monday, March 7 at 11:59 p.m.

Only up to 24 hours of grace period for late submissions can be used towards this homework. We will not accept any submissions after March 8, 11:59pm.

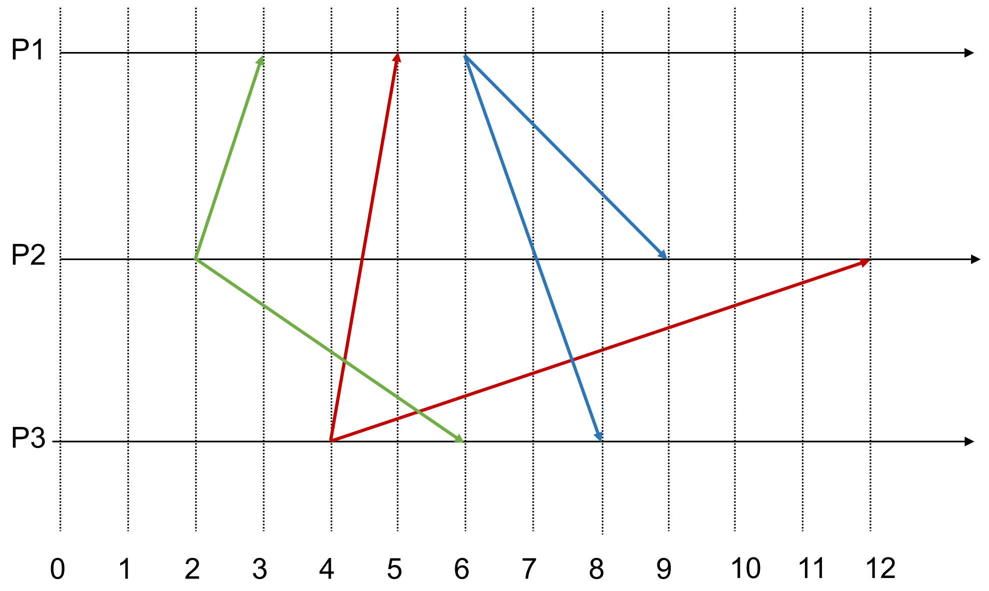

Figure 1: Figure for question 1(1)

1. **Mutual Exclusion – Ricart Agrawala Algorithm**
   Figure 1 shows three process P1, P2, and P3 (with ids 1, 2, and 3 respectively) implementing the Ricart-Agrawala (RA) algorithm for mutual exclusion. The lines indicate requests for accessing the critical section (CS) made by each process – blue, green, and red requests are from P1, P2, and P3 respectively. Other than the replies to CS requests (not shown in the figure), no other messages are exchanged between the processes. The timeline indicates real time. Assume that any reply sent for a CS request reaches the requesting process after exactly one (real) time unit. Further assume that any process that enters the CS, spends 3 (real) time units in it.
   (a) (2 points) What is P1's state as per the RA algorithm when it receives CS request from P2 – Held, Wanted, or neither (Free)? How will P1 handle P2's request upon receiving it – will it immediately send back a reply or will it queue the request? Why?
   (b) (2 points) What is P3's state as per the RA algorithm when it receives CS request from P2 – Held, Wanted, or neither (Free)? How will P3 handle P2's request upon receiving it – will it immediately send back a reply or will it queue the request? Why?
   (c) (2 points) What is P2's state as per the RA algorithm when it receives CS request from P1 – Held, Wanted, or neither (Free)? How will P2 handle P1's request upon receiving it – will it immediately send back a reply or will it queue the request? Why?
   (d) (2 points) What is P3's state as per the RA algorithm when it receives CS request from P1 – Held, Wanted, or neither (Free)? How will P3 handle P1's request upon receiving it – will it immediately send back a reply or will it queue the request? Why?
   (e) (2 points) What is P2's state as per the RA algorithm when it receives CS request from P3 – Held, Wanted, or neither (Free)? How will P2 handle P3's request upon receiving it – will it immediately send back a reply or will it queue the request? Why?

2. **Leader Election – Bully Algorithm**
Consider the following modification of the Bully algorithm: The initiating node (which we assume does not fail) sends an Election message only to the process with the highest id. If it does not get a response after a timeout, it then sends an Election message to the process with the second highest id. If after another timeout it gets no response, it tries the third highest id, and so on. If no higher numbered processes respond, it sends a Coordinator message to all lower-numbered processes.

   (a) (1 point) What should a process do when it receives an Election message in order to minimize turnaround time?

