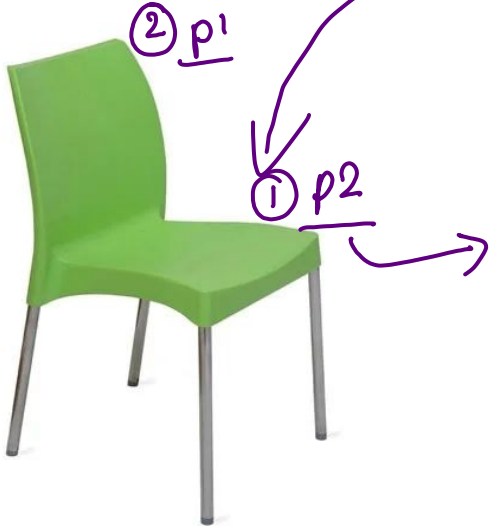
CS 3551 DISTRIBUTED COMPUTING

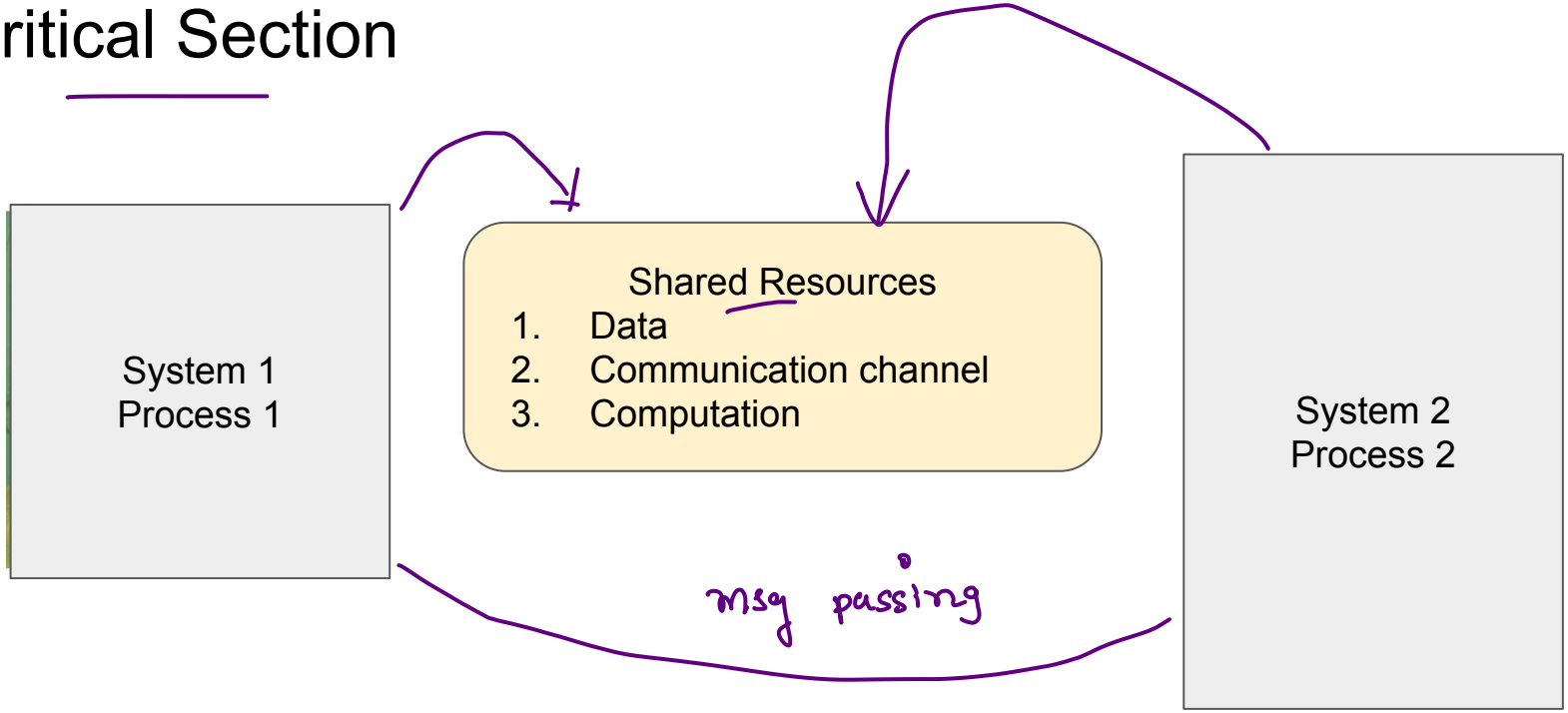
UNIT III DISTRIBUTED MUTEX AND DEADLOCK **10**

Distributed Mutual exclusion Algorithms: Introduction – Preliminaries – Lamport's algorithm – Ricart-Agrawala's Algorithm — Token-Based Algorithms – Suzuki-Kasami's Broadcast Algorithm; Deadlock Detection in Distributed Systems: Introduction – System Model – Preliminaries – Models of Deadlocks – Chandy-Misra-Haas Algorithm for the AND model and OR Model.

Critical Section + Mutual Exclusion



Critical Section



OS
- Semaphore
- Mutex locks

Critical Section

- In process synchronization, a critical section is a section of code that accesses shared resources such as variables or data structures, and which must be executed by only one process at a time to avoid race conditions and other synchronization-related issues.
- Mutual exclusion ensures that concurrent access of processes to a shared resource or data is serialized, that is, executed in a mutually exclusive manner
- In a distributed system, shared variables (semaphores) or a local kernel cannot be used to implement mutual exclusion.
- Message passing is the sole means for implementing distributed mutual exclusion.
- The decision as to which process is allowed access to the CS next is arrived at by message passing, in which each process learns about the state of all other processes in some consistent way

