CS 247: Principles of Distributed Computing
Time: 70 mins

Name and ID: ______________________________

Instructor's name: ______________________________

1. (5 points) A replicated state machine is a form of a distributed system in which all the processes of the system replicate a state such as an integer or a set. It is important for such systems to eventually reach the same final state. This property is called convergence. Is convergence a safety or liveness property?

   Answer:
   Liveness

2. (15 points) Explain whether the following statements are True or False.

   (a) In a synchronous model, no request ever takes more than 1 week to be processed.

   Answer:
   True or False both answers could be correct based on the assumption
   Give the score to everyone.

   (b) In a fail-stop model, every process that crashes is eventually detected.

   Answer:
   True
   The perfect failure detector is complete and provides this property.

   (c) In a fail-stop model, every correct process delivers a message before t time units.

   Answer:
   True
   In a fail-stop model, computations are synchronous and their processing times are bound. t can be picked such that is greater than this bound.

3. (20 points) Can we implement a perfect failure detector in a model where processes can drop messages at most k times? If yes, What is the maximum timeout in a such model to detect failures? If no, explain why and give a counterexample. (assume the transmission delay between two processes is at most t)

   Answer:
   Yes. If the number of possible omissions is known in a synchronous system, we can use it to calibrate the timeout delay of the processes to accurately detect failures. The maximum timeout would not exceed $2tk$.
   Students might have considered t to be round trip time, therefore alternative answer would be αtk where α is some constant.

4. (20 points) Consider the uniform reliable broadcast algorithm implemented with a perfect failure detector. What happens if the perfect failure detector is replaced with an eventual perfect failure detector? Explain and draw a diagram.

```
Implements:
    UniformReliableBroadcast, instance urb.

Uses:
    BestEffortBroadcast, instance beb.
    PerfectFailureDetector, instance $\mathcal{P}$.

upon event ⟨ urb, Init ⟩ do
    delivered := $\emptyset$;
    pending := $\emptyset$;
    correct := $\Pi$;
    forall $m$ do ack[$m$] := $\emptyset$;

upon event ⟨ urb, Broadcast | $m$ ⟩ do
    pending := pending $\cup$ {(self, $m$)};
    trigger ⟨ beb, Broadcast | [DATA, self, $m$] ⟩;

upon event ⟨ beb, Deliver | $p$, [DATA, $s$, $m$] ⟩ do
    ack[$m$] := ack[$m$] $\cup$ {$p$};
    if $(s, m) \notin$ pending then
        pending := pending $\cup$ {($s$, $m$)};
        trigger ⟨ beb, Broadcast | [DATA, $s$, $m$] ⟩;

upon event ⟨ $\mathcal{P}$, Crash | $p$ ⟩ do
    correct := correct \ {$p$};

function candeliver($m$) returns Boolean is
    return (correct $\subseteq$ ack[$m$]);

upon exists $(s, m) \in$ pending such that candeliver($m$) $\wedge$ $m \notin$ delivered do
    delivered := delivered $\cup$ {$m$};
    trigger ⟨ urb, Deliver | $s$, $m$ ⟩;
```

Answer:

Uniform agreement is violated.

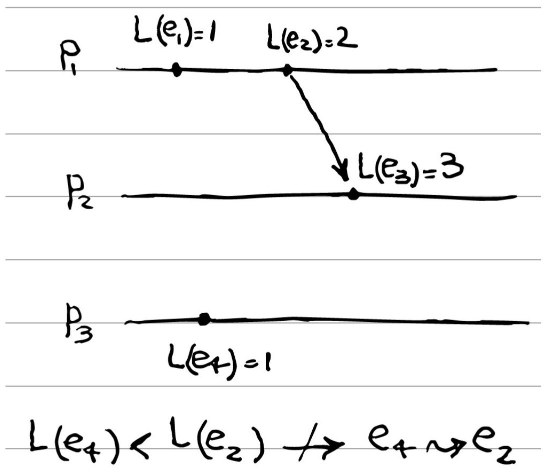
Assume a system of three processes p1, p2 and p3. P1 starts broadcast m. and meanwhile, falsely suspects p2 and p3 to have crashed. Process p1 delivers m and crashes afterward. Here p2 and p3 have never delivered and have no way of knowing about m because p1 has crashed.

