A

**Dependable Distributed Systems (9 CFU)**

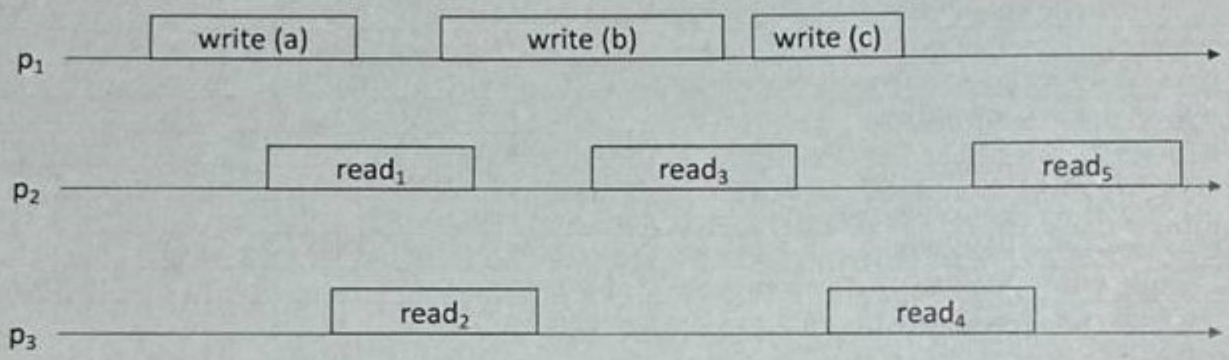
**14/02/2023**

**Exam A**

Family Name______________________ Name______________________ Student ID______________

**Ex 1:** Provide the specification of the regular consensus primitive and describe the implementation presented during the lectures in synchronous systems. Finally, discuss its performance (in terms of number of messages exchanged) to reach consensus.

**Ex 2:** Let us consider the following execution history



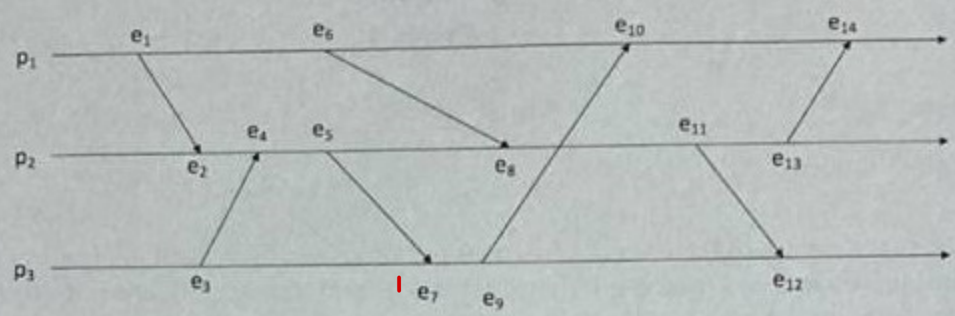

Assuming that the initial value stored in the register is 0, assess the truthfulness of every statement and provide a motivation for your answer:

| | | | |
|---|---|---|---|
| 1 | If the proposed run refers to a regular register, then 0 is a valid value for read$_1$ | T | F |
| 2 | If the proposed run refers to a regular register, then 0 is not a valid value for read$_2$ | T | F |
| 3 | If the proposed run refers to a regular register, then *a* is a valid value for read$_4$ | T | F |
| 4 | If the proposed run refers to a regular register, then *b* is a valid value all read operations | T | F |
| 5 | If the proposed run refers to a regular register, then read$_3$ may return only *b* and *c* | T | F |
| 6 | If the proposed run refers to an atomic register, then read$_1$ and read$_2$ must return the same value | T | F |
| 7 | If the proposed run refers to an atomic register, then read$_3$ returns *b* if and only if read$_1$ returns *b* | T | F |
| 8 | If the proposed run refers to an atomic register, then read$_4$ and read$_5$ must always return the same value | T | F |
| 9 | If the proposed run refers to an atomic register and read$_4$ returns *c*, then read$_3$ necessarily returned *c* | T | F |
| 10 | If the proposed run refers to an atomic register and read$_1$ returned *b* then read$_2$ can return only the value *b* | T | F |

