9 DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS

mutual exclusion ensures that concurrent access of processes to a shared resource (or data) is serialized.

- token-based

  process holding unique token can enter CS. (critical section)

  algos differ by how they search for token.

- non-token based

  1 or more rounds of messages are exchanged to determine who can enter CS next. a process enters CS when an assertion becomes true (and false for everyone else).

- quorum based

  subsets of processes (quorums) are formed in such a way that when 2 processes request CS, at least one process recieves both requests and is responsible to make sure only one process gets CS. (critical)

algorithm performance in load:

low load → seldom more than one request for CS.

heavy load → always a pending request for CS.

(a process is seldom in idle state in heavy load)

algorithm best and worst case performance:

most algos → best response time = 2T + E   round trip msg delay

often, best and worst cases coincide with low and high loads.

process P_i in one of 3 states:

- requesting CS (wait)
- executing CS
- neither (idle)

algorithm requirements:

- safety: only 1 process in CS
- liveness: no deadlock or starvation
- fairness: CS executed in order of arrival

performance metrics:

- message complexity

  no of msgs per CS execution

- synchronization delay SD

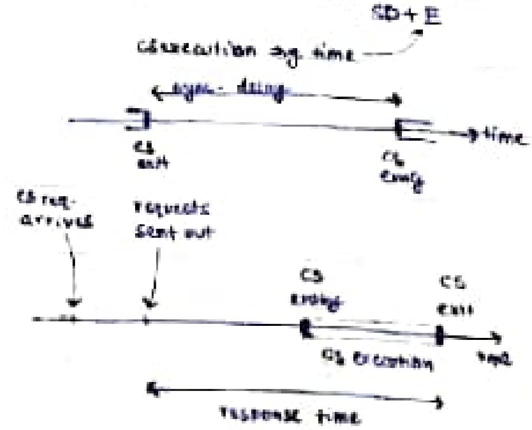
  time gap between CS exit & entry

- response time

  time gap b/w CS request & exit.

- system throughput

  rate of CS execution $\frac{1}{SD + E}$



LAMPORT'S ALGORITHM

processes use scalar clocks and request_queue to perform mutual exclusion. request_queue is ordered by timestamps. process at the top of request_queue enters CS. requires FIFO channels for fairness, and mut. ex.

Proof: mutual exclusion achieved. (by contradiction)

• assume P_1, P_2 in CS — ①

• assume P_1 REQUEST < P_2 REQUEST (ts) — ②

① ⇒ P2 sent REPLY to P1

⇒ P2 has P1 REQUEST before P1 REPLY (FIFO channels)

② ⇒ P1 REQUEST at top of P2 request_queue

but ① ⇒ P2 REQUEST at top of P2 request_queue.

contradiction ⇒ mutual exclusion achieved.

proof: fairness achieved (by contradiction)

assume P2 in CS (before P1) — ①

assume P1 REQUEST < P2 REQUEST (ts) — ②

① ⇒ P2 recieved P_1's REPLY

⇒ P2 recieved P1 REQUEST (FIFO channel)

② ⇒ P1 REQUEST at top of P2 request_queue

but ① ⇒ P2 REQUEST at top of P2 request_queue.

contradiction ⇒ fairness achieved.

optimization:

at P_i, P_i REQUEST < P_j REQUEST

⇒ no need to REPLY P_j

⇒ $O(2(n-1)) \underset{\text{btwn}}{\leftrightarrow} O(3(n-1))$

CS request:

• P_i broadcasts REQUEST(t_i, i). (inc. itself)

• P_j on recieving REQUEST(t_i, i) returns timestamped REPLY to P_i and adds REQUEST to request_queue$_j$.

execute CS:

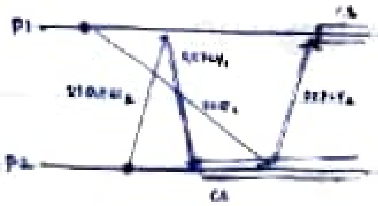
L1: P_i recieved REPLY with larger timestamp from all.

L2: P_i's REQUEST at top of request_queue$_i$.

release CS:

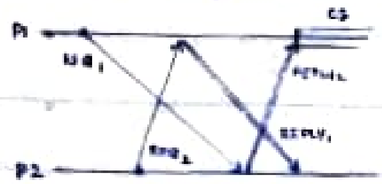
• P_i on CS exit broadcasts timestamped RELEASE to all (inc. itself).

