LECTURE 6

DISTRIBUTED MUTUAL EXCLUSION

Prof. D. S. Yadav
Department of Computer Science
IET Lucknow

INTRODUCTION

- Mutual exclusion: Concurrent access of processes to a shared resource or data is executed in mutually exclusive manner.
- Only one process is allowed to execute the critical section (CS) at any given time.
- 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.

INTRODUCTION

Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.

Three basic approaches for distributed mutual exclusion:

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

- Token-based approach:
  - A unique token is shared among the sites.
  - A site is allowed to enter its CS if it possesses the token.
  - Mutual exclusion is ensured because the token is unique.

INTRODUCTION

- 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.

- Quorum based approach:
  - Each site requests permission to execute the CS from a subset of sites (called a quorum).
  - Any two quorums contain a common site.
  - This common site is responsible to make sure that only one request executes the CS at any time.

PRELIMINARIES

System Model

- The system consists of N sites, S_1, S_2, ..., S_N.
- We assume that a single process is running on each site. The process at site S_i is denoted by p_i.
- A site can be in one of the following three states:
  - 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 can not make further requests for the CS. In the 'idle' state, the site is executing outside the CS.
- In token-based algorithms, a site can also be in a state where a site holding the token is executing outside the CS (called 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.

REQUIREMENTS

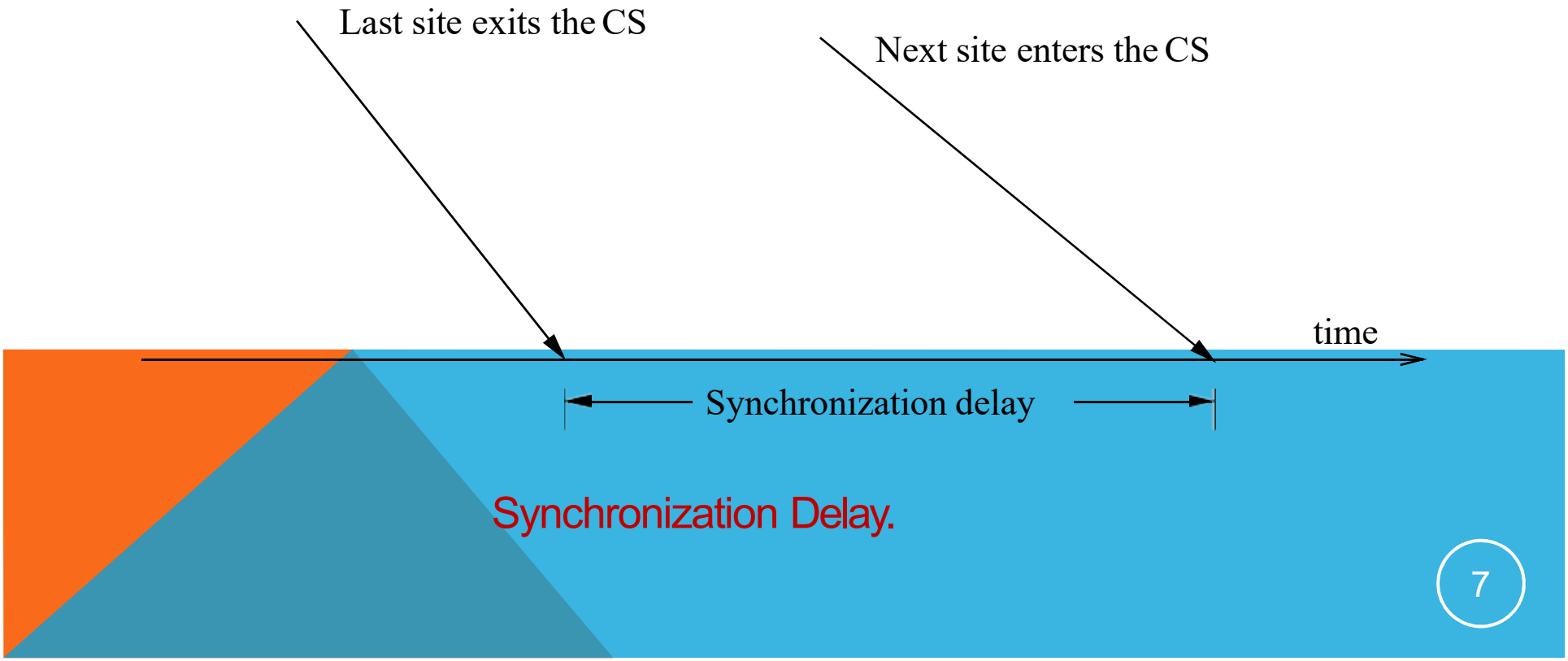
Requirements of Mutual Exclusion Algorithms

