Name **: Sabarivasan V**
Class **: CSE - B**
Reg No **: 205001085**

Consider 4 processes P1, P2, P3 and P4 in a distributed system. The Resource request model is expressed as P1→ P2 || P3 → P4 || P1.
(Note: → indicates sequential and || indicate concurrent executions)

a. Apply Lamport's D-Mutex algorithm for the given resource request model.

Lamport's Distributed Mutual Exclusion Algorithm is a permission based algorithm proposed by Lamport as an illustration of his synchronization scheme for distributed systems. In permission based timestamp is used to order critical section requests and to resolve any conflict between requests. In Lamport's Algorithm, critical section requests are executed in the increasing order of timestamps i.e a request with smaller timestamp will be given permission to execute critical section first than a request with larger timestamp. In this algorithm:

Three types of messages (REQUEST, REPLY and RELEASE) are used and communication channels are assumed to follow FIFO order.

- A site sends a REQUEST message to all other sites to get their permission to enter a critical section.
- A site sends a REPLY message to a requesting site to give its permission to enter the critical section.
- A site sends a RELEASE message to all other sites upon exiting the critical section.
- Every site, Si, keeps a queue to store critical section requests ordered by their timestamps. request_queuei denotes the queue of site Si
- A timestamp is given to each critical section request using Lamport's logical clock. Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section requests is always in the order of their timestamp.