5. (20 points) The Lamport clock algorithm is a simple logical clock algorithm used for ordering the events in a distributed system. Since different processes are not perfectly synchronized, this algorithm is used to provide a partial ordering of events with minimal overhead. The Lamport clock is implemented using an integer and has the following simple rules:

   1. A process increments its own logical clock before each event (e.g., before sending an update message);
   2. When a process sends a message, it includes its logical clock value with the message;
   3. On receiving a message, the recipient's logical clock is updated to the larger of its own logical clock or the one received by the incoming message.
      In a Lamport clock, L, it is guaranteed that if $e_1 \rightsquigarrow e_2$, then $L(e_1) < L(e_2)$. where $\rightsquigarrow$ indicates the *causal order* relation between two events and $L(e)$ denotes the value of the logical clock of event e at whatever process it occurred at.

   (a) (10 points) There is one drawback to Lamport clock, the causal order is not guaranteed even if $L(e_1) < L(e_2)$. Give an example using 3 processes that shows this, i.e, $L(e_1) < L(e_2) \not\Rightarrow e_1 \rightsquigarrow e_2$. Clearly mark which 2 events demonstrate it.

   Answer:
   Or similar figures that show this:

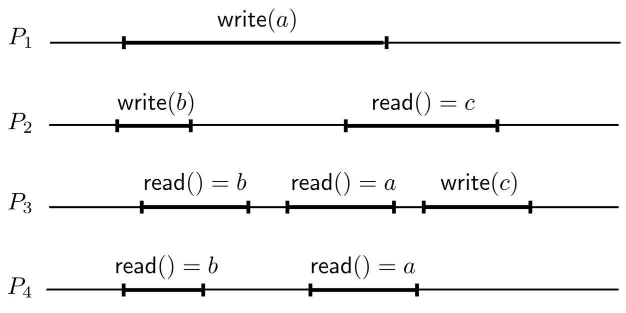


   The two events e_2 and e_4 clearly don't have causal order relation. But their clock value indicate otherwise.

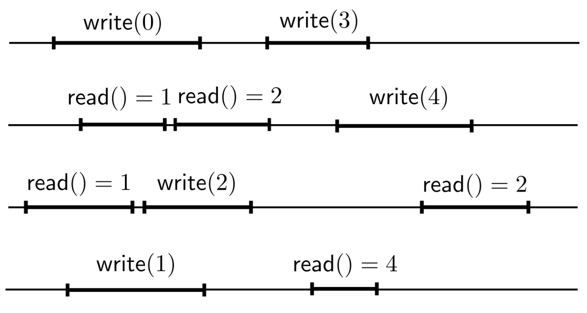
   (b) (10 points) We studied the vector clocks in the class for the causal order broadcast abstraction. Do vector clocks suffer from the mentioned drawback? Clearly explain your answer and revisit your previous example if necessary.

   Answer:
   No, they resolve the mentioned problem by defining a logical clock (integer) per process which construct the vector clock. Two vector clocks are comparable only if *all* the entries in one vector are greater equal, or *all* the entries are less equal than the other vector elements. Intuitively, in the above example the two vector clocks are incomparable.

6. (20 points) The following two diagrams show the execution of a register. Read and write events are clearly marked with the span of the operation as well as the return value for the reads. Which diagram shows the execution of an *atomic* register? Mark the linearization points. If not, explain why.

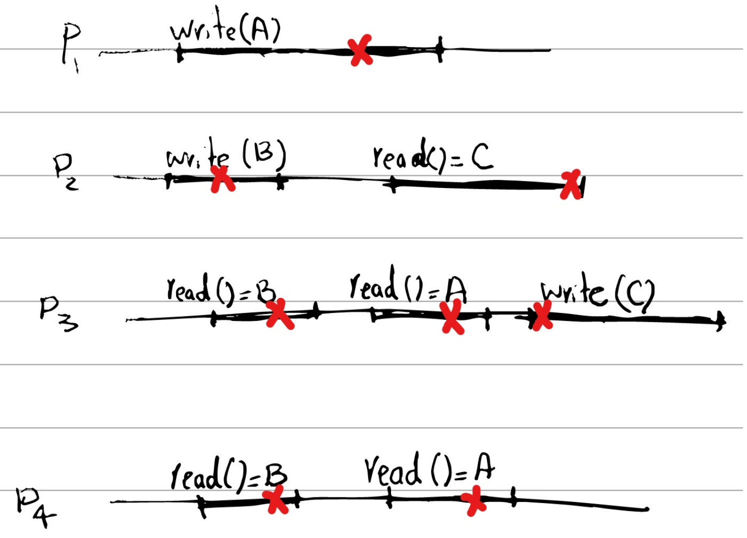


(a)



(b)

Answer:



(a) Is showing an atomic register. The linearization points are marked in red.
(b) Is not an atomic register since there is no linearization point for write(4) and read:4. The last read at the second process returns 2 which makes it impossible to linearize write(4) and read:4.