   For the following parts, consider a distributed system of 8 processes $\{P_1, P_2, \dots P_8\}$. P_8 has the highest id, followed by P_7, then P_6, and so on. The system uses the modified Bully algorithm for leader election (including the solution for 2a). Initially, all 8 processes are alive and P_8 is the leader. Then P_8 fails, P_3 detects this, and initiates the election. P_3 knows that P_8 has failed and P_7 has the highest id among the remaining processes. Assume one-way message transmission time is T, and timeout is set using the knowledge of T.

   (b) (1 point) If no other node fails during the election run, how many *total* messages will be sent by *all* processes in this election run?

   (c) (1 point) If no other node fails during the election run, how long will it take for the election to finish?

   (d) (1 point) Now assume that right after P_3 detects P_8's failure and initiates the election, P_7 fails. How many *total* messages will be sent by *all* processes in this election run?

   (e) (1 point) For the above scenario (where P_7 fails right after P_3 initiates election upon detecting P_8's failure), how long will it take for the election to finish?

3. **Ring-based leader election + consensus**
(5 points) Consider a system of N process that are arranged in a ring, with each process having a ring successor and a predecessor, and a communication channel only to its ring successor. Each process P_i has a unique id i. Further, each process P_i maintains a value x_i (which may not be unique across processes). A process may not know the total number of other processes in the ring.

   Each process P_i is required to set the value of an output variable y_i (initialized to *undecided*) to $\min_{j=1}^{N}(x_j)$. The safety condition for the problem requires that, at any point in time, the variable y_i at process P_i $\forall i \in [1, N]$ is either *undecided* or $\min_{j=1}^{N}(x_j)$.

   A distributed algorithm designed for the above problem works as follows:

   - A process P_i initiates the algorithm by sending $(\textit{propose}, x_i)$ to its ring successor.
   - When a process P_j receives $(\textit{propose}, x)$ from its ring predecessor:
     - if $x < x_j$, it forwards $(\textit{propose}, x)$ to its successor.
     - if $x > x_j$, it sends $(\textit{propose}, x_j)$ to its successor.
     - if $x = x_j$, it concludes that $x = x_j$ is the minimum value, and sends $(\textit{decided}, x)$ to its successor.
   - When a process P_j receives $(\textit{decided}, x)$, it sets $y_j = x$ and forwards $(\textit{decided}, x)$ to its successor (if it had not already done so in the past). Once P_j sets y_j, it ignores any subsequently received *decided* messages.

   Multiple processes may initiate the above algorithm simultaneously. Assume no process fails and the communication channel delivers all messages correctly and exactly once.

   Does the algorithm described above guarantee safety condition for the problem? If yes, prove how. If not, (i) describe a scenario where safety is violated, and (ii) suggest modifications to the algorithm that would guarantee the safety condition.

4. **Synchronous Consensus**

Consider a system of five processes $[P_1, P_2, P_3, P_4, P_5]$. Each process P_i proposes a value x_i. Let $x_1 = 6$, $x_2 = 5$, $x_3 = 8$, $x_4 = 2$, and $x_5 = 10$.

Each process P_k must decide on an output variable y_k (initialized to *undecided*), setting it to one of the proposed values x_i for $i \in [1, 5]$. The safety condition requires that at any point in time, for any two processes P_j and P_k, either y_j or y_k is *undecided*, or $y_j = y_k$ (in other words, the decided value must be same across all processes that have decided).

A consensus algorithm is designed for the above problem that works as follows:

- Each process R-multicasts its proposed value at the same time $t = 0ms$ since start of the system (as per their local clocks).
- As soon as proposed values from all 5 processes are delivered at a process P_j, P_j sets y_j to the minimum of the proposed values it received from the five processes.
- If y_j is still *undecided* at time $(t + timeout)$, P_j computes the minimum of the proposed values it has received so far and sets y_j to that value.
- Once a process P_j decides on y_j, it does not update y_j's value, and ignores future proposals (if any are received).

Assume that all clocks are perfectly synchronized with zero skew with respect to one-another. The proposed value x_i of a process P_i gets self-delivered immediately at time $t = 0ms$ when P_i begins the multicast of x_i. A message sent from a process to any other process takes exactly $T = 10ms$ (and this value is known to all processes). All communication channels are reliable. Processes may fail, but a failed process never restarts.

Suppose the *timeout* value for the above algorithm is set to $25ms$. Answer the following questions with respect to local time at the processes' clock since the start of the system.