DS Assignment -2

205001085
V Sabarivasan
CSE -B

$P_1 \rightarrow P_2 || P_3 \rightarrow P_4 || P_1$

P_1

~~(1, P_1)~~
~~(8, P_2)~~
~~(8, P_3)~~
~~(17, P_1)~~
~~(17, P_4)~~

P_2

~~(1, P_1)~~
~~(8, P_2)~~
~~(8, P_3)~~
~~(17, P_1)~~
~~(17, P_4)~~

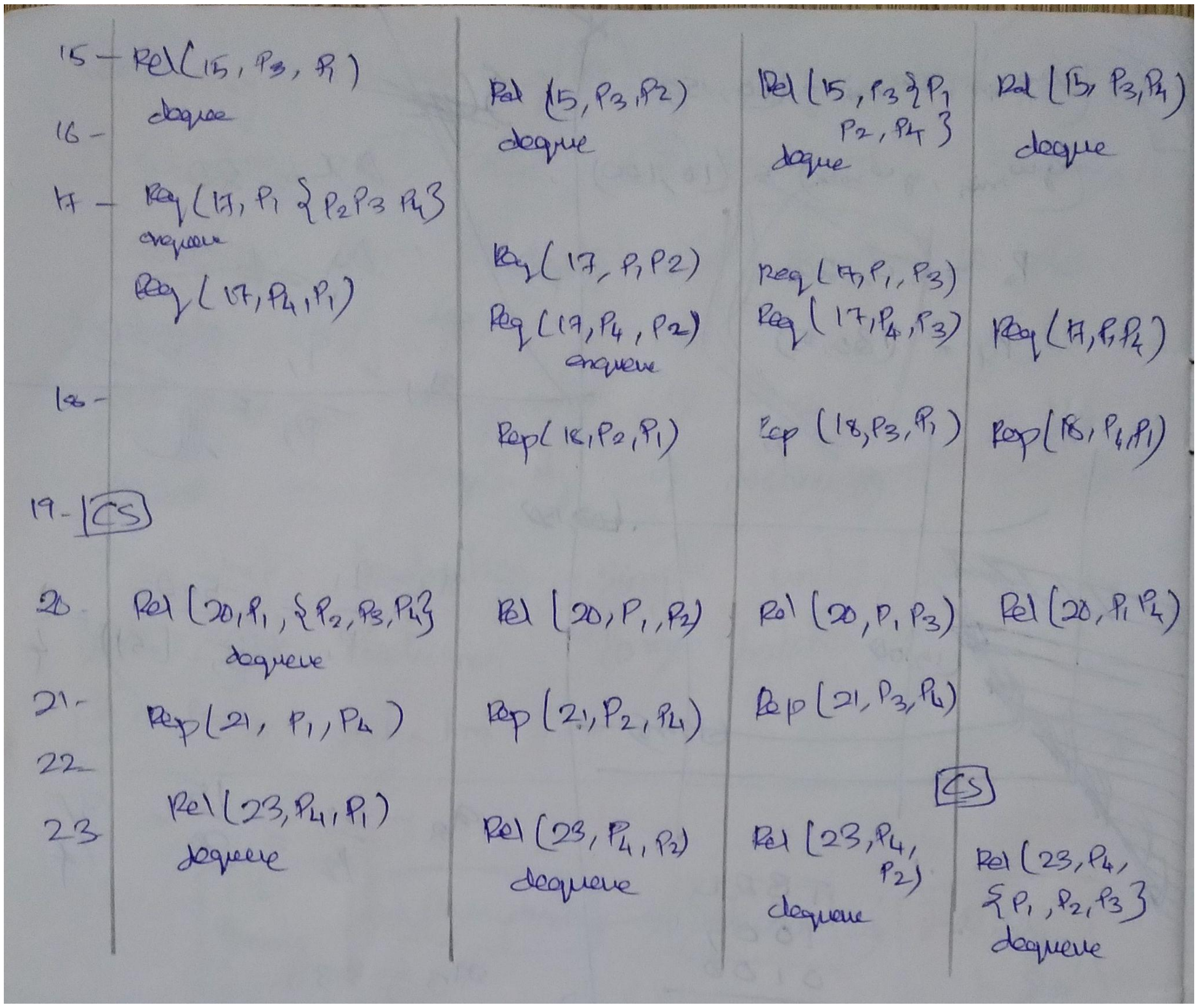
P_3

~~(1, P_1)~~
~~(8, P_2)~~
~~(8, P_2)~~
~~(17, P_1)~~
~~(17, P_4)~~

P_4

~~(1, P_1)~~
~~(8, P_2)~~
~~(8, P_3)~~
~~(17, P_4)~~
~~(17, P_1)~~

	P_1	P_2	P_3	P_4
1 -	Req (P, P_1 {P_2 P_3 P_4} enqueue			
2 -		Req (1, P_1 P_2) enqueue	Req (1, P_1, P_4) enqueue	Req (1, P_1 P_4) enqueue
3 -		Rep (3, P_2, P_1)	Rep (3, P_4, P_1)	Rep (3, P_4, P_1)
	$\boxed{CS}$			
6 -	Rel (6, P_1, {P_2 P_3 P_4}) dequeue	Rel (6, P_1, P_3) deque	Rel (6, P_1, P_4) deque	Rel (6, P_1, P_2) deque
7 -				
8 -	Req (8, P_2, P_1) Req (8, P_3, P_1)	Req (8, P_2 {P_1, P_3, P_4} Req (8, P_3, P_2)	Req (8, P_3, {P_1, P_2, P_4} Req (8, P_2, P_3)	Req (8, P_2, P_4) Req (8, P_3, P_4)
9 -				
10 -		$\boxed{CS}$		
11 -	Rel (11, P_2, P_1) deque	Rel (11, P_2, {P_1 P_3 P_4}) deque	Rel (11, P_2 P_3) deque	Rel (11, P_2, P_4) deque
12 -				
13 -	Rep (13, P_1, P_3)	Rep (13, P_2, P_3)		Rep (13, P_4, P_3)
			$\boxed{CS}$	