• P_j on recieving RELEASE, removes P_i's REQUEST from request_queue$_j$.



non-FIFO channel

mutual exclusion fails



FIFO channel

mutual exclusion & fairness achieved

msg complexity = $O(3(n-1))$ REQUEST REPLY RELEASE

RICART - AGARWALA ALGORITHM

Processes use scalar clocks and request deferred array to perform mutual exclusion without a FIFO channel.

request CS:

- P_i broadcasts timestamped REQUEST.
- P_j on recieving REQUEST from P_i sends REPLY only if P_j is not executing CS and P_j REQUEST timestamp is larger than P_i REQUEST. else, reply is deferred and P_j sets $RD_j[i] = 1$.

execute CS:

- P_i recieved REPLY from all.

release CS:

- P_i sends all deferred REPLY messages: $\forall RD_i[j] = 1$, P_i sends REPLY to P_j and sets $RD_i[j] = 0$.

proof: mutual exclusion achieved (by contradiction)

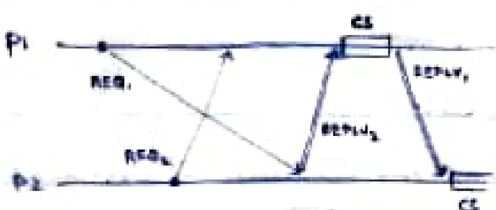
assume P1, P2 in CS — ①

assume P1 REQUEST < P2 REQUEST (ts) — ②

① ⇒ P2 has P1 REPLY

⇒ P1 not in CS & P1 REQ > P2 REQ

contradiction ⇒ mutual exclusion achieved.



msg complexity = $O(2(n-1))$ REQUEST REPLY

MAEKAWA'S ALGORITHM

first quorum based algorithm. a process only requests a subset of processes (quorum). two simultaneously requesting processes within their quorum will have atleast 1 common process, which REPLYs to only one of the two. mutual exclusion achieved.

conditions for request sets — request set (quorum)

$M1: R_i \cap R_j \neq \Phi \quad \forall i \neq j$

$M2: P_i \in R_i \quad \forall i$

$M3: |R_i| = K \quad \forall i$

$M4: P_j \in K$ no. of R_i's $\forall i, j$

$N = K(K-1)+1$ — no. of processes

$\therefore |R_i| = \sqrt{N}$ (theory of projective planes) proof: mutual exclusion achieved (by contradiction)

assume P_i, P_j in CS — ①

assume $R_i \cap R_j = \{P_{ij}\}$ — ②

① ⇒ P_i got REPLY from all in R_i

① ⇒ P_j got REPLY from all in R_j

② ⇒ P_{ij} sent REPLY to P_i and P_j.

contradiction ⇒ mutual exclusion achieved

msg complexity = $O(3\sqrt{N})$ REQUEST REPLY RELEASE

request CS:

- P_i sends REQUEST(i) to all processes in R_i.
- P_j on recieving REQUEST(i), sends REPLY(j) only if it hasn't sent REPLY to a process since last RELEASE msg. else, it queues REQUEST(i) for later consideration.

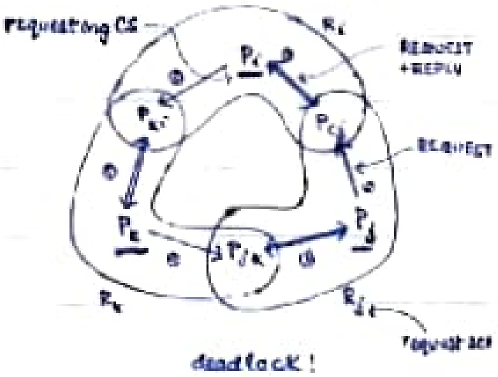
execute CS:

- P_i enters CS after it recieved REPLY from every process in R_i.

release CS:

- P_i sends RELEASE(i) to every site in R_i.
- P_j on recieving RELEASE(i), sends REPLY to next process in its queue after deleting P_i. if queue is empty, it updates its state to no REPLY sent since last RELEASE.

maekawa's algorithm can deadlock because a process is locked by other processes, and requests are not prioritized by timestamps.



deadlock!

handling deadlocks:

- when REQUEST(t, j) from P_j blocks at P_{ij} because P_{ij} sent REPLY to P_i, P_{ij} sends FAILED(ij) to P_j if its request has lower priority. else P_{ij} sends INQUIRE(ij) to P_i.
- in response to INQUIRE(ij) from P_{ij}, P_i sends YIELD(i) to P_{ij} if P_i has recieved FAILED from a process in R_i, and if it sent YIELD to any one but not recieved new REPLY from it.
- in response to YIELD(i) from P_i, P_{ij} assumes as if it has been released by P_i, places request of P_i at appropriate location in queue, and sends a REPLY(ij) to process at top of queue.

msg complexity = $O(5\sqrt{N})$