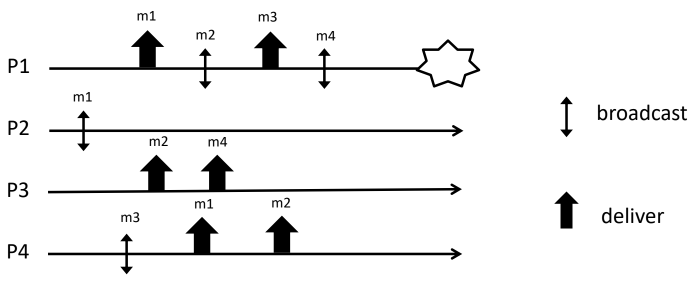
Dependable Distributed Systems
Master of Science in Engineering in Computer Science

AA 2022/2023

Lecture 18 – Exercises
November 9th, 2022

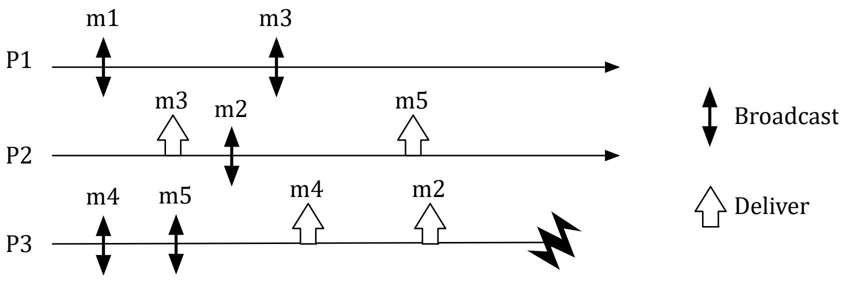
Ex 1: Consider the execution depicted in the Figure



Answer to the following questions:

1. Provide all the delivery sequences that satisfy both causal order and total order
2. Complete the execution in order to have a run satisfying TO(UA, WNUTO), FIFO order but not causal order

Ex 2: Consider the partial execution shown in the Figure and answer to the following questions:



1. Complete the execution in order to have a run satisfying the Regular Reliable Broadcast specification but not Uniform Reliable Broadcast one.
2. For each process, provide ALL the delivery sequences satisfying FIFO Reliable Broadcast but not satisfying causal order.
3. For each process, provide ALL the delivery sequences satisfying total order and causal order.

Ex 3: Consider a distributed system composed of N processes p_1, p_2, … p_N, each having a unique identifier myID. Initially, all processes are correct (i.e. correct={ p_1, p_2, … p_N }). Consider the following algorithm:

```
upon event Xbroadcast(m)
        mysn = mysn+1;
        ∀ p ∈ correct
                pp2pSend ("MSG", m, mysn, myId);

upon event pp2pReceive ("MSG", m, sn, i)
      mysn= mysn+1;
      if (m ∉ delivered)
         trigger XDeliver (m);
         delivered = delivered ∪ {m};

upon event crash (p_i)
        correct = correct / {p_i}
```

Let us assume that: (i) links are perfect, (ii) the failure detector is perfect and (iii) initially local variables are initialized as follows mysn=0 e delivered =∅.
Answer to the following questions:

1. Does the `Xbroadcast()` primitive implement a Reliable Broadcast, a Best Effort Broadcast or none of the two?
2. Considering only the ordering property of broadcast communication primitives discussed during the lectures (FIFO, Causal, Total), explain which ones can be satisfied by the `Xbroadcast()` implementation.

Provide examples to justify your answers.

Ex 4: Consider a distributed system constituted by *n* processes ∏={p_1, p_2… p_n} with unique identifiers that exchange messages through FIFO perfect point-to-point links and are structured through a line (i.e., each process p_i can exchange messages only with processes p_{i-1} and p_{i+1} when they exists). Processes may crash and each process is equipped with a perfect oracle (having the interface *new_right(p)* and *new_left(p)*) reporting a new neighbor when the previous one is failing.
Write the pseudo-code of an algorithm implementing a Perfect failure detector primitive.

