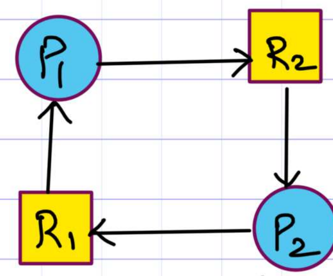
Distributed Mutual Exclusion (D-Mutex)

In distributed systems, state is not consistent at all times.



Deadlock

Solutions

1) non token based.
2) token based.

[Token - similar to the token ring topology of networks]

Non Token Based Algorithm

Designed by Leslie Lamport.

1) uses distributed queues
2) broadcast
3) uses Lamport logical clock.

Phases : Request, Reply, Release.

Request :

1) increment the sequence number.
2) place req(s, i). [i-process id]
3) broadcast to everyone.

Reply :

1) the receiver replies immediately when the receiver is not requesting critical section.

Release :

1) When exec. of CS is over, release (s,i) is broadcasted.
Causally, receiver deletes req (s, i) on receiving.

Example:

	P_1	P_2	P_3
1	s = 0	s = 0	s = 0
2	req(1, 1)	rec req(1, 1)	rec req(1, 1)
3		rep req (1, 1)	rep req (1, 1)
4			
5	rec rep (1, 1) P_2		
6	rec rep (1, 1) P_3		
7	exec CS		
8	rel req(1, 1)		
9		rec rel (1, 1)	rec rel (1, 1)
10		del req (1, 1)	del req (1, 1)

P_1 becomes malicious node (under DOS attack)

Maintain fairness by : If P_i is in CS, no more requests until release is issued by P_i.

$\boxed{\text{Message Complexity : } 3(N-1).}$

$\boxed{\text{Synchronization Delay : } T \text{ (round-trip delay)}}$

$\boxed{\text{System Throughput : } (T+E)^{-1}}$

Example:

P_1 queue	P_2 queue	P_3 queue
~~(1,1)~~ ~~(2,2)~~	~~(1,1)~~ ~~(2,2)~~	~~(1,1)~~ ~~(2,2)~~

	P_1	P_2	P_3
0	s = 0	s = 0	s = 0
1	req(1,1)		
2		rec req(1,1)	rec req(1,1)
3		req(2,2)	rep (1,1)
4	rec req(2,2)	rep (1,1)	rec req(2,2)
5	rec rep(1,1) P_2		rep (2,2)
6	rec rep(1,1) P_3	rec rep(2,2) P_3	
7	exec CS		
8	del req(1,1)		
9	rel (1,1)		
10	rep(2,2)	rec rel(1,1)	rec rel(1,1)
		del req(1,1)	del req(1,1)
11		rec rep(2,2) P_1	
12		exec CS	
		del req(2,2)	
		rel req(2,2)	
13	rec rel(2,2)		rec rel(2,2)
	del req(2,2)		del req(2,2)

Though P_2 is part of CS, it replies immediately, because P_1's request is in top of queue.

P_1 does not reply immediately for req(2,2) because its CS is in top of queue. It deserves to execute CS first.

Example:

(1,1) is before (1,2) because of the deadlock relinquish

P1 queue	P2 queue
~~(1,1)~~ ~~(1,2)~~	~~(1,1)~~ ~~(1,2)~~

	P1		P2
1	S=0		S=0
2	req (1,1)		req (1,2)
3	rec req (1,2)	DEADLOCK.	rec req (1,1)

P2 IS FORCED TO RELINQUISH SINCE PID OF P2 > P1. (2 > 1).

	P1	P2
4		rep (1,1)
5	rec rep(1,1) P2	
6	exec CS	
7	del (1,1)	
8	rel (1,1)	
9	rep (1,2)	rec rel (1,1)
		del (1,1)
10		rec rep (1,2) P1
11		exec CS
12		del (2,1)
13		rel (1,2)
14	rec rel(1,2)	
15	del (1,2)	

P1 queue	P2 queue
(1,1) ~~(2,1)~~	~~(2,1)~~ (3,2)

	P1		P2
1	S=0		S=0
2	req (1,1)	delayed.	
3	req (2,1)		
4			rec req(2,1)
5			rep (2,1)
6	rec rep(2,1)		req (3,2)
7	exec CS		

always update clock value.

8	delayed reception @ P2	← rec req (1,1)
9	(do not enqueue) ← ignore request (1,1). (message is outdated)	ignore req (1,1)

Requirements for MutEx Algorithms:

- Safety Property – only one P_i @ CS at all times.
- Liveness Property – no endless wait, i.e. no deadlock/starvation.
- Fairness – each P_i gets fair chance to execute CS. (i.e. by FIFO, logical clock)

Performance Metrics:

- Message Complexity – messages per CS execution.
- Synchronization Delay (SD) – time between last CS exit & next CS start.
- Response Time (RT) – $\boxed{[T_{cs\ exit} - T_{cs\ req}]}$
- System Throughput (ST) – rate @ which system executes CS requests.

$\boxed{ST = [SD + E]^{-1}}$. (E : avg. CS exec time)

- Low load & High Load Performance
  (seldom > 1 req.) (always pending reqs. present)
- Best & Worst Case Performance

(T: round-trip msg. delay) $\boxed{\text{Best RT} = 2T + E}$ (E: exec. time)

Proof By Contradiction

Queue

(1, 1)	(2, 1)	

Queue

(1,1) ×
(2,1)

Let us assume 2 processes are in CS.
⇒ Queue looks like above.

(1, 1) ⟶ (2, 1) (happened before)

The second request can only be made after the first request. (since seq. no. was incremented)

⇒ They happened at 2 different time instants, contrary to our assumption that both are in CS at the same instant.

Thus our Mutual Exclusion algorithm is valid.

Optimizing the Algorithm

Ricart - Agarwal Algorithm.

From 3(N-1) complexity to 2(N-1).

Release is a post-CS operation.
We can fine tune the release part.

2 rounds :

Request ⟶ N - 1
Reply. ⟶ N - 1

No release.

P1	P2	
(1, 1)	(2, 2)	– reply
(1, 1)	(1, 2)	– forced to reply (P2)
(2, 1)	(1, 2)	– no reply

tie forced by lower Pid no. first.

	P_1	P_2	P_3
Q	~~(1,1)~~ ~~(1,2)~~	~~(1,1)~~ ~~(1,2)~~ swap	~~(1,2)~~ ~~(1,1)~~
	s = 0	s = 0	s = 0
	req (1,1)	req (1,2)	rec req (1,2)
	rec req (2,1)	rec req (1,1)	rec req (1,1)
		rep (1,1) forced	rep (1,2)
	rec rep (1,1):P2		rep (1,1)
	rec rep (1,1): P3	rec rep (1,2) : P3	
	exec CS		
	del req (1,1)		
	rep (1,2)		
		rec rep (1,2) : P1	
		exec CS	
		del req (1,2)	

	P_1	P_2	P_3
Q	~~(1,2)~~ ~~(1,3)~~	~~(1,2)~~ ~~(1,3)~~	swap. ~~(1,3)~~ ~~(1,2)~~
	s = 0	s = 0	s = 0
		req (1, 2)	req (1, 3)
	rec req (1, 2)	rec req (1, 3)	rec req (1, 2)
	rec req (1, 3)		rep (1, 2) forced.
and deleted from queue	rep (1,2)	rec rep (1,2) : P3	
	rep (1,3)	rec req (1, 2) : P1	
		exec CS	rec rep (1,3) : P1
	exit CS	del req (1,2)	
		rep (1,3)	
			rec rep (1,3) : P2
			exec CS
			del req (1,3)