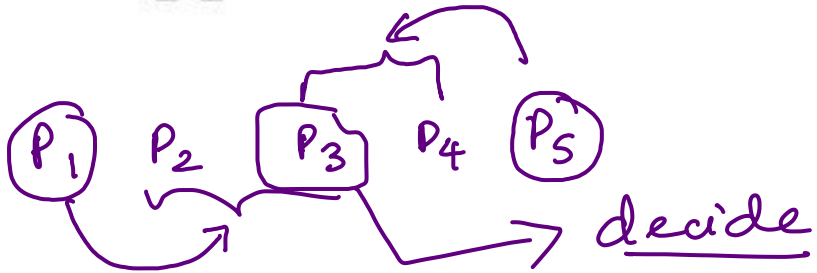
3 Approaches for Implementing Mutex

S_1 S_2 S_3 S_4 m

1. Token-based approach.
2. Non-token-based approach.
3. Quorum-based approach.

⇒ atm-card (privilege msg) (M)

⇒ Set of msgs
⇒ local variable



Key Points

- **Token Based Approach**
  - In the token-based approach, a unique token (also known as the PRIVILEGE message) is shared among the sites.
  - A site is allowed to enter its CS if it possesses the token and it continues to hold the token until the execution of the CS is over.
  - Mutual exclusion is ensured because the token is unique.
- **Non-token-based approach**
  - two or more successive rounds of messages are exchanged among the sites to determine which site will enter the CS next.
  - A site enters the critical section (CS) when an assertion, defined on its local variables, becomes true. Mutual exclusion is enforced because the assertion becomes true only at one site at any given time.
- **Quorum-based approach**
  - each site requests permission to execute the CS from a subset of sites (called a quorum).
  - The quorums are formed in such a way that when two sites concurrently request access to the CS, at least one site receives both the requests and this site is responsible to make sure that only one request executes the CS at any time.

