# Final Exam

Siddhant Dilip Godshalwar

May 1, 2023

## 1 Question 1

*Consider the following execution history of three processes.*

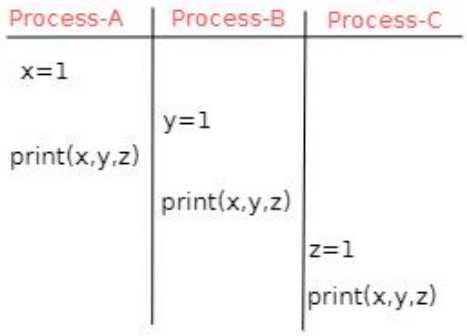


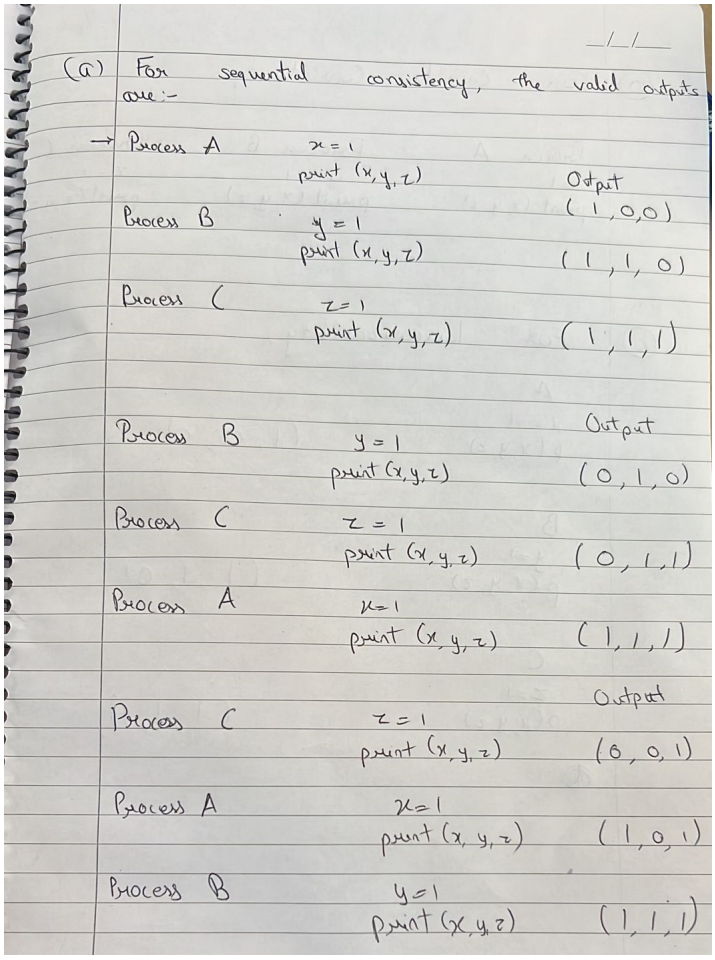

Figure 1: Execution histories of three processes

*(a) (5 points) What are all the valid outputs with sequential consistency? You can either enumerate all outputs or provide the general pattern.*

*(b) (5 points) All valid outputs for causal consistency.*

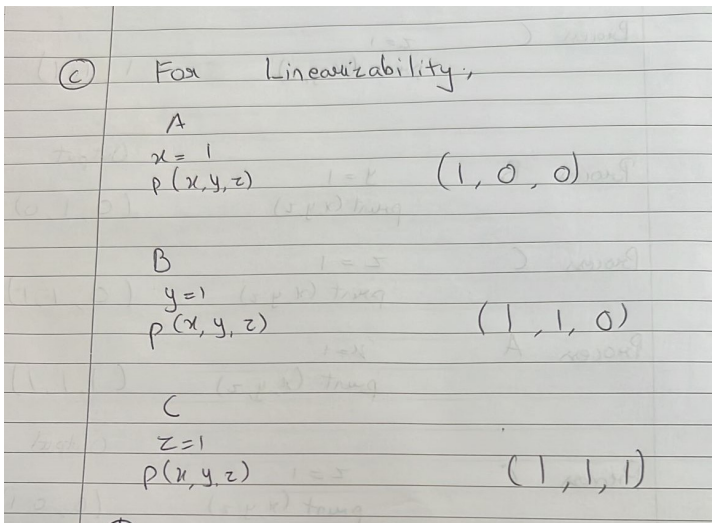
*(c) (5 points) All valid outputs with linearizability.*

**Answer A** For sequential consistency, the program execution order will never change and hence this will be the output for different scenarios considered.