**Ex 3:** Consider the broadcast communication primitives studied during the course, assess the truthfulness of every statement below and for each statement provide a motivation (also using suitable examples).

| | | | |
|---|---|---|---|
| 1 | If a run *R* satisfies Causal Order Broadcast, then *R* satisfies FIFO order Broadcast | T | F |
| 2 | If a run *R* satisfies Total Order Broadcast, then *R* satisfies Regular Reliable Broadcast | T | F |
| 3 | It is not possible to define a run *R* that satisfies both Total Order Broadcast and FIFO order Broadcast | T | F |
| 4 | If a run *R* satisfies TO (NUA, WUTO), then *R* satisfies also Uniform Reliable Broadcast | T | F |
| 5 | If a run *R* satisfies TO (NUA, WUTO), then *R* satisfies TO (NUA, WNUTO) | T | F |

**Ex 4:** Let us consider the following execution history




Let us denote with $ck(e_i)$ the logical clock associated to event $e_i$. 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 $ck(e_6) > ck(e_5)$ | T | F |
| 2 | $e_2$ happened before $e_3$ (according with Lamport's definition of happened-before) | T | F |
| 3 | If we use scalar clocks for timestamping events, then $ck(e_{14}) = ck(e_{10}) + 1$ | T | F |
| 4 | $e_6$ and $e_7$ are concurrent events | T | F |
| 5 | If we use scalar clocks for timestamping events, then $ck(e_9) < ck(e_{11})$ | T | F |
| 6 | If we use vector clocks for timestamping events, then $ck(e_{13}) = [4, 6, 4]$ | T | F |
| 7 | If we use vector clocks for timestamping events, then $ck(e_1) = ck(e_3)$ | T | F |
| 8 | If we use vector clocks for timestamping events, then $ck(e_5)$ and $ck(e_{10})$ are not comparable | T | F |
| 9 | If we use vector clocks for timestamping events, then $ck(e_8) < ck(e_{10})$ | T | F |
| 10 | If we use vector clocks for timestamping events, then $ck(e_4) = [1, 1, 1]$ | T | F |

**Ex 5:** Let us consider a distributed system composed of a set C={c1, c2, ... cm} of clients and a set R={r1, r2, ... rn} of replicas. Clients and replicas are univocally identified by an integer. Replicas communicate among themselves by message passing and are arranged in a unidirectional ring topology. Every replica ri can send messages (over a perfect point-to-point link) only to its neighbor whose name is stored in a local variable `next_i`. Replicas may fail by crash and every replica $r_i$ is equipped with a perfect oracle that notifies, through the `new_next(r_j)` event, the identifier $r_j$ of the new $r_i$'s neighbor in case of failure.
Replicas need to maintain a shared object by implementing the primary-backup replication schema.
Every client may communicate with replicas using perfect point-to-point links. Initially, clients only know the identifier of the current primary and they store it in a local variable `primary`. So, when they need to issue operations, they just need to interact with it.

Given the current scenario, answer the following points:

1. Write the pseudo-code of the distributed protocol implementing the replication schema (both client and replicas side).
2. Assuming a long enough period [t, t'] without any failure, compute the expected response time (from the client point of view) to execute an operation *op* knowing that the inter arrival time between requests is exponentially distributed with parameter $\lambda = 2$ req/sec, that the average service time to execute *op* on a replica is 1/4 sec, that all P2P links have a service time of 1/7 sec per message, and that all service times are exponentially distributed (the upstream and downstream of a P2P link can be assumed independent).
3. Assuming that any replica fails every 24h on average and that the replicas take around 30 minutes to elect a new primary, provide an estimation of the steady state availability of the shared object considering the previous times exponentially distributed.