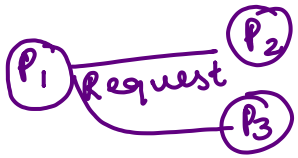
LIKE

COMMENT

SHARE

SUBSCRIBE

1. System Model



- The system consists of N sites, S_1, S_2, , S_N .with a single process is running on each site.
- The process at site S_i is denoted by p_i.
- All these processes communicate asynchronously over an underlying communication network.
- A process wishing to enter the CS requests all other or a subset of processes by sending REQUEST messages, and waits for appropriate replies before entering the CS.
- While waiting the process is not allowed to make further requests to enter the CS.
- **States for a Site:**
  - requesting the CS
  - executing the CS
  - or neither requesting nor executing the CS (i.e., idle).
- In the “requesting the CS” state, the site is blocked and cannot make further requests for the CS.
- In the “idle” state, the site is executing outside the CS.
- In the token-based algorithms, a site can also be in a state where a site holding the token is executing outside the CS. Such state is referred to as the idle token state.
- At any instant, a site may have several pending requests for CS. A site queues up these requests and serves them one at a time.
- the smaller the timestamp of a request, the higher its priority to execute the CS

2. Requirements for Algorithms

1. **Safety property** The safety property states that at any instant, only one process can execute the critical section. This is an essential property of a mutual exclusion algorithm.
2. **Liveness property** This property states the absence of deadlock and starvation. Two or more sites should not endlessly wait for messages that will never arrive. In addition, a site must not wait indefinitely to execute the CS while other sites are repeatedly executing the CS. That is, every requesting site should get an opportunity to execute the CS in finite time.
3. **Fairness** Fairness in the context of mutual exclusion means that each process gets a fair chance to execute the CS. In mutual exclusion algorithms, the fairness property generally means that the CS execution requests are executed in order of their arrival in the system (the time is determined by a logical clock).

3. Performance Metrics

Figure 9.1 Synchronization delay.

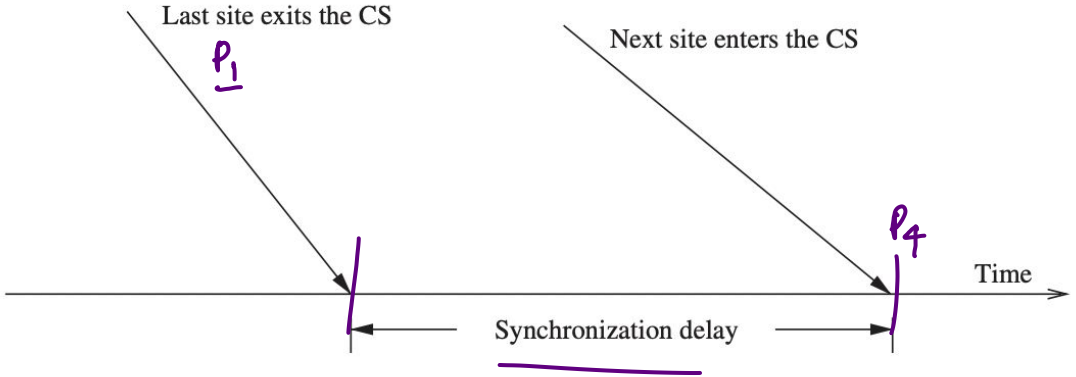
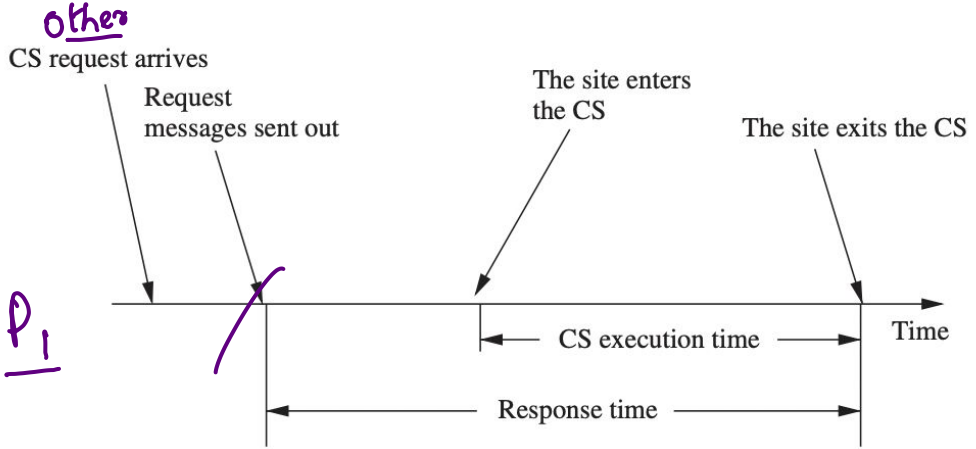


Figure 9.2 Response time.



