Quiz 1 (75 Minutes) Marks 50

Name:

ID:

General Instructions:

- ***This quiz is open book, open notes and you can use your electronic devices to refer to lecture notes, slides and likewise. However, usage of ChatGPT and other LLMs are strictly prohibited. If discovered violating this rule, the candidate will be disqualified from the quiz following SUTD regulations.***
- *If you are answering some of the questions (e.g., diagrams) by pen/pencil and paper, and not via the soft copy, then please put a note "**see hard copy**" at corresponding places in the soft copy.*
- *For each white paper used in the hard copy, write your name and student ID.*

1. Consider the scenario in invitation protocol where the network with 2*N nodes is partitioned exactly into two halves resulting in an election in each partition. Assuming that each partition uses Bully algorithm for coordinator election, compute (approximately) the maximum number of messages exchanged until the partitions are merged and a single coordinator is selected. You may assume that no more partitions or node failure occur during the entire election and merge process. Do not just provide the final answer, show the derivation of how you arrive at the answer.
(6 marks)

Once the network is partitioned, each partition will first use Bully algorithm to select their own coordinator.

Secondly, coordinator of each partition will periodically look for other nodes to merge to. We assume this is performed once before the partitions were merged. The extension if such was done multiple times, can be similarly multiplied with a factor.

Thirdly, once the low-ID coordinator is aware of the higher-ID coordinator (previously, in another partition), it will broadcast the new co-ordinator ID to all in its partition.

The first stage will take approximately O(N^2) messages in the worst case.
The second stage will take approximately O(2*N) messages.
The third stage will involve another O(N) messages.

2. Discuss one disadvantage of Berkley's clock synchronization protocol as compared to Cristian's protocol. Illustrate the shortcoming with an example. You may assume that the network is ideal in the sense that sending and receiving messages take equal amount of time. No marks will be awarded in the absence of the example.
(4 marks)

One of the disadvantage of Berkley in comparison to Cristian is that the Berkley protocol may never synchronize with the true time, as the protocol is merely to synchronize with the average of the population. There is no special consideration for synchronizing with the true time (e.g., UTC).

3. (a) Consider the basic version of the voting protocol (i.e., without deadlock prevention) with five (5) nodes. Assume two out of these five nodes, say N1 and N2, wish to enter the critical section concurrently. N1 receives three votes and N2 receives two votes. Construct a scenario *where N1 receives vote(s) after exiting the critical section.* To this end, you should draw a sequence diagram to show the order of messages. To simplify the sequence diagram, you may skip the *request* messages sent to enter the critical section.

What is important to show:

→ Exchange of messages.
→ **The status of the queue (need to show state variables if they are important to answer).**
→ **Entry and exit of the critical section (need to show in this question).**

The sequence of operations:

1. N1 enters the critical section by taking votes from N2, N4 and N3. Additionally, N3 and N4 queues N2.
2. N2 waits to enter the critical section by taking votes from N1 and N5.
3. The queue of N5 has the request of N1 in the queue although N1 can enter the critical section.
4. N1 finishes the critical section.
5. N1 releases votes of N2, N3 and N4.
6. N2 can receive vote from N3 and N4. Enters the critical section.
7. N2 finishes the critical section. Send release vote to N5.
7. N5 has N1 in the queue, vote for N1. But N1 has already exited critical section.

(b) For the same system above with five nodes, is it possible for N3/N4/N5 to have two requests of N1 in the queue? If the aforementioned situation is possible, then construct an example (e.g., a sequence diagram). If the situation is not possible, then provide a proof. Note that the queue is maintained in each node for outstanding requests that have not been voted. Also note that any node can only make one request (to enter critical section) *at a time (i.e., it does not make another request before releasing the lock of the current request)*. No marks will be awarded in the absence of the example or a proof.

(5 marks + 5 marks)

In the previous sequence, do everything until point 6. Then, N1 makes another request and this request is sent to N5. The request from N1 reaches N5 before N2 finishes critical section and send the release vote to N5. Thus, N5 will end up having two requests from N1 in its queue: one already finished executing CS and another about to enter CS with sufficient votes.

4. a) Consider Lamport's shared priority queue protocol for distributed mutual exclusion with Ricart and Agrawala's optimization. Prove that the protocol satisfies *fairness (in terms of logical clock). In your proof, carefully consider all cases.*

We prove this by contradiction.

Consider two arbitrary requests with timestamp T (from Machine P1) and T+X (from Machine P2). We note that even though logical timestamps might be the same for two concurrent requests, there is always a possibility to impose a total order.

Therefore, for the contradiction to hold, P2 should execute the critical section before P1 executes the critical section.

Case 1 (Request from P2 is received **after** P1 makes request): In this case, P1 will only respond to P2 when P1 finishes critical section, since T < T+X. Hence, it is not possible for P2 to receive all responses before P1 finishes critical section, leading to a contradiction of our initial assumption.

Case 2 (Request from P2 is received **before** P1 makes request): In this case, the request of P1 cannot have a timestamp "T" by definition of logical clock. This is because, once the request from P2 is received at P1, the logical clock must have advanced to at least T+X+1. Hence, any request at P1 will have a timestamp >= T+X+1.

The contradiction shows that the fairness violation is not possible. Hence, the Lamport's protocol with the optimization remains fair.

b) By design, ring-based protocol for mutual exclusion is not fault tolerant. Design a simple protocol to make the ring-based protocol fault tolerant. Specifically, discuss the following: 1) messages sent and received and the protocol to handle them, 2) how to handle nodes leaving the network, 3) how to handle nodes joining the network, 4) how to handle intermittent faults. Do not provide generic answer like "I will use Dynamo", this will not award any marks.