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: ____________________

**Ex 1:** Provide the specification of the regular consensus primitive and describe the implementation presented during the lectures in synchronous systems. Finally, discuss its performance (in terms of number of messages exchanged) to reach consensus.

IT'S A MECHANISM THAT ALLOWS A SET OF PROCESSES IN A DISTRIBUTED SYSTEM TO AGREE ON A SINGLE VALUE, EVEN IN PRESENCE OF FAILURES. IT IS BASED ON:

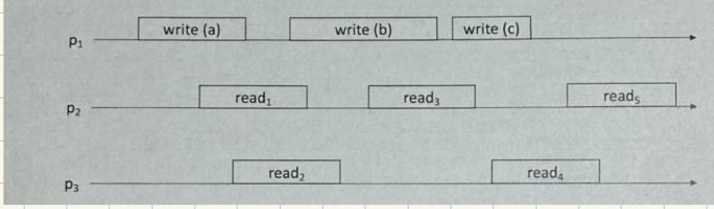
VALIDITY. IF A PROCESS DECIDES v, THEN v WAS PROPOSED BY SOME PROCESSES.
INTEGRITY: NO PROCESS DECIDES TWICE.
AGREEMENT: NO TWO CORRECT PROCESSES DECIDE DIFFERENTLY.
TERMINATION: EVERY CORRECT PROCESS EVENTUALLY DECIDES SOME VALUE.

IT'S IMPOSSIBLE TO GUARANTEE CONSENSUS IN AN ASYNCHRONOUS SYSTEM

FLOODING CONSENSUS IS AN ALGORITHM BASED ON A PROPAGATION FOR FLOODING MESSAGES TO ENSURE THAT ALL CORRECT PROCESSES REACH AN AGREEMENT ON THE VALUE TO BE DECIDED.
EACH PROCESS BEGINS BY PROPOSING A VALUE AND TRANSMITS IT TO OTHERS USING beb. THE PROCESSES COLLECT THE PROPOSALS RECEIVED IN THE FOLLOWING ROUNDS, UPDATING A SET OF CANDIDATE VALUES. IF A PROCESS RECEIVES MESSAGES FROM ALL THE NODES FROM WHICH HE HAD RECEIVED IN THE PREVIOUS ROUND, HE CHOOSES THE MINIMUM VALUE BETWEEN THOSE PROPOSED AND PROPAGATES IT AS A FINAL DECISION. THE PROCESS ENDS WHEN A DECISION MESSAGE IS DECIDED, WHICH IS IMMEDIATELY PROPAGATED TO GUARANTEE CONVERGENCE.

THE ALGORITHM HAS A COMPLEXITY EQUAL TO $O(n^2)$, WITH n # OF PROCESSES. EACH OF THEM SENDS ITS VALUE TO ALL THE OTHERS, AND EACH NODE FORWARDS THE VALUES RECEIVED, CAUSING A FLOODING EFFECT.

**Ex 2:** Let us consider the following execution history




Assuming that the initial value stored in the register is 0, assess the truthfulness of every statement and provide a motivation for your answer:

| | | | |
|---|---|---|---|
| 1 | If the proposed run refers to a regular register, then 0 is a valid value for read$_1$ | X | F |
| 2 | If the proposed run refers to a regular register, then 0 is not a valid value for read$_2$ | T | X |
| 3 | If the proposed run refers to a regular register, then $a$ is a valid value for read$_4$ | T | X |
| 4 | If the proposed run refers to a regular register, then $b$ is a valid value all read operations | T | X |
| 5 | If the proposed run refers to a regular register, then read$_3$ may return only $b$ and $c$ | T | X |
| 6 | If the proposed run refers to an atomic register, then read$_1$ and read$_2$ must return the same value | T | X |
| 7 | If the proposed run refers to an atomic register, then read$_3$ returns $b$ if and only if read$_1$ returns $b$ | T | X |
| 8 | If the proposed run refers to an atomic register, then read$_4$ and read$_5$ must always return the same value | T | X |
| 9 | If the proposed run refers to an atomic register and read$_4$ returns $c$, then read$_3$ necessarily returned $c$ | T | X |
| 10 | If the proposed run refers to an atomic register and read$_1$ returned $b$ then read$_2$ can return only the value $b$ | X | F |

