Homework 2

CS425/ECE428 Spring 2024

Due: Wednesday, Mar 6 at 11:59 p.m.

Process ID	Time when "enter" is called (since start of system)	Time spent in critical section after "enter" returns, before calling "exit"
P_1	10ms	50ms
P_2	200ms	30ms
P_3	25ms	10ms
P_4	5ms	20ms
P_5	105ms	5ms

Table 1: Timings for Q1

1. Consider a distributed system of five processes $\{P_1, P_2, P_3, P_4, P_5\}$. Each process needs mutually exclusive access to a critical section. Assume that each process wishes to enter a critical section only once. Table 1 lists the time when each process first makes a blocking call to "enter" the critical section (since the start of the system). It also lists the time each process spends in the critical section after "enter" succeeds, before calling "exit".

   For each of the subparts below, assume that the one-way network delay between any two different processes is fixed at 10ms (i.e. it takes exactly 10ms for a message to go from P_i to P_j, when $i \neq j$). The network delay for any message that a process P_i sends to itself is zero. Other than the network delays and the time spent in critical section, assume all other processing takes negligible amount of time.

   (a) (5 points) Suppose the system uses the central server algorithm for mutual exclusion, electing P_3 as the leader. The leader grants requests in the order in which it receives them. When will each process start executing its critical section?

   (b) (5 points) Now suppose that the system uses ring-based algorithm for mutual exclusion, with the ring structured as shown below (P1 to P2 to P3 to P4 to P5 to P1).

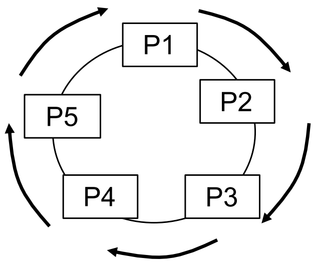
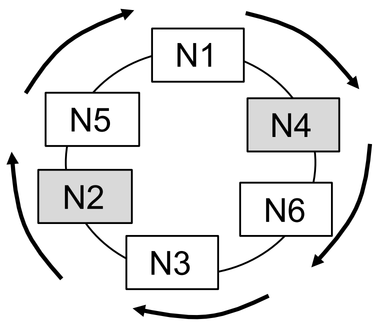


Figure 1

   At time 0ms (when the system starts up), the token is at P_1. As specified above, the network delay for passing the token from a given process to its ring successor is 10ms. When will each process start executing its critical section?

   (c) (5 points) Now suppose the processes use the Ricart-Agrawala algorithm for mutual exclusion. Assume all processes start off with a Lamport timestamp of zero, and no other events occur at the processes outside of the events that are part of the algorithm. When will each process start executing its critical section?

2. Consider a system of 6 processes $\{N1, N2, N3, N4, N5, N6\}$ with ids 1 to 6 (i for Ni). The system uses the ring-based algorithm for leader election, with the six processes arranged in a ring as shown below. The leader election algorithm uses the optimization proposed by Chang & Roberts to reduce the number of messages passed around the ring when multiple processes initiate the election. The process with highest id must be elected as the leader (which is N6 in this case). It is given that no process fails and no messages are dropped. The time taken to transmit a message from a process to its ring successor is exactly $20ms$, and the processing time at each process is negligible.



If processes N2 and N4 initiate the election simultaneously (at exactly the same time),

(a) (2 points) what is the total number of messages that get exchanged before N2 sets N6 as its elected leader?

(b) (2 points) how long does it take for N2 to set N6 as its elected leader after N2 and N4 initiate the election?

3. 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 7 processes $\{P_1, P_2, \ldots P_7\}$. P_7 has the highest id, followed by P_6, then P_5, and so on. The system uses the modified Bully algorithm for leader election (including the solution for 3a). Initially, all 7 processes are alive and P_7 is the leader. Then P_7 fails, P_3 detects this, and initiates the election. P_3 knows that P_7 has failed and P_6 has the highest id among the remaining processes. Assume one-way message transmission time between any two processes is fixed at 20ms, and timeout is set using the knowledge of this message transmission time.

(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_7's failure and initiates the election, P_6 fails. How many *total* messages will be sent by *all* processes in this election run?

(e) (1 point) For the above scenario (where P_6 fails right after P_3 initiates election upon detecting P_7's failure), how long will it take for the election to finish?

4. Consider a system of six processes $[P_1, P_2, P_3, P_4, P_5, P_6]$. Each process P_i proposes a value x_i. Let $x_1 = 10$, $x_2 = 8$, $x_3 = 5$, $x_4 = 12$, $x_5 = 15$, $x_6 = 6$.

   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 $i \in [1, 6]$. 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 6 processes are delivered at a process P_j, P_j sets y_j to the maximum of the proposed values it received from the six processes.
   - If y_j is still *undecided* at time $(t + timeout)$, P_j computes the maximum 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) (1 point) 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) (1 point) Assume P_5 fails right after unicasting x_5 to P_2 and P_6 but just before it could initiate the unicast of x_5 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_5 fails at right after unicasting x_5 to P_2 but just before it could initiate the unicast of x_5 to any of the other processes. P_2 fails right after issuing unicast of x_5 to P_6 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_5 fails at right after unicasting x_5 to P_2 but just before it could initiate the unicast of x_5 to any of the other processes. P_2 fails right after issuing unicast of x_5 to P_6 but just before it unicasts it to any other process. Then P_6 fails right after issuing unicast of x_5 to P_1 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?

   (e) (2 points) Assume P_4 fails right before it could unicast x_4 to any process. P_5 fails right after unicasting x_5 to P_2 but just before it could initiate the unicast of x_5 to any of the other processes. P_2 fails right after issuing unicast of x_5 to P_6 but just before it unicasts it to any other process. When will each of the remaining alive processes decide on a value and what will each of their decided values be?

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

   (g) (2 points) Answer Q4c assuming that the timeout is updated to the value in Q4f.

   (h) (2 points) Answer Q4d assuming that the timeout is updated to the value in Q4f.

   (i) (2 points) Answer Q4e assuming that the timeout is updated to the value in Q4f.