LAMPORT'S ALGORITHM

- 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.
- Three type of messages (REQUEST, REPLY and RELEASE) are used and communication channels are assumed to follow FIFO order.
- A site send a REQUEST message to all other site to get their permission to enter critical section.
- A site send a REPLY message to requesting site to give its permission to enter the critical section.
- A site send a RELEASE message to all other site 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 request is always in the order of their timestamp.

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, it places site S_i's request on $request\_queue_j$ and 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 $request\_queue_i$.

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.

Fig: Lamport's distributed mutual exclusion algorithm

To enter Critical section:

- When a site S_i wants to enter the critical section, it sends a request message Request(ts_i, i) to all other sites and places the request on request_queue$_i$. Here, T_{si} denotes the timestamp of Site S_i.
- When a site S_j receives the request message REQUEST(ts_i, i) from site Si, it returns a timestamped REPLY message to site Si and places the request of site Si on request_queue$_j$

To execute the critical section:

- A site S_i can enter the critical section if it has received the message with timestamp larger than (ts_i, i) from all other sites and its own request is at the top of request_queue$_i$.

To release the critical section:

- When a site S_i exits the critical section, it removes its own request from the top of its request queue and sends a timestamped RELEASE message to all other sites. When a site S_j receives the timestamped RELEASE message from site S_i, it removes the request of S_{ia} from its request queue.

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 Sj 's own request is at the top of its own request queue when a smaller timestamp request, Si 's request, is present in the request queue$_j$ – a contradiction!

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 Si 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!

Message Complexity:

Lamport's Algorithm requires invocation of 3(N – 1) messages per critical section execution. These 3(N – 1) messages involves

- (N – 1) request messages
- (N – 1) reply messages
- (N – 1) release messages

Drawbacks of Lamport's Algorithm:

- **Unreliable approach**: failure of any one of the processes will halt the progress of entire system.
- **High message complexity:** Algorithm requires 3(N-1) messages per critical section invocation.

Performance:

Synchronization delay is equal to maximum message transmission time. It requires 3(N – 1) messages per CS execution. Algorithm can be optimized to 2(N – 1) messages by omitting the REPLY message in some situations.