1) $R_1()$: 0, a, b

2) $R_2()$: 0, a, b

3) $R_4()$: b, c

4) $R_5()$: c

5) $R_3$: a, b, c

6) $R_1$ AND $R_2$ ARE CONCURRENT

7) $R_3()$: b ALSO IF $R_1()$: a

8) $R_4$ CAN RETURN b BUT $R_5$ ALWAYS c

9) $R_3$ CAN RETURN a OR b

10) $R_1$ AND $R_2$ ARE CONCURRENT, $R_2$ IS THE SUBSEQUENT OF $R_1$ AND CANNOT READ c

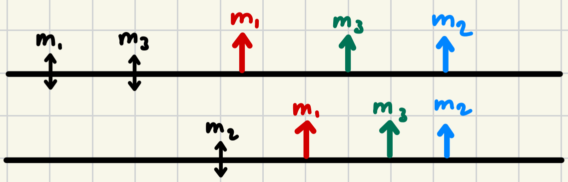
**Ex 3:** Consider the broadcast communication primitives studied during the course, assess the truthfulness of every statement below and for each statement provide a motivation (also using suitable examples).

| | | | |
|---|---|---|---|
| 1 | If a run *R* satisfies Causal Order Broadcast, then *R* satisfies FIFO order Broadcast | ~~T~~ | F |
| 2 | If a run *R* satisfies Total Order Broadcast, then *R* satisfies Regular Reliable Broadcast | ~~T~~ | F |
| 3 | It is not possible to define a run *R* that satisfies both Total Order Broadcast and FIFO order Broadcast | T | ~~F~~ |
| 4 | If a run *R* satisfies TO (NUA, WUTO), then *R* satisfies also Uniform Reliable Broadcast | T | ~~F~~ |
| 5 | If a run *R* satisfies TO (NUA, WUTO), then *R* satisfies TO (NUA, WNUTO) | ~~T~~ | F |

1) CASUAL ORDER = FIFO ORDER + LOCAL ORDER

2) TO IS A RB ORDER WITH ORDER PROPERTIE

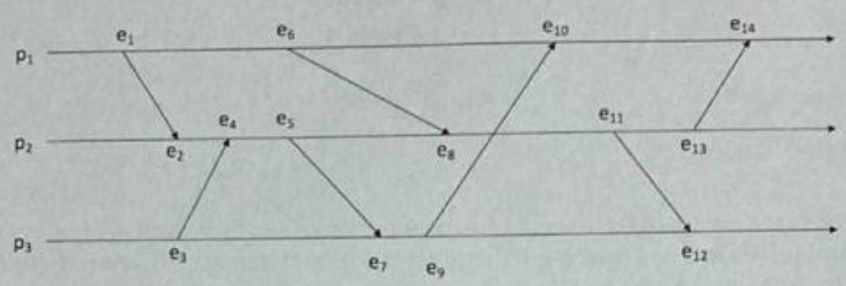
3)




4) NUA: FAULTY PROCESSES DELIVER MSGs THAT ARE NO DELIVERED BY THE CORRECT.
UNIFORM AGREEMENT: IF A PROCESS (CORRECT OR FAULTY) DELIVER m, EVERY CORRECT PROCESS DELIVERS m

