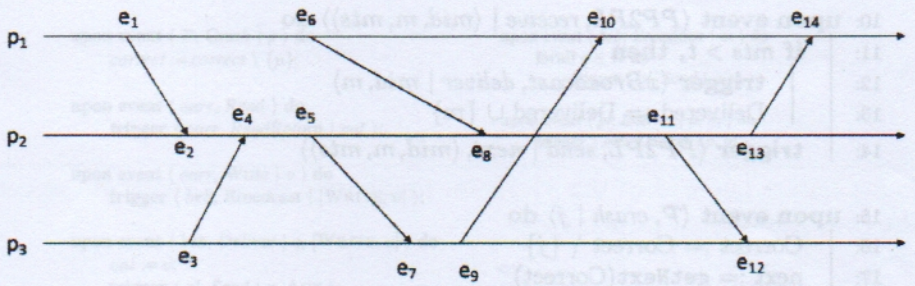
A

Dependable Distributed Systems (9 CFU)
20/02/2024

Family Name_____Name_____Student ID_____

Ex1: Describe what a publish-subscribe communication system is highlighting its main characteristics. In addition compare the characteristics of the one-to-many communication obtained when leveraging a publish-subscribe system and a Reliable Broadcast-based communication system.

Ex2: Let us consider the following execution history



Let us denote with ck(ei) the logical clock associated with event ei. Considering the execution history shown in the figure above, assess the truthfulness of every statement and provide a motivation for your answer:

1	If we use scalar clocks for timestamping events, then ck (e6) = ck (e5)	T	F
2	e2 happened before e6 (according with Lamport's definition of happened-before)	T	F
3	If we use scalar clocks for timestamping events, then ck (e7) = ck (e3) +1	T	F
4	e7 and e8 are concurrent events	T	F
5	If we use scalar clocks for timestamping events, then ck (e13) > ck (e12)	T	F
6	If we use vector clocks for timestamping events, then ck (e13) = [4, 6, 0]	T	F
7	If we use vector clocks for timestamping events, then ck(e1) = ck (e3)	T	F
8	If we use vector clocks for timestamping events, then ck(e5) and ck (e6) are not comparable	T	F
9	If we use vector clocks for timestamping events, then ck(e8) < ck (e14)	T	F
10	If we use vector clocks for timestamping events, then ck (e4) = [0, 1, 1]	T	F

Ex3: Consider a distributed system composed of n processes, each with a unique integer sequential identifier. Processes can communicate by exchanging messages over perfect point-to-point links. Processes are equipped with a perfect failure detector P.
The xBroadcast primitive is deployed in such a system managing two events: < *xBroadcast, send | m* > and < *xBroadcast, deliver | ps, m* >, and is implemented via the following algorithm.

Algorithm 10 xBroadcast

```
1:  procedure INIT
2:      Correct := Π
3:      Delivered := ∅
4:      t_s := 0
5:      // id : process's identifier
6:      Next := (id + 1) mod n

7:  upon event ⟨xBroadcast, send | m⟩ do
8:      t_s := t_s + 1
9:      trigger ⟨PP2PL, send | next, ⟨id, m, t_s⟩⟩

10: upon event ⟨PP2PL, receive | ⟨mid, m, mts⟩⟩ do
11:     if mts > t_s then
12:         trigger ⟨xBroadcast, deliver | mid, m⟩
13:         Delivered := Delivered ∪ {m}
14:     trigger ⟨PP2PL, send | next, ⟨mid, m, mts⟩⟩

15: upon event ⟨P, crash | j⟩ do
16:     Correct := Correct \ {j}
17:     next := getNext(Correct)
```

Assess the truthfulness of every statement and provide a motivation for your answer:

1	Assuming that NO CRASH occurs, the xBroadcast primitive guarantees that if a correct process p broadcasts a message m, then p eventually delivers m	T	F
2	Assuming that NO CRASH occurs, the xBroadcast primitive guarantees that no message is delivered more than once	T	F
3	Assuming that NO CRASH occurs, the xBroadcast primitive guarantees that if a process delivers a message m with sender s, then m was previously broadcast by process s.	T	F
4	Assuming that NO CRASH occurs, the xBroadcast primitive guarantees that if a message m is delivered by some correct process, then m is eventually delivered by every correct process.	T	F
5	Assuming that NO CRASH occurs, the xBroadcast primitive guarantees that if some process broadcasts message m1 before it broadcasts message m2, then no correct process delivers m2 unless it has already delivered m1	T	F
6	Assuming that CRASH FAULTS MAY occur, the xBroadcast primitive guarantees that if a correct process p broadcasts a message m, then p eventually delivers m	T	F
7	Assuming that CRASH FAULTS MAY occur, the xBroadcast primitive guarantees that no message is delivered more than once	T	F
8	Assuming that CRASH FAULTS MAY occur, the xBroadcast primitive guarantees that if a process delivers a message m with sender s, then m was previously broadcast by process s.	T	F
9	Assuming that CRASH FAULTS MAY occur, the xBroadcast primitive guarantees that if a message m is delivered by some correct process, then m is eventually delivered by every correct process.	T	F
10	Assuming that CRASH FAULTS MAY occur, the xBroadcast primitive guarantees that if some process broadcasts message m1 before it broadcasts message m2, then no correct process delivers m2 unless it has already delivered m1	T	F

Ex4: Consider a distributed system where a (1,N)-RegularRegister primitive is deployed through the Read-One Write-All protocol. Each process is able to send 2 msg per second through perfect point-to-point links. A process takes 1 second on average to return from a read operation. The time that intercurs between <onrr, Write> events is 20 seconds on average. The <onrr, Read> is triggered with rate 0.1 requests per second by every process. Unless differently stated, the computation delay is negligible and processes are always correct. The inter-arrival and service times follow an exponential distribution.

Algorithm 4.1: Read-One Write-All

Implements:
(1, N)-RegularRegister, **instance** *onrr*.

Uses:
PerfectPointToPointLinks, **instance** *pl*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event ⟨ *onrr*, *Init* ⟩ **do**
val := ⊥;
correct := Π;
writeset := ∅;

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

upon event ⟨ *onrr*, *Read* ⟩ **do**
trigger ⟨ *onrr*, *ReadReturn* | *val* ⟩;

upon event ⟨ *onrr*, *Write* | *v* ⟩ **do**
trigger ⟨ *beb*, *Broadcast* | [WRITE, *v*] ⟩;

upon event ⟨ *beb*, *Deliver* | *q*, [WRITE, *v*] ⟩ **do**
val := *v*;
trigger ⟨ *pl*, *Send* | *q*, ACK ⟩;

upon event ⟨ *pl*, *Deliver* | *p*, ACK ⟩ **do**
writeset := *writeset* ∪ {*p*};

upon *correct* ⊆ *writeset* **do**
writeset := ∅;
trigger ⟨ *onrr*, *WriteReturn* ⟩;

upon event ⟨ *beb*, *Broadcast* | *m* ⟩ **do**
forall *q* ∈ Π **do**
trigger ⟨ *pl*, *Send* | *q*, *m* ⟩;

upon event ⟨ *pl*, *Deliver* | *p*, *m* ⟩ **do**
trigger ⟨ *beb*, *Deliver* | *p*, *m* ⟩;

Assess the truthfulness of every statement and provide a motivation for your answer:

1	The system is stable, assuming a workload of only Read operations, independently from the number of processes	T	F
2	The system is stable, assuming a workload of only Read operations, if the system is composed by 20 processes	T	F
3	The system is stable, assuming a workload of only Write operations, independently from the number of processes	T	F
4	The system is stable, assuming a workload of only Write operations, if the system is composed by 20 processes	T	F
5	The system is stable, assuming a workload of only Write operations, if the system is composed by 10 processes and the expected time that intercurs between <onrr, Write> and <onrr, WriteReturn> is 2/3 secs.	T	F
6	Assume that each process may fail by crash, with MTBF = 90h and MTTR = 10h associated with each process, and a workload of only Read operations, the steady-state availability perceived by a fixed process is independent of the number of processes in the system	T	F
7	Assume that each process may fay by crash, with MTBF = 90h and MTTR = 10h associated to each process, and a workload of only Write operations, the steady-state availability perceived by the writer process is below than 80%	T	F
8	Assume that each process may fay by crash, with MTBF = 90h and MTTR = 10h associated to each process, and a workload of only Write operations, the steady-state availability perceived by the writer process increases if we increase the number of processes.	T	F

Ex5: Let us consider a distributed system composed of a set of n processes $\{p_1, p_2, \ldots p_n\}$. Each process p_i has a partial knowledge of the system i.e., it just knows a subset of the system members whose identifiers are stored in a local variable called $\texttt{view}_i$. Every process p_i can communicate only with its neighbors (i.e., only with processes belonging to its local view) through perfect point-to-point links.

