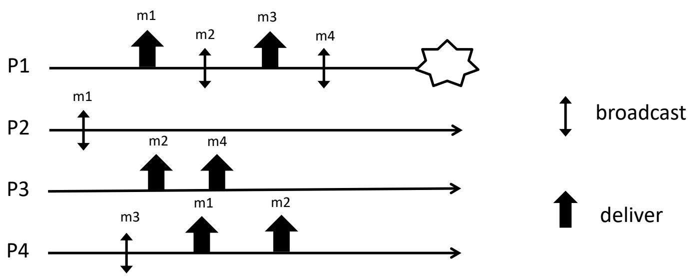
Distributed Systems
Master of Science in Engineering in Computer Science

AA 2020/2021

Lecture 18 – Exercises
December 7^{th}, 2020

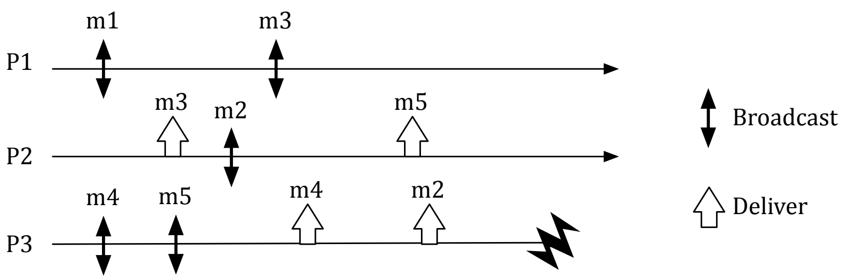
Ex 1: Consider the execution depicted in the Figure



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:



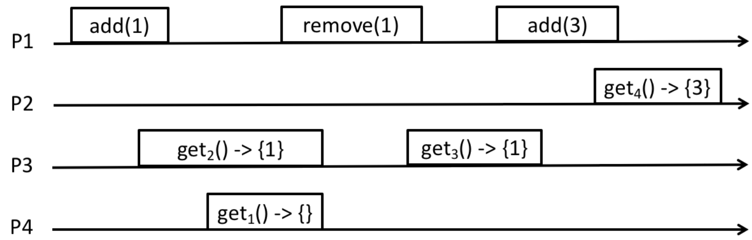
1. Complete the execution in order to have a run satisfying the Regular Reliable Broadcast specification but not Uniform Reliable Broadcast one.
2. For each process, provide <u>ALL</u> the delivery sequences satisfying FIFO Reliable Broadcast but not satisfying causal order.
3. For each process, provide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

Ex 3: 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()".
3. Provide an execution that satisfy get validity and that is not linerizable.

Ex 4: Consider a distributed system composed of N processes $p_1, p_2, \dots p_N$, each having a unique identifier myID. Initially, all processes are correct (i.e. correct={ $p_1, p_2, \dots p_N$ }). Consider the following algorithm:

upon event `Xbroadcast(m)`
    mysn = mysn+1;
    $\forall$ p $\in$ correct
        `pp2pSend` ("MSG", m, mysn, myId);

upon event `pp2pReceive ("MSG", m, sn, i)`
    mysn= mysn+1;
    if (m $\notin$ delivered)
        trigger `XDeliver (m)`;
        delivered = delivered $\cup$ {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 5: Consider a distributed system constituted by *n* processes $\prod=\{p_1, p_2\ldots p_n\}$ 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 6: 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 7: Consider a distributed system constituted by *N* client processes $C=\{c_1, c_2\ldots c_N\}$ and *M* server processes $S=\{s_1, s_2\ldots S_M\}$ with unique identifiers. Both clients and servers may exchange messages through perfect point-to-point links and each process (both clients and servers) may potentially send a message to any other.
The system is assumed to be synchronous: clocks are perfectly synchronized, the computation time is negligible and the latency of communication channels is upper bounded by a constant δ known by every processes.
Clients may experience crash failure while up to f servers (with $f < M/3$) may experience Byzantine failures.
Servers have access to a Byzantine tolerant consensus primitive.
Write the pseudo-code of an algorithm implementing the replication of a shared object X by following the active replication schema.