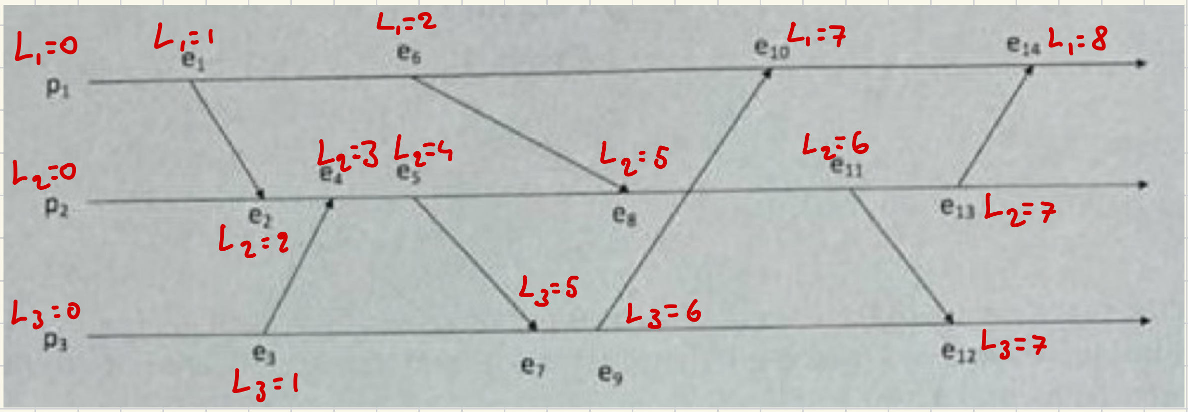
5) WNUTO IS A SUBSET OF WUTO

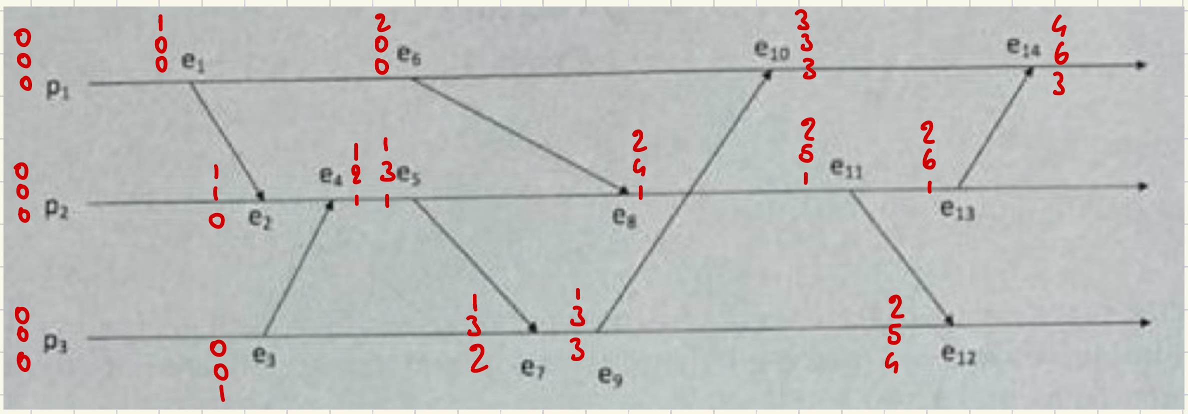
**Ex 4:** Let us consider the following execution history




Let us denote with ck($e_i$) the logical clock associated to event $e_i$. 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 ($e_6$) > ck ($e_5$) | T | X |
| 2 | $e_2$ happened before $e_3$ (according with Lamport's definition of happened-before) | T | X |
| 3 | If we use scalar clocks for timestamping events, then ck ($e_{14}$) = ck ($e_{10}$) +1 | X | F |
| 4 | $e_6$ and $e_7$ are concurrent events | X | F |
| 5 | If we use scalar clocks for timestamping events, then ck ($e_9$) < ck ($e_{11}$) | T | X |
| 6 | If we use vector clocks for timestamping events, then ck ($e_{13}$) = [4, 6, 4] | T | X |
| 7 | If we use vector clocks for timestamping events, then ck($e_1$) = ck ($e_3$) | T | X |
| 8 | If we use vector clocks for timestamping events, then ck($e_5$) and ck ($e_{10}$) are not comparable | T | X |
| 9 | If we use vector clocks for timestamping events, then ck($e_8$) < ck ($e_{10}$) | T | X |
| 10 | If we use vector clocks for timestamping events, then ck ($e_4$) = [1, 1, 1] | T | X |