Ex 5: Consider a distributed system composed of n processes p1, p2,… pn connected through a ring topology. Initially, each process knows the list of correct processes and maintains locally a `next` variable where it stores the id of the following process in the ring.
Each process can communicate only with its next trough FIFO perfect point-to-point channels (i.e. the process whose id is stored in the `next` variable).
Processes may fail by crash and each process has access to a perfect failure detector.

Write the pseudo-code of a distributed algorithm implementing a (1, N) atomic register.

Ex 6: A transient failure is a failure that affects a process temporarily and that alter randomically the state of the process (i.e., when the process is affected by a transient failure, its local variables assume a random value).
Let us consider a distributed system composed by N processes where f_c processes can fail by crash and f_t processes can suffer transient failures between time t_0 and t_{stab}.

Let us consider the following algorithm implementing the Regular Reliable Broadcast specification

Algorithm 3.2: Lazy Reliable Broadcast

```
Implements:
    ReliableBroadcast, instance rb.

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

upon event ⟨ rb, Init ⟩ do
    correct := $\Pi$;
    from[p] := $[\emptyset]^N$;

upon event ⟨ rb, Broadcast | m ⟩ do
    trigger ⟨ beb, Broadcast | [DATA, self, m] ⟩;

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

upon event ⟨ $\mathcal{P}$, Crash | p ⟩ do
    correct := correct $\setminus \{p\}$;
    forall $m \in$ from[p] do
        trigger ⟨ beb, Broadcast | [DATA, p, m] ⟩;
```

Answer to the following questions:

1. For every property of the Regular Reliable Broadcast specification, discuss if it guaranteed between time t_0 and t_{stab} and provide a motivation for your answer.
2. For every property of the Regular Reliable Broadcast specification, discuss if it eventually guaranteed after t_{stab} and provide a motivation for your answer.
3. Assuming that the system is synchronous, explain how you can modify the algorithm (no pseudo-code required) to guarantee that No Duplication, Validity and Agreement properties will be eventually guaranteed after t_{stab}.

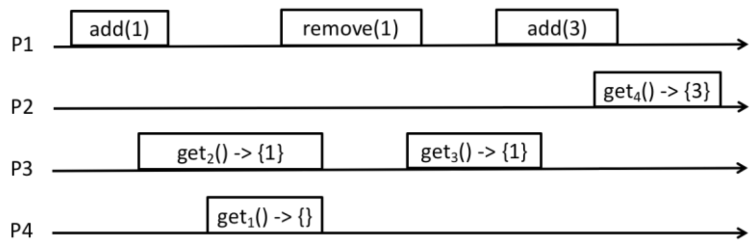
Ex 7: A service is delivered through 3 components that sequentially handles the requests and it receives around 75 requests per second. The components, A, B and C, serve respectively 100, 80 and 90 requests per second. The clients are complaining about the response time of the system. There is the possibility either 1) to double one component employing a perfect load balancer or 2) to substitute one component with an improved one serving 40% more requests per second.
Which component upgrade and option reduce the response time the most? Compute the expected response time.

Ex 8: Consider a set object that can be accessed by a set of processes. Processes may invoke the following operations on the object:

- add(v): it adds the value v in to the set
- remove(v) it removes the value v from the set
- get(): it returns the content of the set.

Informally, every get() operation returns all the values that have been added before its invocation and that have not been removed by any remove().
For the sake of simplicity, assume that a value can be added/removed just once in the execution.

Consider the distributed execution depicted in the Figure



Answer to the following questions:

1. Is the proposed execution linearizable? Motivate your answer with examples.
2. Consider now the following property: "every get() operation returns all the values that have been added before its invocation and that have not been removed by any remove(). If an add(v)/remove(v) operation is concurrent with the get, the value v may or may be not returned by the get()".
   Provide an execution that satisfy get validity and that is not linerizable.