1. Safety Property: At any instant, only one process can execute the critical section.
2. Liveness Property: This property states the absence of deadlock and starvation. Two or more sites should not endlessly wait for messages which will never arrive.
3. Fairness: Each process gets a fair chance to execute the CS. Fairness property generally means the CS execution requests are executed in the order of their arrival (time is determined by a logical clock) in the system.

PERFORMANCE METRICS

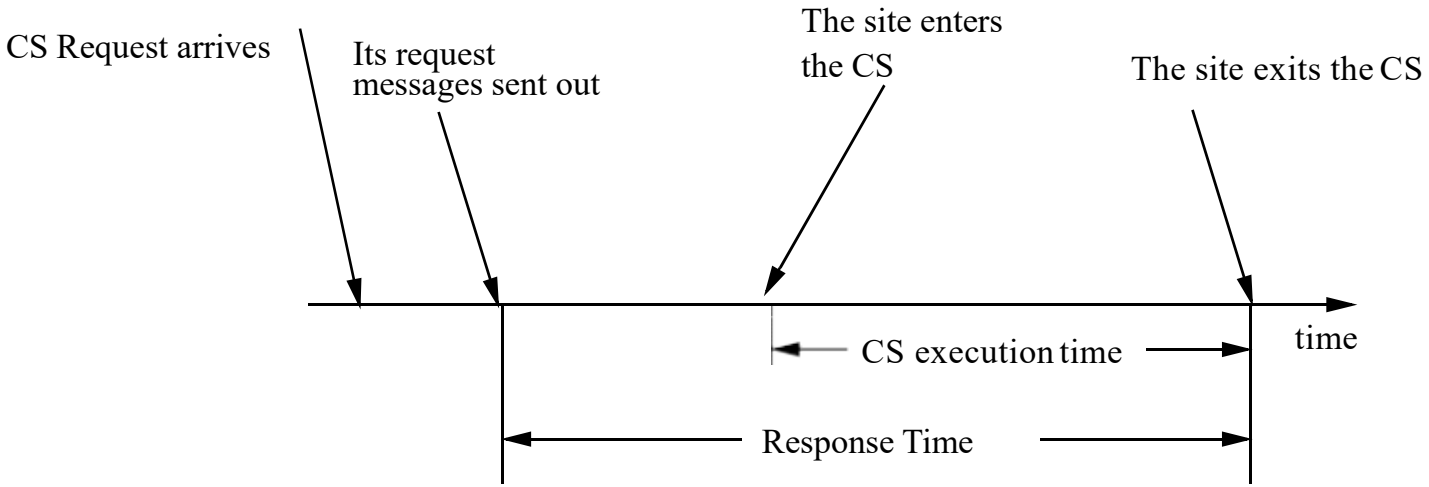
The performance is generally measured by the following four metrics:

- Message complexity: The number of messages 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.



PERFORMANCE METRICS

- **Response time:** The time interval a request waits for its CS execution to be over after its request messages have been sent out.



Response Time.

- **System throughput**: The rate at which the system executes requests for the CS.

  system throughput=1/(*SD*+*E*)

  where ***SD* is the synchronization delay** and ***E* is the average critical section execution time.**

PERFORMANCE METRICS

Low and High Load Performance:

- We often study the performance of mutual exclusion algorithms under two special loading conditions, viz., "low load" and "high load".
- The load is determined by the arrival rate of CS execution requests.
- Under *low load* conditions, there is seldom 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.

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.

END OF LECTURE 6