Lamport's Algorithm

- Requests for CS are executed in the increasing order of timestamps and time is determined by logical clocks.
- Every site S_i keeps a queue, $request\_queue_i$, which contains mutual exclusion requests ordered by their timestamps.
- This algorithm requires communication channels to deliver messages the FIFO order.

The Algorithm

Requesting the critical section:

- When a site S_i wants to enter the CS, it broadcasts a REQUEST(ts_i, i) message to all other sites and places the request on $request\_queue_i$. ((ts_i, i) denotes the timestamp of the request.)
- When a site S_j receives the REQUEST(ts_i, i) message from site S_i,places site S_i's request on $request\_queue_j$ and it returns a timestamped REPLY message to S_i.

Executing the critical section: Site S_i enters the CS when the following two conditions hold:

L1: S_i has received a message with timestamp larger than (ts_i, i) from all other sites.

L2: S_i's request is at the top of $request\_queue_i$.

The Algorithm

Releasing the critical section:

- Site S_i, upon exiting the CS, removes its request from the top of its request queue and broadcasts a timestamped RELEASE message to all other sites.
- When a site S_j receives a RELEASE message from site S_i, it removes S_i's request from its request queue.

When a site removes a request from its request queue, its own request may come at the top of the queue, enabling it to enter the CS.

correctness

Theorem: Lamport's algorithm achieves mutual exclusion.
Proof:

- Proof is by contradiction. Suppose two sites S_i and S_j are executing the CS concurrently. For this to happen conditions L1 and L2 must hold at both the sites *concurrently*.
- This implies that at some instant in time, say t, both S_i and S_j have their own requests at the top of their *request_queues* and condition L1 holds at them. Without loss of generality, assume that S_i's request has smaller timestamp than the request of S_j.
- From condition L1 and FIFO property of the communication channels, it is clear that at instant t the request of S_i must be present in $request\_queue_j$ when S_j was executing its CS. This implies that S_j's own request is at the top of its own *request_queue* when a smaller timestamp request, S_i's request, is present in the $request\_queue_j$ – a contradiction!

correctness

Theorem: Lamport's algorithm is fair.
Proof:

- The proof is by contradiction. Suppose a site S_i's request has a smaller timestamp than the request of another site S_j and S_j is able to execute the CS before S_i.
- For S_j to execute the CS, it has to satisfy the conditions L1 and L2. This implies that at some instant in time say t, S_j has its own request at the top of its queue and it has also received a message with timestamp larger than the timestamp of its request from all other sites.
- But *request_queue* at a site is ordered by timestamp, and according to our assumption S_i has lower timestamp. So S_i's request must be placed ahead of the S_j's request in the $request\_queue_j$. This is a contradiction!

Performance

- For each CS execution, Lamport's algorithm requires $(N-1)$ REQUEST messages, $(N-1)$ REPLY messages, and $(N-1)$ RELEASE messages.
- Thus, Lamport's algorithm requires $3(N-1)$ messages per CS invocation.
- Synchronization delay in the algorithm is T.

An optimization

- In Lamport's algorithm,REPLY messages can be omitted in certain situations. For example, if site S_j receives a REQUEST message from site S_i after it has sent its own REQUEST message with timestamp higher than the timestamp of site S_i's request, then site S_j need not send a REPLY message to site S_i.
- This is because when site S_i receives site S_j's request with timestamp higher than its own, it can conclude that site S_j does not have any smaller timestamp request which is still pending.
- With this optimization, Lamport's algorithm requires between $3(N-1)$ and $2(N-1)$ messages per CS execution.