Assuming that:

- every process p_i knows the overall number n of processes in the system
- processes may fail by crash
- the overlay network resulting from the union of all the views is connected and remains connected when crash occurs
- the system is synchronous (clocks are synchronized, computation times are negligible and the communication latency over every perfect point to point link is bounded by a known constant δ)

write the pseudocode of a primitive implementing a perfect failure detector that is able to report crashes of any process in the system (i.e., it also reports failures of processes not included in the local view).

According to the Italian law 675 of the 31/12/96, I authorize the instructor of the course to publish on the web site of the course results of the exams.

Signature: _____

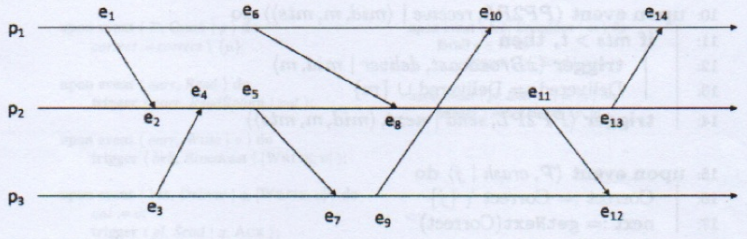
Ex1: Describe what a publish-subscribe communication system is highlighting its main characteristics. In addition compare the characteristics of the one-to-many communication obtained when leveraging a publish-subscribe system and a Reliable Broadcast-based communication system.

A PUBLISH-SUBSCRIBER COMMUNICATION SYSTEM IS AN ASYNCHRONOUS COMMUNICATION PARADIGM IN WHICH THE PRODUCERS OF MESSAGES (PUBLISHERS) DO NOT DIRECTLY SEND MESSAGES TO CONSUMERS (SUBSCRIBERS), BUT PUBLISH THEM ON TOPICS OR CHANNELS MANAGED BY AN INTERMEDIARY (BROKER).
THE SUBSCRIBER EXPRESS THEIR INTEREST IN SPECIFIC TOPICS AND ARE AUTOMATICALLY NOTIFIED OF RELEVANT MESSAGES WHEN THEY ARE PUBLISHED.

THIS APPROACH IS CHARACTERIZED BY SPACE AND TIME DECOUPLING, MEANING THAT INTERACTING PARTIES DON'T NEED TO KNOW EACH OTHER, AND THEY DON'T NEED TO PARTECIPATE IN THE INTERACTION AT THE SAME TIME, SO SYNCHRONIZATION BETWEEN THEM IS NOT NEEDED.

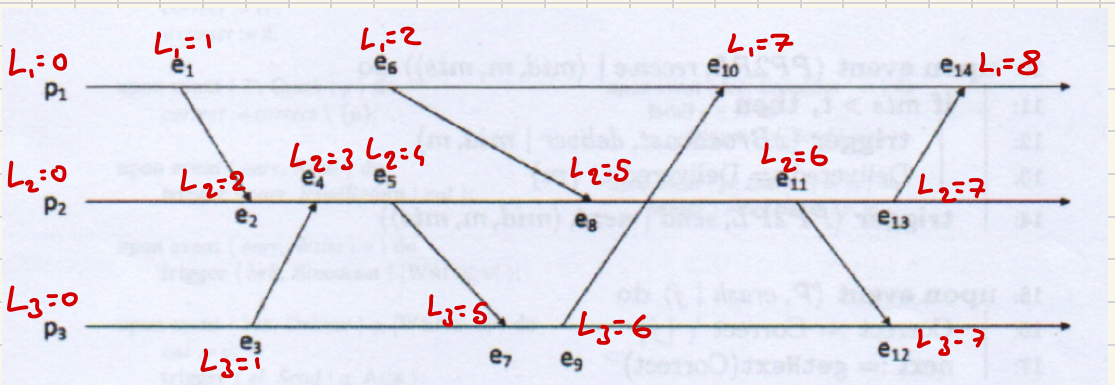
IN RB, EACH CORRECT PROCESS RECEIVES EVERY MESSAGE, SENDER AND RECIPIENTS ARE KNOW (NO DECOUPLING) AND IT'S LESS SCALABLE.