The performance of mutual exclusion algorithms is generally measured by the following four metrics:

- **Message complexity** This is the number of messages that are required per CS execution by a site.
- **Synchronization delay** After a site leaves the CS, it is the time required and before the next site enters the CS (see Figure 9.1). Note that normally one or more sequential message exchanges may be required after a site exits the CS and before the next site can enter the CS.
- **Response time** This is the time interval a request waits for its CS execution to be over after its request messages have been sent out (see Figure 9.2). Thus, response time does not include the time a request waits at a site before its request messages have been sent out.
- **System throughput** This is the rate at which the system executes requests for the CS. If SD is the synchronization delay and E is the average critical section execution time, then the throughput is given by the following equation:

$$\text{System throughput} = \frac{1}{(SD + E)}.$$

Low and High Load Performance

- Under low load conditions, there is rarely more than one request for the critical section present in the system simultaneously.
- Under heavy load conditions, there is always a pending request for critical section at a site. Thus, in heavy load conditions, after having executed a request, a site immediately initiates activities to execute its next CS request.
- A site is seldom in the idle state in heavy load conditions

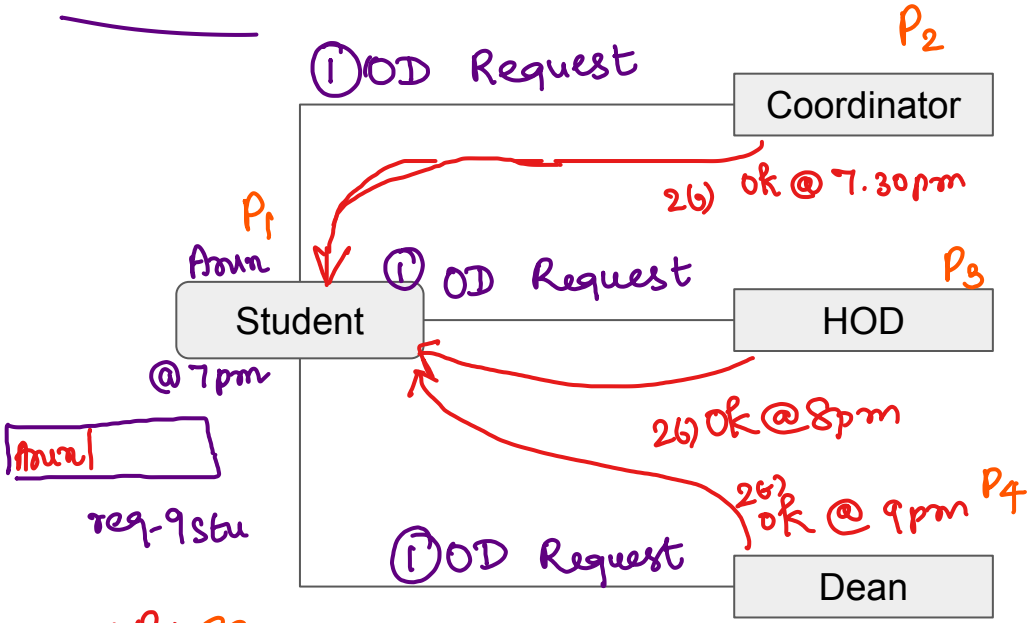
LIKE

COMMENT

SHARE

SUBSCRIBE

Lamport's Algorithm



G | A X

A | --- √√

OD approve CS enter

1) all ok timestamp > req-timestamp.

2) own queue ⇒ first

CS exit

1) Remove own queue entry.

2) RELEASE ⇒ from queue process is removed.

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.

Algorithm 9.1 Lamport's algorithm.

Figure 9.3 Sites S_1 and S_2 are Making Requests for the CS.

