# Homework 3

CS425/ECE428 Spring 2021

**Due:** Thursday, March 18 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". |
|---|---|---|
| $P_3$ | 10ms | 20ms |
| $P_2$ | 15ms | 10ms |
| $P_1$ | 30ms | 15ms |
| $P_4$ | 45ms | 30ms |
| $P_5$ | 150ms | 25ms |

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. 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".

   (a) (5 points) Suppose the system uses the central server algorithm for mutual exclusion, electing $P_2$ as the leader. Assume that message latency at $P_2$ for communicating with the leader (itself) is zero, i.e. it takes negligible time for $P_2$'s token request to reach the leader upon calling enter, to receive the token after its request has been granted, and for the token to be released back to the leader upon calling exit. For all other processes, assume the one-way network latency for communicating with the leader ($P_2$) is fixed at 10ms, i.e. it takes 10ms each for the token request to reach P2 after calling "enter", 10ms for the token to reach the process after the leader has granted the request, and 10ms for the token to reach the leader after the process has called "exit". 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).

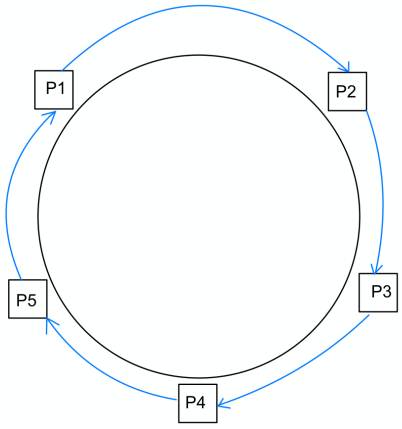


Figure 1

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

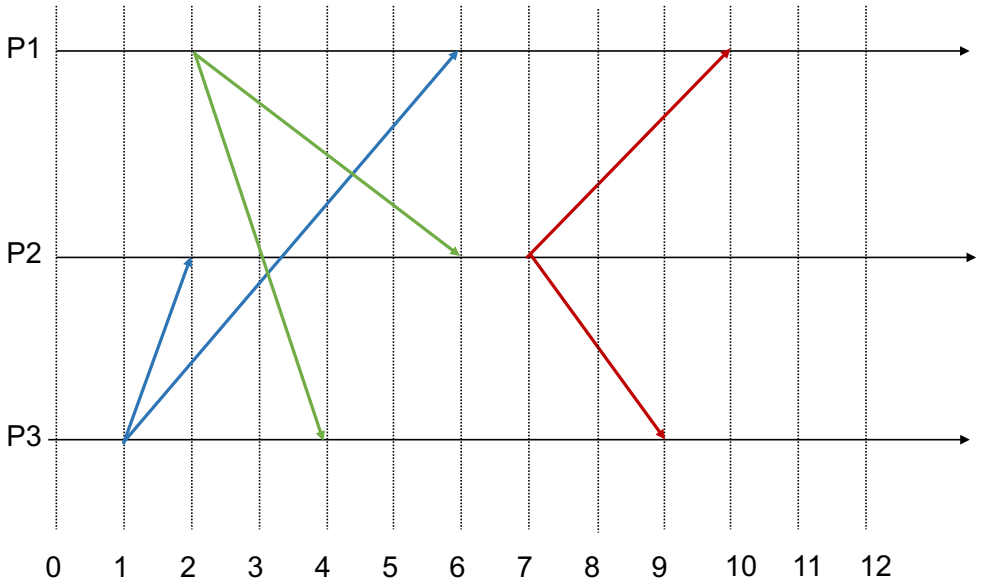


Figure 2: Figure for question 2(2)

2. Figure 2 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 – green, red, and blue 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 5 (real) time units in it.

   (a) (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?

   (b) (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?

   (c) (2 points) What is P1's state as per the RA algorithm when it receives CS request from P3 – Held, Wanted, or neither (Free)? How will P1 handle P3'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 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?

   (e) (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?

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) (2 points) 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 9 processes $\{P_1, P_2, \ldots P_9\}$. $P_9$ has the highest id, followed by $P_8$, then $P_7$, and so on. The system uses the modified Bully algorithm for leader election (including the solution for 3a). Initially, all 9 processes are alive and $P_9$ is the leader. Then $P_9$ fails, $P_5$ detects this, and initiates the election. $P_5$ knows that $P_9$ has failed and $P_8$ 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) (2 points) If no other node fails during the election run, how many *total* messages will be sent by *all* processes in this election run?

(c) (2 points) If no other node fails during the election run, how long will it take for the election to finish?

(d) (2 points) Now assume that right after $P_5$ detects $P_9$'s failure and initiates the election, $P_8$ fails. How many *total* messages will be sent by *all* processes in this election run?

(e) (2 points) For the above scenario (where $P_8$ fails right after $P_5$ initiates election upon detecting $P_9$'s failure), how long will it take for the election to finish?

4. 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 $i$. A process $P_i$ maintains a value $x_i$ (these values may not be unique across processes).

(a) (6 points) Consider a problem where each process $P_k$ is required to set the value of an output variable $y_k$ (initialized to *undecided*) to $\max_{i=1}^{N}(x_i)$. The safety condition for the problem requires that, at any point in time, the variable $y_k$ at process $P_k$ $\forall k \in [1, N]$ is either *undecided* or $\max_{i=1}^{N}(x_i)$.

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

- A process $P_i$ initiates the algorithm by sending $(propose, x_i)$ to its ring successor.
- When a process $P_j$ receives $(propose, x)$ from its ring predecessor:
  - if $x > x_j$, it forwards $(propose, x)$ to its successor.
  - if $x < x_j$, it sends $(propose, x_j)$ to its successor.
  - if $x = x_j$, it concludes that $x = x_j$ is the maximum value, and sends $(decided, x)$ to its successor.
- When a process $P_j$ receives $(decided, x)$, it sets $y_j = x$ and forwards $(decided, x)$ 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.

(b) (4 points) Now consider a modified problem. The value of $x_i$ at a process $P_i$ is either 0 or 1. Each process $P_k$ must decide on a value $y_k$ which is the majority value of $x_i$ across all processes $P_i$ (for $i \in [1, N]$). Assume for simplicity that the number of processes ($N$) is odd. Design a ring-based algorithm for this problem, which follows the constraint that processes are arranged in a ring, with each process having a ring successor and a predecessor, and a communication channel only to its ring successor. You may assume that no process fails and all messages are delivered correctly and exactly once. Multiple processes may initiate your algorithm simultaneously.