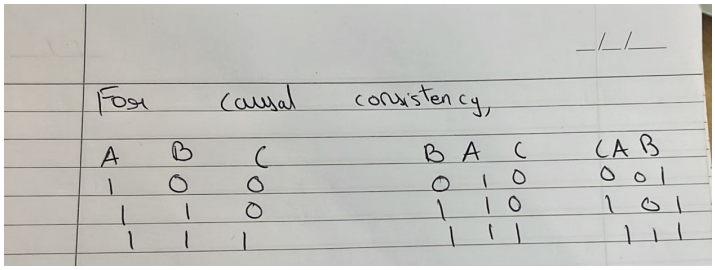



**Answer B**

There is only one write on each x,y and z and after that only print statements are performed hence, every combination is possible




**Answer C**

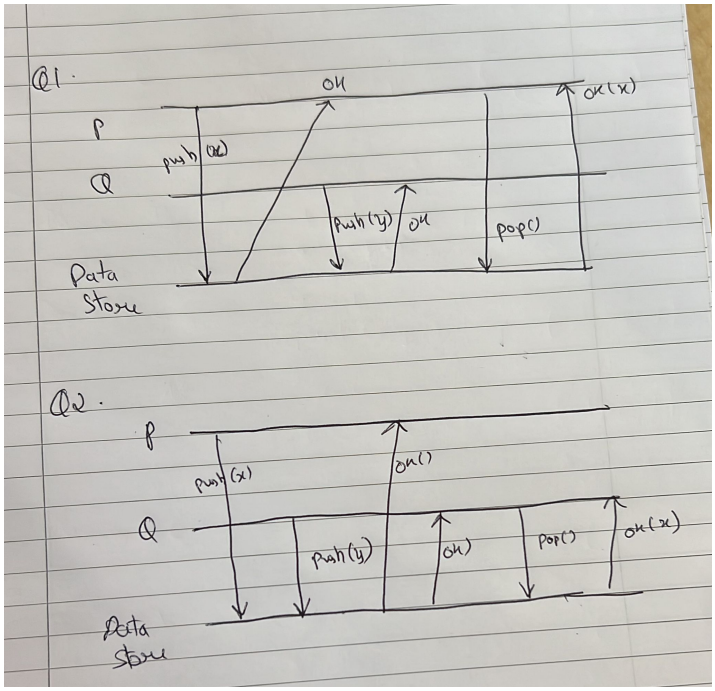



## 2 Question 2

*(5 points) Consider a concurrent stack accessed by processes P and Q. Which of the following histories are linearizable? Which of them are sequentially consistent? Justify your answer.*

1. *P:push(x), P:ok(), Q:push(y), Q:ok(), P:pop(), P:ok(x)*
2. *P:push(x), Q:push(y), P:ok(), Q:ok(), Q:pop(), Q:ok(x)*

**Answer:-**




**Answer A:-**

No, this sequence of operations is not sequentially consistent.

In a sequentially consistent system, the result of any execution should be the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by their program.

In this sequence of operations, we can see that P pushes element x onto its queue and signals that the push operation is complete by calling ok(). Then, Q pushes element y onto its queue and signals that the push operation is complete by calling ok(). Next, P pops an element from its queue and signals that the pop operation is complete by calling ok(x).

The issue with this sequence is that it violates the order specified by the program. Specifically, the ok(x) call from P occurs after the push and ok calls from Q. This violates the requirement that the operations of each individual processor appear in the order specified by their program.

Therefore, this sequence is not sequentially consistent but is linearizable.

**Answer B:-**

Yes, this sequence of operations is both sequentially consistent and linearizable.

As mentioned in my previous response, this sequence of operations is sequentially consistent because the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by their program.

In addition, this sequence is also linearizable, because it appears as if all operations were executed atomically in some order. Specifically, we can see that the P:push(x) and Q:push(y) operations occur first, followed by the P:ok() and Q:ok() operations. Then, the Q:pop() and Q:ok(x) operations occur.