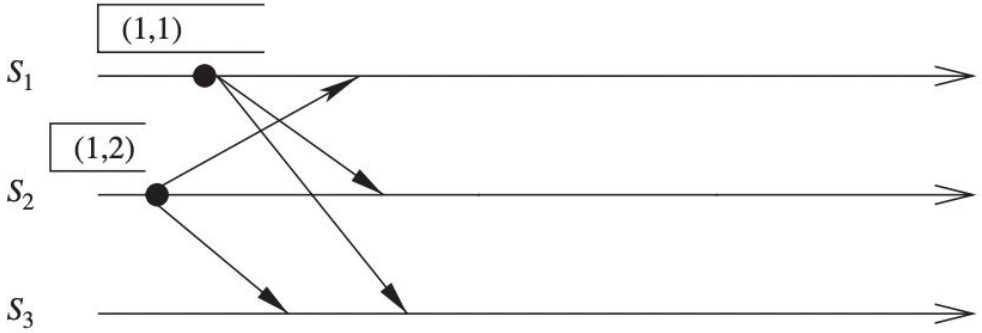


Figure 9.4 Site S_1 enters the CS.

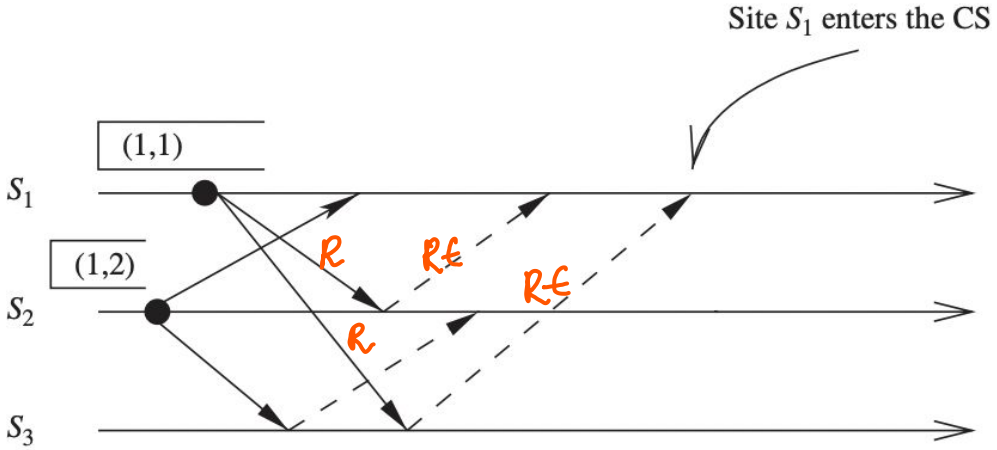


Figure 9.5 Site S_1 exits the CS and sends RELEASE messages.

