

LAMPORT'S ALGORITHM FOR MUTUAL EXCLUSION IN DISTRIBUTED SYSTEM

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 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 S_i , **keeps a queue** to store critical section **requests ordered by their timestamps.**
request_queue_i denotes the queue of site S_i
- **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.

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, Ts_i denotes the timestamp of Site S_i
 - When a site S_j receives the request message **REQUEST(ts_i, i)** from site S_i , it returns a **timestamped REPLY** message to site S_i and places the request of site S_i 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_i from its request queue

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.