RAMAIAH
Institute of Technology

M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Course Name: Distributed Systems
Course Code: CSE20/CSE751
Credits: 3:0:0

Term: Oct 2021-Feb 2022

Faculty:
Sini Anna Alex

RAMAIAH Institute of Technology

Distributed Mutual Exclusion Algorithms

- 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 (see Figure 1).

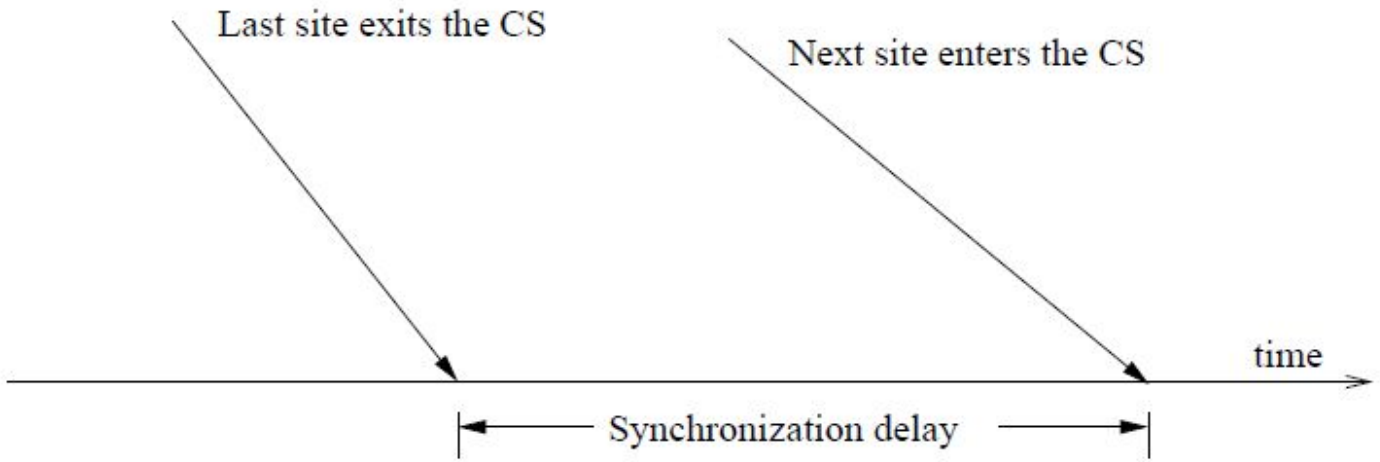

Figure 1: Synchronization Delay.

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 (see Figure 2).

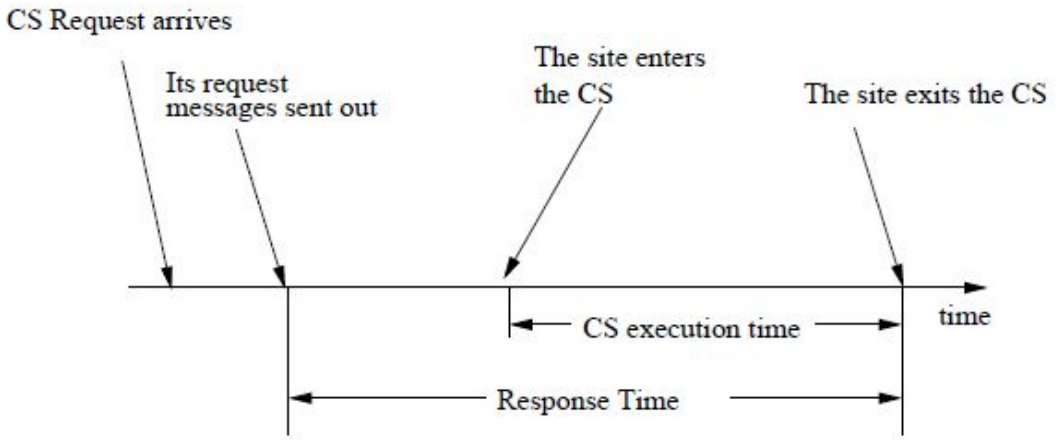

Figure 2: Response Time.

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

$$\text{system throughput} = 1/(SD+E)$$

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 Distributed Mutual Exclusion 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.

Lamport's Algorithm

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_j, 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.

Lamport's Algorithm

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 **$request\_queue_i$**

Lamport's Algorithm

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

Lamport's Algorithm

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.

Thank you