If we consider the Q:pop() and Q:ok(x) operations to be a single atomic operation, then we can see that this sequence of operations appears as if all operations were executed atomically in the order P:push(x), Q:push(y), P:ok(), Q:ok(), Q:pop(), Q:ok(x).

Therefore, this sequence of operations is both sequentially consistent and linearizable

# 3 Question 3

*(10 points) For the local read algorithm for sequential consistency, show that it may generate a history that is not linearizable.*

***Answer:-***

The local read algorithm is a technique used to implement sequential consistency in a distributed system. However, this algorithm may generate histories that are not linearizable. Here is an explanation and itemization of this scenario:

Explanation:

Consider a scenario with two processors, P1 and P2, and one shared variable x. Let P1 write a value 1 to x, and then P2 reads the value of x and receives the value 1. However, due to network delays, the broadcast of the write operation from P1 to P2 is delayed. In the meantime, P2 performs another read operation on x and receives the value 0 from its local copy. Finally, the write operation from P1 is broadcast to P2.

This scenario results in the following history:

P1: write(x, 1)
P2: read(x):- 1
P2: read(x) :- 0
P1 broadcast write(x, 1)
Moreover:-

- The local read algorithm is a method used to implement sequential consistency in a distributed system.
- In this algorithm, each processor maintains a local copy of the shared memory.
- When a processor reads a value, it reads it from its local copy, and when it writes a value, it writes it to its local copy and broadcasts the new value to all other processors.
- However, the local read algorithm can generate histories that are not linearizable.
- A history is linearizable if it appears as if all operations were executed in some sequential order.
- In the scenario described above, due to network delays, P2 reads the value of x from P1's write operation after it has performed another read operation and received a different value from its local copy.
- As a result, the history generated by this scenario is not linearizable because there is no total order of operations that is consistent with the order of operations of each individual process.
- Any total order of operations would need to place the write operation before the first read operation of P2, which violates the order of operations of P2.

# 4 Question 4

*Failures with different system configurations*

*(a) (5 points) Assume you have fully replicated data store implemented on N servers, with each server replicating the entire dataset. It is sufficient for clients to access any single server (i.e., eventual consistency). If the MTTF of each server is M, what is the overall MTTF of the entire system?*

*(b) (5 points) Now assume that this data store provides strict consistency (using the local read algorithm). All the N servers are part of the sequentially consistent data storage system, and each server has MTTF=M. What is the effective MTTF of this setup?*

**Answer A:-**

MTTFofSys = 1 / ((1 / M) + (1 / M) + ... + (1 / M)) (N times)

MTTFofSys = 1 / (N / M)

MTTFofSys = M / N

# 5 Question 5

*CAP Theorem*

*(a) (5 points) State the CAP theorem as precisely as you can. Explain what C, A, and P mean. Give an intuitive explanation for why it holds.*

*(b) (5 points) Your friend Lee.T.Hacker excitedly tells you that he has made a distributed key-value store, and he says, "In my testing, I can provide all the properties mentioned in the CAP theorem". Should you believe him? Why or why not?*

***Answer A:-*** According to the CAP theorem, it is impossible to concurrently guarantee all three of the following qualities in a distributed system:

**1** Consistency (C): Every read receives the most recent write or an error.

**2** Availability (A): Every request receives a response, without guarantee that it contains the most recent version of the information.

**3** Partition tolerance (P): The system continues to operate despite arbitrary message loss or failure of part of the system.

Precisely, Only two of the three properties can be provided by a distributed system. In distributed systems, partition tolerance is considered a must, therefore the tradeoff is between consistency and availability in the event of network partition.

Intuitively, The CAP theorem describes a fundamental tradeoff in distributed systems, which are made up of multiple nodes that communicate with each other to share information. Maintaining consistency, which means that all nodes have the same view of the data, and availability, which means that nodes can quickly respond to requests, are both important goals. However, network partitions and failures can make it difficult to achieve both simultaneously. When a network partition occurs, nodes on each side of the partition cannot communicate with each other, leading to potential inconsistencies in the data. In this situation, a choice must be made between maintaining consistency and responding quickly to requests. The CAP theorem suggests that in the event of a network partition, a distributed system can only guarantee two out of the three properties of consistency, availability, and partition tolerance. Therefore, architects of distributed systems must carefully consider which properties to prioritize in their design based on the specific needs of their application.