(a) (2 points) Assume no process fails in the system. When will each process decide on a value and what will each of their decided values be?

(b) (2 points) Assume P_4 fails right after unicasting x_4 to P_3 and P_5 but just before it could initiate the unicast of x_4 to any of the other processes. When will each of the alive processes decide on a value and what will each of their decided values be?

(c) (2 points) Assume P_4 fails at right after unicasting x_4 to P_3 but just before it could initiate the unicast of x_4 to any of the other processes. P_3 fails right after it has relayed x_4 to P_5 but just before it unicasts it to any other process. When will each of the alive processes decide on a value and what will each of their decided values be?

(d) (2 points) Assume P_4 fails right after unicasting x_4 to P_3 but just before it could initiate the unicast of x_4 to any of the other processes. P_3 fails right after it has relayed x_4 to P_5 but just before it unicasts it to any other process. P_2 fails right before it could unicast x_2 to any process. When will each of the remaining alive processes decide on a value and what will each of their decided values be?

(e) (2 points) What is the smallest value that the *timeout* should be set to for ensuring safety in this system?

(f) (1 point) Answer Q4c assuming that the timeout is updated to the value in Q4e.

(g) (1 point) Answer Q4d assuming that the timeout is updated to the value in Q4e.

5. **Paxos**

Consider a system of five processes that implement the Paxos algorithm for consensus. As shown in Figure 2, P1 and P2 are proposers. P3, P4, P5 are acceptors. P1 sends a *prepare* message with proposal number 2 to processes P4 and P5, receives their replies, and sends an *accept* message with proposed value of 10 for proposal #2. P2 concurrently sends a *prepare* message with proposal #5, with an initial intention to propose a value of 15 if it receives sufficient replies. Only a subset of responses from processes P4, and P5 are shown in the figure. Assume no other proposals are initiated. Answer the following sub-questions.

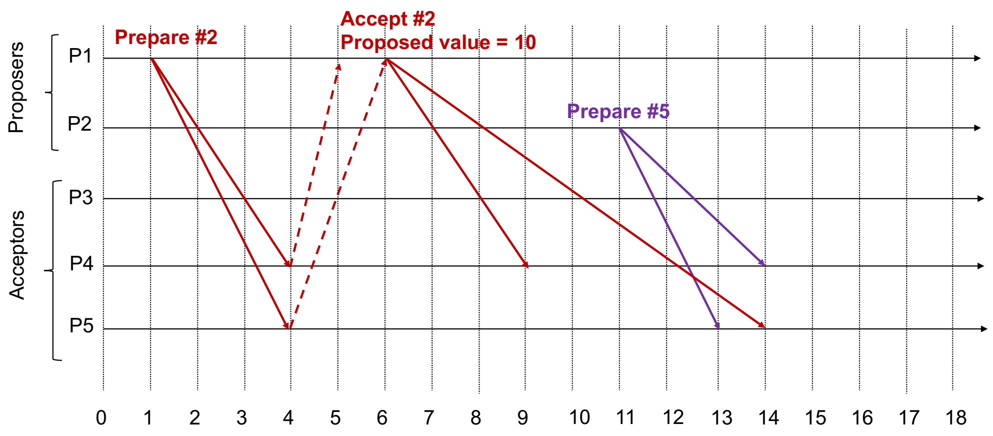

Figure 2: Figure for question 5(f)

(a) (1 point) Which processes will *accept* P1's proposal?

(b) (1 point) Which processes will reply back to P2's *prepare* message?

(c) (2 points) Will P2 send out an *accept* message for its proposal #5? If yes, what will be the corresponding proposed value?

(d) (2 points) Consider the state of the system at time 10 units. Has the proposed value 10 been implicitly decided upon at this point? If yes, explain why? If not, construct a possible scenario that may occur after time 10 units where the system ends up deciding on a different proposed value. The scenario would involve a temporal sequence of events that may differ from the one shown in the figure beyond time t=10units but must comply with what is shown until t=10units. An event in such a sequence may include a prepare/accept request sent by a proposer or received by an acceptor, or a prepare reply received by the proposer at some time unit.

(e) (1 point) Suppose that P1's *accept* request reaches P5 at time 8 units (instead of 14 units). If we now consider the state of the system at time 10 units, has the proposed value 10 been implicitly decided upon?

(f) (1 point) Suppose that P1's *accept* request reaches P4 at time 15 units (instead of 9 unit) and reaches P5 at the original time 14 units. Will P2 send out an *accept* message for its proposal #5? If yes, what will be the corresponding proposed value?