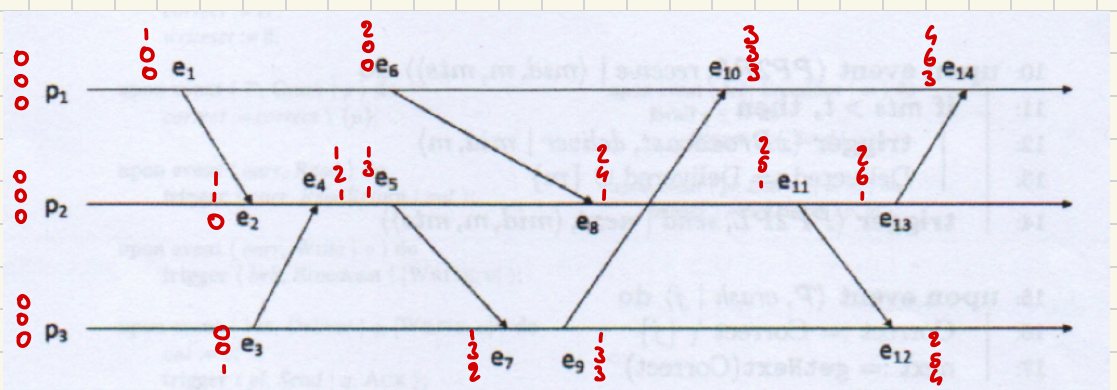
Ex2: Let us consider the following execution history



Let us denote with ck(ei) the logical clock associated with event ei. Considering the execution history shown in the figure above, assess the truthfulness of every statement and provide a motivation for your answer:

1	If we use scalar clocks for timestamping events, then ck (e6) = ck (e5)	T	F
2	e2 happened before e6 (according with Lamport's definition of happened-before)	T	F
3	If we use scalar clocks for timestamping events, then ck (e7) = ck (e3) +1	T	F
4	e7 and e8 are concurrent events	T	F
5	If we use scalar clocks for timestamping events, then ck (e13) > ck (e12)	T	F
6	If we use vector clocks for timestamping events, then ck (e13) = [4, 6, 0]	T	F
7	If we use vector clocks for timestamping events, then ck(e1) = ck (e3)	T	F
8	If we use vector clocks for timestamping events, then ck(e5) and ck (e6) are not comparable	T	F
9	If we use vector clocks for timestamping events, then ck(e8) < ck (e14)	T	F
10	If we use vector clocks for timestamping events, then ck (e4) = [0, 1, 1]	T	F





Ex3: Consider a distributed system composed of n processes, each with a unique integer sequential identifier. Processes can communicate by exchanging messages over perfect point-to-point links. Processes are equipped with a perfect failure detector P.
The xBroadcast primitive is deployed in such a system managing two events: < *xBroadcast, send | m* > and < *xBroadcast, deliver | ps, m* >, and is implemented via the following algorithm.

Algorithm 10 xBroadcast

```
1:  procedure INIT
2:  |  Correct := Π
3:  |  Delivered := ∅
4:  |  t_s := 0
5:  |  // id : process's identifier
6:  |  Next := (id + 1) mod n

7:  upon event ⟨xBroadcast, send | m⟩ do
8:  |  t_s := t_s + 1
9:  |  trigger ⟨PP2PL, send | next, ⟨id, m, t_s⟩⟩

10: upon event ⟨PP2PL, receive | ⟨mid, m, mts⟩⟩ do
11: |  if mts > t_s then
12: |  |  trigger ⟨xBroadcast, deliver | mid, m⟩
13: |  |  Delivered := Delivered ∪ {m}
14: |  trigger ⟨PP2PL, send | next, ⟨mid, m, mts⟩⟩

15: upon event ⟨P, crash | j⟩ do
16: |  Correct := Correct \ {j}
17: |  next := getNext(Correct)
```

Assess the truthfulness of every statement and provide a motivation for your answer:

1	Assuming that NO CRASH occurs, the xBroadcast primitive guarantees that if a correct process p broadcasts a message m, then p eventually delivers m	T	F ✗
2	Assuming that NO CRASH occurs, the xBroadcast primitive guarantees that no message is delivered more than once	T	F ✗
3	Assuming that NO CRASH occurs, the xBroadcast primitive guarantees that if a process delivers a message m with sender s, then m was previously broadcast by process s.	T ✓	F
4	Assuming that NO CRASH occurs, the xBroadcast primitive guarantees that if a message m is delivered by some correct process, then m is eventually delivered by every correct process.	T	F ✗
5	Assuming that NO CRASH occurs, the xBroadcast primitive guarantees that if some process broadcasts message m1 before it broadcasts message m2, then no correct process delivers m2 unless it has already delivered m1	T	F ✗
6	Assuming that CRASH FAULTS MAY occur, the xBroadcast primitive guarantees that if a correct process p broadcasts a message m, then p eventually delivers m	T	F ✗
7	Assuming that CRASH FAULTS MAY occur, the xBroadcast primitive guarantees that no message is delivered more than once	T	F ✗
8	Assuming that CRASH FAULTS MAY occur, the xBroadcast primitive guarantees that if a process delivers a message m with sender s, then m was previously broadcast by process s.	T ✓	F
9	Assuming that CRASH FAULTS MAY occur, the xBroadcast primitive guarantees that if a message m is delivered by some correct process, then m is eventually delivered by every correct process.	T	F
10	Assuming that CRASH FAULTS MAY occur, the xBroadcast primitive guarantees that if some process broadcasts message m1 before it broadcasts message m2, then no correct process delivers m2 unless it has already delivered m1	T	F

1) m IS FORWARDED IN A CIRCULAR WAY BETWEEN THE PROCESSES. WHEN m RETURN TO THE PROCESS THAT SENT IT, THE CONDITION IF $mts > ts$ WILL NOT BE SATISFIED AS $ts = mts$.

2) THERE IS NO IF m ∉ DELIVERED.

3) THE SENDER mID IS ALWAYS THE SAME.

4) SINCE THERE ARE NO CRASHS, EVERY PROCESS WILL RECEIVE AND DELIVER m BUT THE ONE THAT SENT IT.

5) THERE IS NO FIFO MECHANISM

6) AS 1

7) AS 2

8) AS 3

9)

10)

?

Ex5: Let us consider a distributed system composed of a set of n processes $\{p_1, p_2, \ldots p_n\}$. Each process p_i has a partial knowledge of the system i.e., it just knows a subset of the system members whose identifiers are stored in a local variable called $view_i$. Every process p_i can communicate only with its neighbors (i.e., only with processes belonging to its local view) through perfect point-to-point links.

Assuming that:

- every process p_i knows the overall number n of processes in the system
- processes may fail by crash
- the overlay network resulting from the union of all the views is connected and remains connected when crash occurs
- the system is synchronous (clocks are synchronized, computation times are negligible and the communication latency over every perfect point to point link is bounded by a known constant δ)

write the pseudocode of a primitive implementing a perfect failure detector that is able to report crashes of any process in the system (i.e., it also reports failures of processes not included in the local view).

```
UPON EVENT < PFD, INIT > DO
  VIEW= GET_VIEW()
  CORRECT_V = VIEW
  DETECTED= ∅
  STARTTIMER (2·δ)

UPON EVENT < TIMEOUT > DO
  FORALL p ∈ VIEW DO
    IF (p ∉ CORRECT_V) ∧ (p ∉ DETECTED) THEN
      DETECTED= DETECTED ∪ {p}
      TRIGGER < PFD, CRASH | p >
      FORALL q ∈ VIEW DO
        TRIGGER < PP2PL, SEND | [FAILURE, p] >

  FORALL p ∈ VIEW DO
    TRIGGER < PP2PL, SEND | p, HRTBTRQ >

  CORRECT_V = ∅
  STARTTIMER (2·δ)

UPON EVENT < PP2PL, DELIVER | q, HRTBTRQ > DO
  TRIGGER < PP2PL, SEND | q, HRTBTRPLY >

UPON EVENT < PP2PL, DELIVER | q, HRTBTRPLY > DO
  CORRECT_V = CORRECT_V ∪ {p}

UPON EVENT < PP2PL, DELIVER | [FAILURE, p] > DO
  IF p ∉ DETECTED THEN
    DETECTED= DETECTED ∪ {p}
    TRIGGER < PFD, CRASH | p >
    FORALL q ∈ VIEW DO
      TRIGGER < PP2PL, SEND | [FAILURE, p] >
```