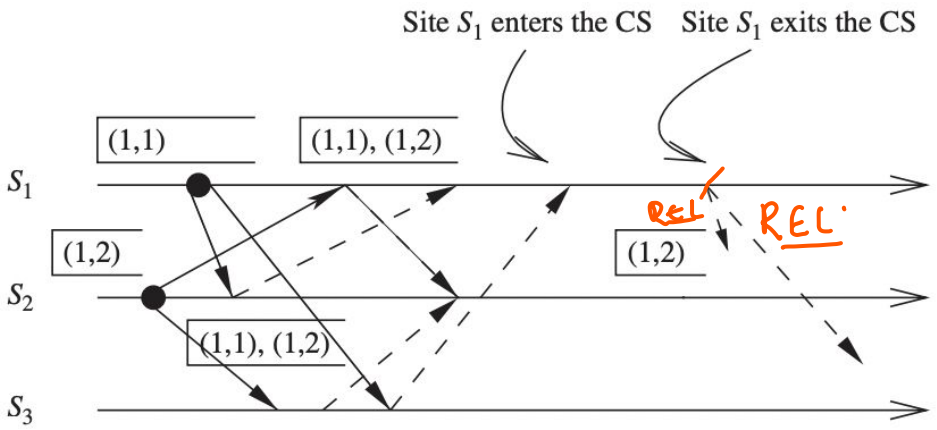
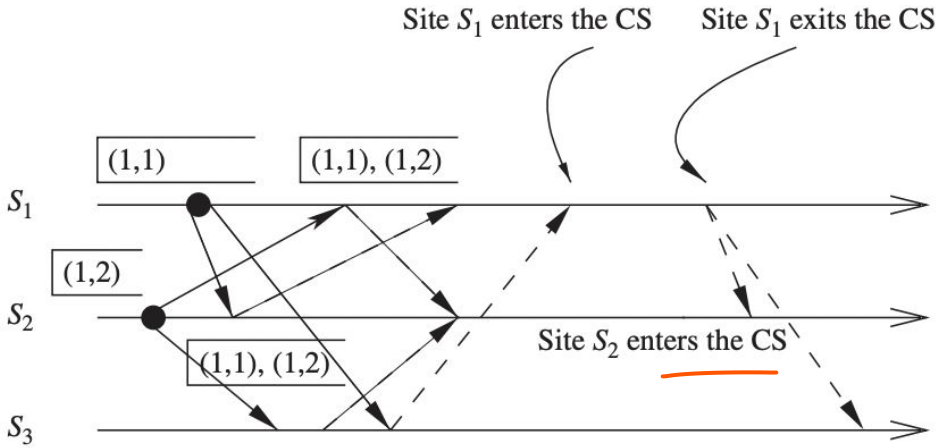


Figure 9.6 Site S_2 enters the CS.



Key Points

- The algorithm is fair in the sense that a request for CS are executed in the order of their timestamps and time is determined by logical clocks.
- When a site processes a request for the CS, it updates its local clock and assigns the request a timestamp.
- The algorithm executes CS requests in the increasing order of timestamps.
- Every site S_i keeps a queue, $request\_queue_i$, which contains mutual exclusion requests ordered by their timestamps. (Note that this queue is different from the queue that contains local requests for CS execution awaiting their turn.)
- This algorithm requires communication channels to deliver messages in FIFO order.
- 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. Clearly, when a site receives a REQUEST, REPLY, or RELEASE message, it updates its clock using the timestamp in the message

LIKE

COMMENT

SHARE

SUBSCRIBE

① Lamport's algorithm achieves mutual exclusion

Contradiction

① assume both P_1 and P_2 are in CS @ same time.

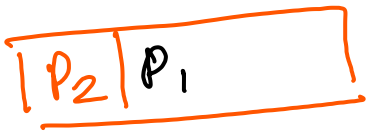
② Let $P_1 req_T < P_2 req_T$

$L_2 \rightarrow top$

Req-queue

P_1 :-

P_2 :-



X

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! Hence, Lamport's algorithm achieves mutual exclusion. □

S_j S_i
X

Lamport's algorithm is fair.

Lreq. executed in order of arrival.

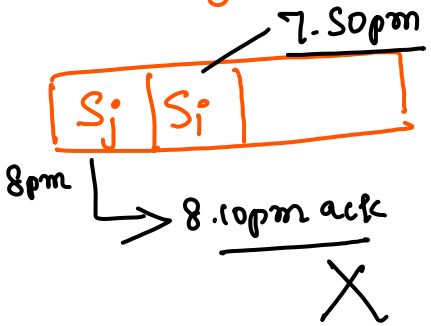
$S_{i_T} < S_{j_T}$

CS

S_j

Req queue

S_j



L1 → ack + time > S_{jreq}

Proof A distributed mutual exclusion algorithm is fair if the requests for CS are executed in the order of their timestamps. 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 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. Hence Lamport's algorithm is a fair mutual exclusion algorithm. □

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. The 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 a 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 a timestamp higher than its own, it can conclude that site S_j does not have any smaller timestamp request which is still pending (because communication channels preserves FIFO ordering).

With this optimization, Lamport's algorithm requires between $3(N-1)$ and $2(N-1)$ messages per CS execution.

LIKE

COMMENT

SHARE

SUBSCRIBE