A

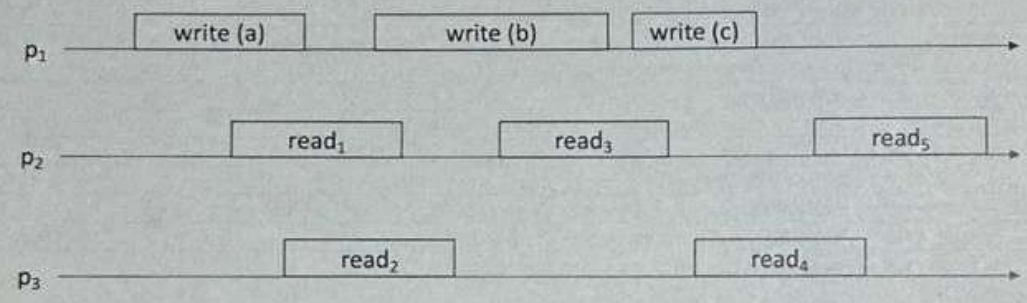
FATTO fuine ES.5

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



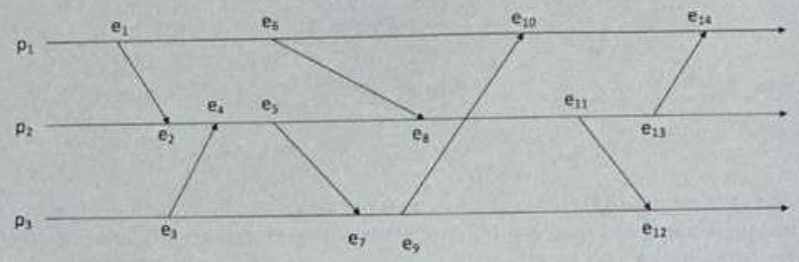
Assuming that the initial value stored in the register is 0, assess the truthfulness of every statement and provide a motivation for your answer:

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.

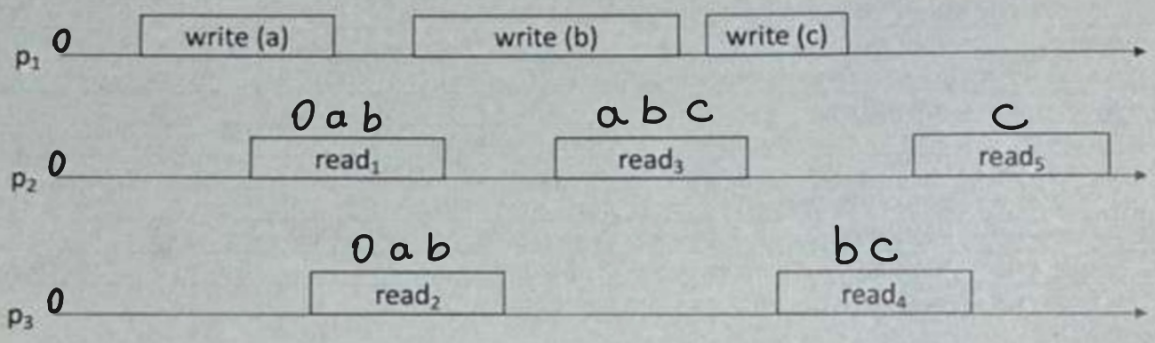
Given the current scenario, answer the following points:

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 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	T	X

Regular register: R_1 : 0 ab R_4 : bc

R_2 : 0 ab R_5 : c

R_3 : abc

Atomic register: R_1 : 0 ab R_4 : bc → if $R_3 = (a,b)$ then $R_4 = (b,c)$ otherwise $R_4 = c$

R_2 : 0 ab R_5 : c

R_3 : abc

↓

if $R_1/R_2 = 0 \rightarrow R_3 = (a,b,c)$

if $R_1/R_2 = a \rightarrow R_3 = (a,b,c)$

if $R_1/R_2 = b \rightarrow R_3 = (b,c)$

1) T, because 0 is the last written value

2) F, because $read_1$ and $read_2$ are concurrent so 0 is the last written value

3) F, because the last written value is b by the write(b) and the write(b) comes after write(a)

4) F, because the $read_5$ comes after the write(c) so the last written value is c (it is the only reading value for $read_5$)

5) F, $read_3$ can return b and c because is concurrent with write(b) and write(c), but the last written value is a.