15 –	Rel(15, P3, P1)	Rel (15, P3, P2)	Rel (15, P3 {P1 P2, P4}	Rel (15, P3, P4)
16 –	deque	deque	deque	deque
17 –	Req (17, P1 {P2 P3 P4} enqueue	Req (17, P1 P2)	req (17, P1, P3)	
	Req (17, P4, P1)	Req (17, P4, P2) enqueue	Req (17, P4, P3)	Req (17, P1 P4)
18 –		Rep (18, P2, P1)	Rep (18, P3, P1)	Rep (18, P4, P1)
19 – $\boxed{CS}$				
20	Rel (20, P1, {P2, P3, P4} dequeue	Rel (20, P1, P2)	Rel (20, P1 P3)	Rel (20, P1 P4)
21 –	Rep (21, P1, P4)	Rep (21, P2, P4)	Rep (21, P3, P4)	
22				$\boxed{CS}$
23	Rel (23, P4, P1) dequeue	Rel (23, P4, P2) dequeue	Rel (23, P4, P2) dequeue	Rel (23, P4, {P1, P2, P3} dequeue

b. Inspect the steps for the occurrence of starvation

Starvation occurs when a process is unable to make progress because it cannot acquire the necessary resources. In the provided resource request model (P1 → P2 || P3 → P4 || P1), let's analyze the steps to identify potential scenarios of starvation:

P1 → P2: Process P1 requests resources from P2. Until P2 releases the resources, P1 cannot proceed. If P2 never releases the resources, P1 could potentially starve.

P3 → P4: Process P3 requests resources from P4. Until P4 releases the resources, P3 cannot proceed. If P4 never releases the resources, P3 could potentially starve.

Concurrent execution (||): The concurrent execution of P1 → P2 and P3 → P4 suggests that P1 and P3 are making resource requests simultaneously. However, there is no indication of how resources are allocated or whether there are any mechanisms to ensure fair access. If one process consistently acquires resources over the other, the starved process may face delays in execution.

Starvation in a distributed system often depends on the resource allocation policy and the fairness mechanisms in place. If the system is designed to prioritize certain processes over others, those that are lower in priority might face starvation.

c. Conclude whether the system suffers due to starvation or not for the given scenario.

In the given scenario and from the steps of progress, there is no evidence of starvation. Each process eventually enters the critical section and no process is indefinitely denied access to the resume. The algorithm guarantees progress and ensures fairness in accessing the critical section. There is no indication of a process being consistently denied access to the resource.

The progress from P3 → P4 alone can cause a delay. This is because P1 is given chance before P4 which means P4 is delayed. This does not result in no progress or consistant denial for accessing the critical section.

From part B inspection we can conclude that the system does not suffer due to starvation.

d. Examine the importance of reliability of the processes involved in the system.

Reliability is a crucial aspect in the context of distributed systems, and it becomes especially important in the scenario described with processes P1, P2, P3, and P4 in the resource request model (P1 → P2 || P3 → P4 || P1). Here are several reasons why the reliability of processes is of paramount importance:

Resource Management: In a distributed system, processes often compete for shared resources. If a process is unreliable and frequently fails or experiences errors, it can disrupt the overall resource management. Unreliable processes may not release resources properly, leading to resource leaks, contention, and potential system instability.

Deadlocks and Starvation: Reliable processes are essential to avoid deadlocks and starvation. If a process becomes unreliable and fails to release resources, it may lead to deadlocks where processes are unable to proceed. Conversely, if a process consistently fails to acquire resources due to unreliability, it may suffer from starvation.

Consistency and Correctness: Reliability is critical for maintaining the consistency and correctness of the distributed system. If a process fails to execute its tasks correctly or consistently, it can introduce inconsistencies in the system state, leading to unpredictable behavior and potential data corruption.

Fault Tolerance: Reliable processes contribute to the overall fault tolerance of the distributed system. A reliable process is more likely to handle errors gracefully, recover from failures, and continue functioning in the presence of faults. Unreliable processes, on the other hand, can introduce vulnerabilities and increase the risk of cascading failures.

System Availability: The reliability of processes directly impacts the availability of the distributed system. Unreliable processes may experience frequent downtime or failures, leading to service disruptions and decreased overall system availability.

Data Integrity: Reliable processes are essential for ensuring the integrity of data in a distributed system. If a process is prone to errors or failures, it may compromise the accuracy and reliability of the data it manages, leading to data corruption and potential data loss.

The Lamport's D- Mutex algorithm is based on reliable communication and reliability of the processes. Hence reliability of the processes involved in the system is an important key aspect in the algorithm.