The key idea here is to integrate functionalities similar to Ring-based Election together with the Ring-based DME protocol.

Assumptions: The node failure can be detected reliably via a timeout. This is not perfect, but we need to start somewhere.

Key change: When the token is passed to the successor in DME, wait for a timeout until an acknowledgement is received from the successor. If the acknowledgement is not received within the timeout, then send to the successor of successor a message on upgrading or downgrading the logical ring structure. Thus, every node in the ring has the full logical structure of the ring.

Handling intermittent faults: If a fault is detected by timeout, then compute and send the new logical structure to the successor of successor. This continues until a live node is found. Once a node receives a new ring structure, it will work based on the new ring structure.

Joining the ring: Send any message to a node in the current ring. This will trigger a ring upgrade process. The same upgrade will also be triggered when an outdated token is received from a node which is not in the ring.

Leaving the ring: Send an downgrade message to the successor to downgrade the new logical ring. Technically the upgrade and downgrade are the same messages, however, the former adds a node whereas the latter removes a node from the ring.

c) For your protocol in (b), prove/discuss whether it satisfies *safety and fairness*.

Based on the assumptions, the protocol is safe, as the token will be held by only one node at a time. If a node does not respond to the token forward message, then based on the assumption, it will not be executing the critical section. If the node failed after getting the token and perhaps within CS, then we could roll back to the state previous to the critical section. If it failed after executing the CS, the protocol is still safe due to the mutually exclusive nature of critical section time execution windows.

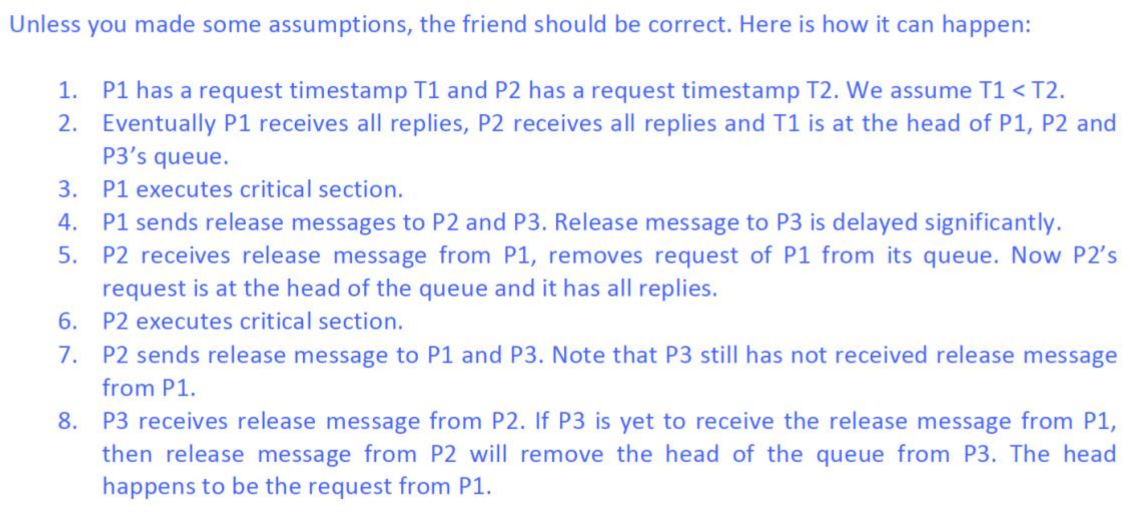
Fairness is not preserved for the above protocol, as we do not timestamp the requests. However, a similar extension for fair ring protocol is possible.

(4 marks + 10 marks + 6 marks)

5. Consider Lamport's shared priority queue protocol for distributed mututal exclusion without Ricart and Agrawala's optimization. You have three machines P1, P2 and P3, none of which is malicious. Let us assume P1 and P2 request to enter the critical section concurrently. Is it possible to encounter a scenario where the following happens: *Upon receiving the RELEASE message from P2, machine P3 removes the request of P1 from P3's priority queue*. If the scenario is impossible, then provide a proof. If the scenario is possible, then draw a sequence diagram or discuss the scenario point-by-point as it happens. You may assume that the network layer does not drop any message.

Discuss whether such a behaviour affects the safety and fairness properties of the protocol.

(6 marks + 4 marks)



Unless you made some assumptions, the friend should be correct. Here is how it can happen:

1. P1 has a request timestamp T1 and P2 has a request timestamp T2. We assume T1 < T2.
2. Eventually P1 receives all replies, P2 receives all replies and T1 is at the head of P1, P2 and P3's queue.
3. P1 executes critical section.
4. P1 sends release messages to P2 and P3. Release message to P3 is delayed significantly.
5. P2 receives release message from P1, removes request of P1 from its queue. Now P2's request is at the head of the queue and it has all replies.
6. P2 executes critical section.
7. P2 sends release message to P1 and P3. Note that P3 still has not received release message from P1.
8. P3 receives release message from P2. If P3 is yet to receive the release message from P1, then release message from P2 will remove the head of the queue from P3. The head happens to be the request from P1.

Despite the aforementioned behaviour, it does not affect the safety and fairness of Lamport's protocol. This is because the behaviour happens after the requests have been processed and the critical sections were executed. Hence, the behaviour does not affect safety and fairness as these properties are related to when certain processes and nodes enter critical sections.