6) F, because $read_1$ and $read_2$ are concurrent so both read the last written value (0), a (write(a)) and b (write(b)) but can return different value without violating the atomic register

7) F, the $read_3$ return b if $read_1 = (0, a, b)$. Particularly,
if $R_1 = 0 \rightarrow R_3 = (a,b,c)$
if $R_1 = a \rightarrow R_3 = (a,b,c)$
if $R_1 = b \rightarrow R_3 = (b,c)$

8) F, because $read_4$ and $read_5$ despite are concurrent have a different set of value. Indeed $read_4$ can return (b, c).

9) F, $read_3$ can return (a, b, c) accordingly to the previous execution.
For example, $R_1 = 0$ $R_2 = a$ $R_3 = b$ $R_4 = c$ $R_5 = c$

10) F, because the two read are concurrent.
For example, $R_1 = 0$ $R_2 = a$ $R_3 = b$ $R_4 = c$ $R_5 = c$

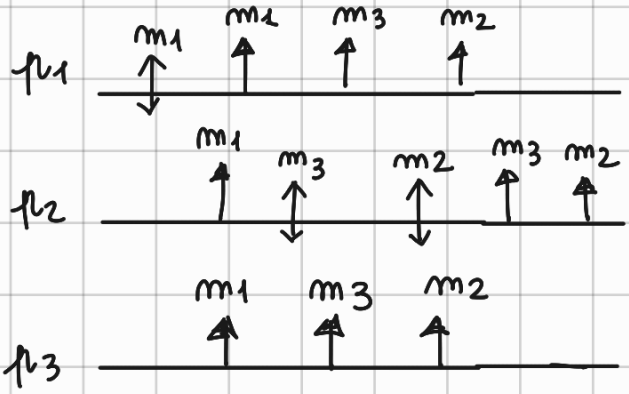
Ex 3: Consider the broadcast communication primitives studied during the course, assess the truthfulness of every statement below and for each statement provide a motivation (also using suitable examples).

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

1) T, because in order to have causal order we need to respect FIFO + local order

2) T, because the correct processes have the same set of delivery. TO implies RB.

3) F, because as follow:



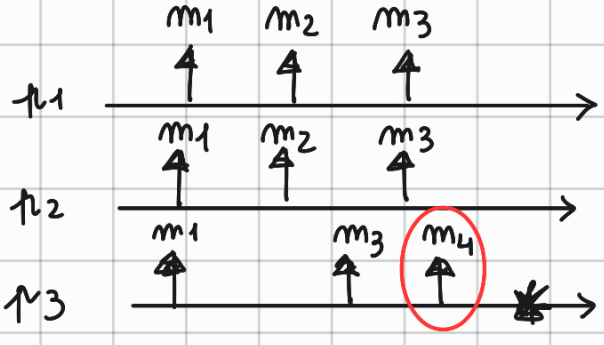
FIFO: $m_3 \rightarrow m_2$

TOTAL: same sequence for correct processes

$m_1 \rightarrow m_3 \rightarrow m_2$

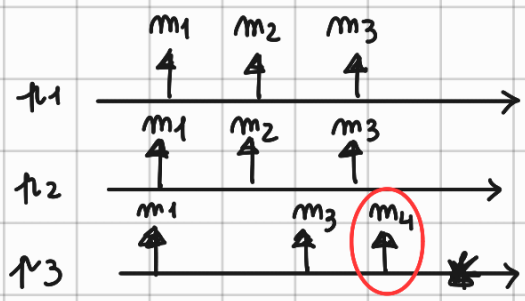
4) F, because URB thanks to the uniform agreement the set of the faulty process is a subset of the correct set and in TO (NUA, WUTO) we have no uniform agreement.

TO (NUA, WUTO)

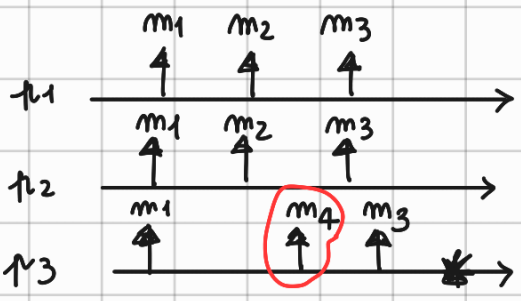


5) ~~F~~ T, because as follow:

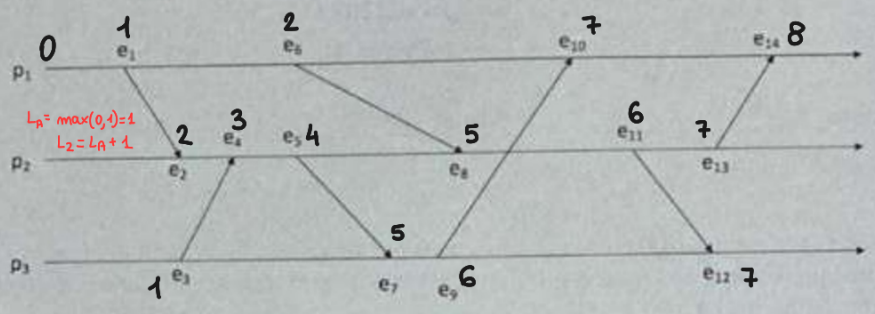
TO(NUA, WUTO)



TO(NUA, WNUTO)

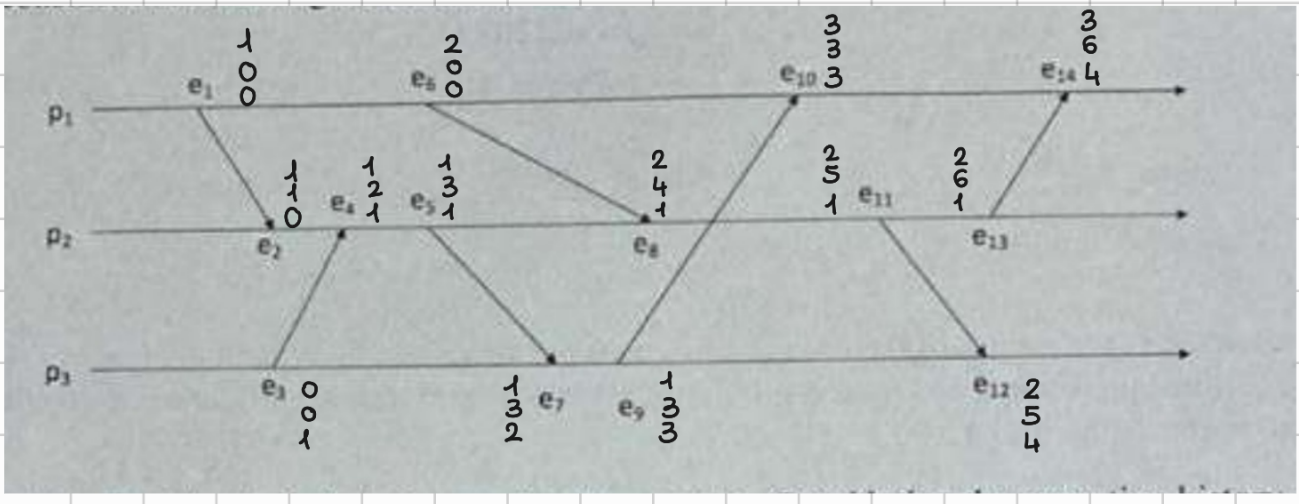


Ex 4: Let us consider the following execution history



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



8) F, because e_5 $\begin{pmatrix}1\\3\\1\end{pmatrix}$ $e_5 < e_{10}$

e_{10} $\begin{pmatrix}3\\3\\3\end{pmatrix}$ $\begin{pmatrix}1\\3\\1\end{pmatrix} < \begin{pmatrix}3\\3\\3\end{pmatrix}$: comparable

9) F, because e_8 $\begin{pmatrix}2\\4\\1\end{pmatrix}$ $e_8 < e_{10}$: concurrent

e_{10} $\begin{pmatrix}3\\3\\3\end{pmatrix}$ $\begin{pmatrix}2\\4\\1\end{pmatrix} < \begin{pmatrix}3\\3\\3\end{pmatrix}$ $4 < 3$ NO

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.

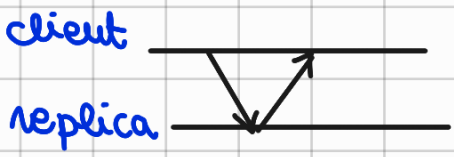
Given the current scenario, answer the following points:

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.

2) $\lambda = 2$ req/s

$\mu = 4$ req/s execute op on replica

$\mu = 7$ req/s ↑↓ per msg



① $R_1 = \frac{1}{\mu - \lambda} = \frac{1}{4-2}$

② $R_2 = \frac{1}{\mu - \lambda} = \frac{1}{7-2}$ $\quad R_2' = 2 \cdot R_2$

$R_{TOT} = R_1 + R_2'$

3) Fails every 24 h

30 m to elect new primary

$$A = \frac{MTBF}{MTBF + MTTR}$$

MTBF: t. before failure

MTTR: t. to repair

MTBF = 24 h = 1440 m

MTTR = 30 m

$$A = \frac{1440}{1440 + 30} = 0{,}979$$

for every replica