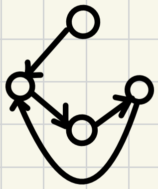


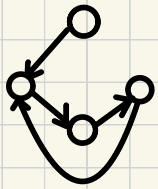



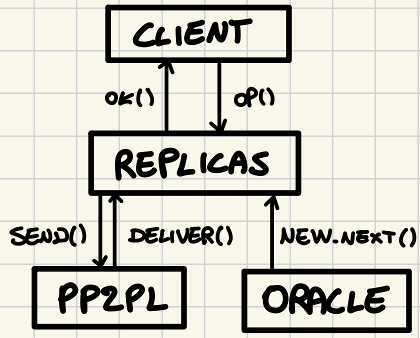
**Ex 5:** Let us consider a distributed system composed of a set C={c1, c2, … cm} of clients and a set R={r1, r2, … rn} of replicas. Clients and replicas are univocally identified by an integer. Replicas communicate among themselves by message passing and are arranged in a unidirectional ring topology. Every replica ri can send messages (over a perfect point-to-point link) only to its neighbor whose name is stored in a local variable `next_i`. Replicas may fail by crash and every replica $r_i$ is equipped with a perfect oracle that notifies, through the `new_next(r_j)` event, the identifier $r_j$ of the new $r_i$'s neighbor in case of failure.
Replicas need to maintain a shared object by implementing the primary-backup replication schema.
Every client may communicate with replicas using perfect point-to-point links. Initially, clients only know the identifier of the current primary and they store it in a local variable `primary`. So, when they need to issue operations, they just need to interact with it.

Given the current scenario, answer the following points:

1. Write the pseudo-code of the distributed protocol implementing the replication schema (both client and replicas side).








REPLICAS:

```
UPON EVENT <R, INIT> DO
    NEXT = SELF.NEXT()
    VAL_X = DEFAULT
    CORRECT = π
    PRIMARY = MAX_RANK (CORRECT)

UPON EVENT < R, OPERATION | V > DO
    VAL_X = EXECUTE OP (V)
    TRIGGER < PP2PL, SEND | [UPDATE, VAL.X, SELF] > TO NEXT

UPON EVENT <PP2PL, DELIVER | [UPDATE, VAL.X, P] >
    IF P = SELF THEN
        TRIGGER < R, OK | OP >
    ELSE
        VAL.X = VAL
        TRIGGER < PP2PL, SEND | [UPDATE, VAL_X, P]>

UPON EVENT < PO, NEW_NEXT | NEXT > DO
    FAIL = NEXT
    CORRECT = CORRECT \ {FAIL}
    IF FAIL == PRIMARY THEN
        PRIMARY = MAX.RANK (CORRECT)
    ELSE:
        TRIGGER < PP2PL, SEND | [UPDATE, VAL_X, P]
    TRIGGER < PP2PL, SEND | [CRASH, FAIL] > TO NEXT

UPON EVENT <PP2PL, DELIVER | [CRASH, FAIL] >
    CORRECT = CORRECT \ {FAIL}
    TRIGGER < PP2PL, SEND | [CRASH, FAIL] > TO NEXT
    IF FAIL == PRIMARY
        PRIMARY = MAX_RANK (CORRECT)
```

## CLIENT:

```
UPON EVENT <CL, INIT> DO
    OPERATIONS = ∅

UPON EVENT <CL, DELIVER | REQUEST, OP > DO
    OPERATIONS = OPERATIONS ∪ {OP}
    IF |OPERATIONS| == 1 THEN
        STARTTIMER (TIMER)

UPON EVENT <CL, OK | OP > DO
    OPERATIONS = OPERATIONS \ {OP}

UPON EVENT <TIMEOUT>
    IF OPERATIONS != ∅ THEN
        TRIGGER <PP2PL, SEND | OP, OPERATION (1)>
        STARTTIMER (TIMER)
```