***Answer B:-*** We should believe Lee.T.Hacker's claims as the CAP theorem in itself states that it is impossible to achieve all 3 properties in a distributed system with network partition.

If Lee.T.Hacker states that his system can deliver all three features (consistency, availability, and partition tolerance), he is either uninformed or intentionally misrepresenting its capabilities. His system may deliver excellent consistency and partition tolerance while sacrificing availability during network partitions. His method might also prioritize availability during network partitions without sacrificing consistency.

As a result, before adopting Lee.T.Hacker's dubious claim, it is critical to have a better knowledge of his method and properly test it under various scenarios.

# 6 Question 6

*(10 points) Explain how to implement a set CRDT*

***Answer:-*** A set CRDT (Conflict-free Replicated Data Type) can be implemented using the following steps:

- Partition-tolerant sets: Create two sets, one for added items and one for deleted items, which can handle network partitions.
- Merge operation: The merge operation is the set union operation. When a set is merged with another set, the added items are merged and the deleted items are discarded.
- Actual set: The actual set is the result of subtracting deleted items from added items.
- Prune sets: When the system is unpartitioned, the sets can be pruned by applying the delete operations to remove deleted items.
- Deep result: Combining two CRDT's results in a CRDT, as the CRDT's ensure conflict-free replication of data.
- Complex CRDT's: CRDT's can be combined to form more complex CRDT's such as sets, maps, counters, graphs, sequences, and JSON objects.

By following the above steps, we can implement a set CRDT that can handle network partitions and ensure eventual consistency of the data. The use of CRDT's makes it easier to handle network partitions, as the conflict resolution is done automatically. This results in less overhead and more reliable and fault-tolerant systems.

# 7 Question 7

*Paxos Scenarios*

*(a) (5 points) Acceptor A1 has already responded to a prepare message from proposer P1 with ballot number 10, and also responded to accept message from P1 with the same ballot number (10) with value 3. Assume that A1 has not received any prepare or accept messages with ballot number greater than 10 yet. In each case below, determine whether A1 will respond if it now receives the specified message, and if so, what would be the contents of its response:*

*1. A1 receives a prepare message containing ballot number 8 and value 5.*

*2. A1 receives a prepare message containing ballot number 11 and value 6*

*(b) (5 points) Suppose that proposer P1 sends a prepare message with ballot number 25 and value 9, and receives responses from a quorum (majority) of acceptors. Acceptor A1's response to P1 contains a proposal with ballot number 22 and value 17; Acceptor A2's response to P1 contains a proposal with ballot number 23 and value 6; the remaining responses do not contain a proposal. Which proposal number and value would P1 include in its accept messages?*

***Answer A:-***

- 1. In the Paxos consensus protocol, an acceptor can only accept a proposal with a ballot number greater than or equal to any previous proposal it has seen. In this scenario, the proposer sends a prepare message with ballot number 10 and value 5. Acceptor A1 has previously accepted a proposal with ballot number 8, which is less than 10. Therefore, A1 will not respond to the prepare message, as it cannot accept a proposal with a lower ballot number.
- 2. A1 has received a prepare message with number 10 from the proposer. Since A1 has already accepted proposal (10,3), which has a number of 10, it will respond with a promise not to accept any proposal with a number less than 10. Additionally, A1 will include its highest numbered proposal (10,3) in its response, along with a new sequence number of 11 to indicate the response to the prepare message. The response message from A1 will be (11,(10,3)).

***Answer B:-*** P1 would send accept(25,6) to the acceptors, i.e, the proposal number will be 25. The accept message (25,6) means that:

1. 25 is the sequence number chosen by P1.
2. 6 is the value of the proposal with the largest sequence number that P1 has received in responses from the acceptors.
3. By sending this message, P1 is requesting that the acceptors accept the proposal with value 6 for the chosen sequence number 25.

# 8 Question 8

*More Paxos*

*(a) (5 points) Suppose 3 out of 5 acceptors have accepted a proposal with value X. Once this has happened, is it possible that any server in the cluster could accept a different value Y? Recall that any node can crash-fail at any time. Explain your answer, and be explicit about how different failure scenarios affect the consensus result.*

*(b) (5 points) Does Paxos always terminate? Provide either a proof-sketch or counter example*

***Answer A:-***
**CASE 1:** When all 5 acceptors are functioning correctly and communicating with each other:

- Once 3 out of 5 acceptors have accepted a proposal with value X, no other server in the cluster can accept a different value Y.
- This is because the Paxos protocol requires a majority of acceptors to agree on a proposal before it can be accepted.
- Therefore, any subsequent proposal with a different value will be rejected by the acceptors, as it cannot achieve a majority.

**CASE 2:** When one acceptor fails and is unable to communicate with the other acceptors:

- It is possible that a different value could be accepted by the remaining acceptors.
- This is because the remaining 4 acceptors are not a majority, so they could potentially split into two groups of two that each accept a different value.
- In this case, the consensus cannot be reached, as no value can achieve a majority.

**CASE 3:** When more than one acceptor fails and the cluster enters a "split-brain" situation:

- Different subsets of acceptors are unable to communicate with each other and accept different values, leading to a lack of consensus.
- This is a more severe failure scenario that can occur in Paxos.
- It can be mitigated by implementing additional measures such as leader election and quorum rules.

***Answer B:-*** 1. The protocol starts with a single proposer proposing a value to the acceptors.
2. If a majority of the acceptors accept the proposal, the proposer sends a message to the acceptors to commit the value.
3. Once the acceptors receive the commit message, they commit the value and send an acknowledgement to the proposer.
4. The proposer waits for acknowledgements from a majority of the acceptors before committing the value.

To see why Paxos always terminates, consider the following:
1. If a proposal is accepted by a majority of acceptors, it is guaranteed to be committed. This is because any two majorities of acceptors must have at least one acceptor in common, and that acceptor will have accepted the proposal.

2. If a proposal is not accepted by a majority of acceptors, the proposer can choose a new proposal number and try again.

3. The proposer cannot be stuck in an infinite loop because there are only a finite number of proposal numbers to try.

4. The protocol ensures that eventually a proposal will be accepted by a majority of acceptors because the acceptors will continue to respond to prepare and accept messages until they receive a message from a proposer with a higher proposal number.

Therefore, Paxos always terminates and reaches a consensus on a single value.

# 9 Question 9

*(10 points) Alice and Bob are neighbors, and they share a yard. Alice owns a cat and Bob owns a dog. Both pets like to run around in the yard, but (naturally) they do not get along. After some unfortunate experiences, Alice and Bob agree that they should coordinate to make sure that both pets are never in the yard at the same time. Of course, we rule out trivial solutions that do not allow any animals into an empty yard. For coordination, each of them sets up a flag pole in their own section of the yard. The flags are visible to each other across the yard. Write the protocol for Alice and Bob for safely sharing the yard, using the flags. This is a mutual exclusion problem. They cannot communicate in any other way.*

**Ans:** We can implement the following algorithm in this scenario:-

1. Both Alice and Bob start with their flags down.

2. Whenever Alice wants to send her pet in the yard, she will hoist the flag showing that the yard is occupied and Bob shouldn't let his dog out right now.

3. Once Alice is happy with her yard time, she can lower the flag and now Bob can take up the yard, starting with hoisting the flag and then letting his pet out.

This solution is great but it needs to ensure fairness and also, as Alice and Bob are humans who don't have the time to constantly keep checking out the window for a hoisted flag, we can start with setting up some ground rules in an update to this algorithm:-

Rule 1:- When flag is hoisted, the other person cannot let their pet in the yard.

Rule 2:- If you want to raise a request to occupy the yard, you can hoist the flag at half mast length.

Rule 3:- You can pre set an amount of time which the person who has occupied the yard has (let's say an hour) to clear the yard once a request(half mast flag) has been made.

Although this solution isn't an exact replica of the mutual exclusion algorithm, it does save the humans in this scenario, a ton of effort.

# 10 Question 10

*(10 points) Free points. What were the most interesting parts of the course?*

***Ans:-*** The most interesting parts of the course were for me in the first half of the semester. The Mapreduce and other Multicast theories and their explainations were very intriguing.