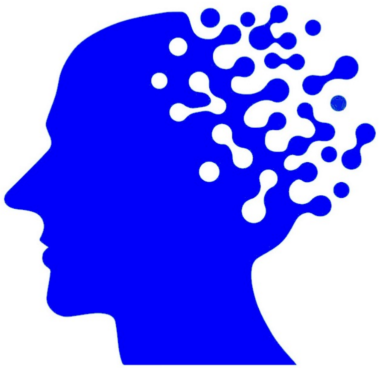


Spring 2023

Distributed Computing

- Basics of Mutual Exclusion algorithms

Dr. Rajendra Prasath

Indian Institute of Information Technology Sri City, Chittoor

21st March 2023 (http://rajendra.2power3.com)

> Distributed Computing?

- ➤ How will you design a Distributed Algorithm?

- ➤ Learn to Solve using Distributed Algorithms

Overview

Recap: Distributed Systems

A Distributed System:

- A collection of independent systems that appears to its users as a single coherent system
- A system in which hardware and software components of networked computers communicate and coordinate their activity only by passing messages
- A computing platform built with many computers that:
  - Operate concurrently
  - Are physically distributed (have their own failure modes)
  - Are linked by a network
  - Have independent clocks

Recap: Characteristics

- Concurrent execution of processes:
  - Non-determinism, Race Conditions, Synchronization, Deadlocks, and so on
- No global clock
  - Coordination is done by message exchange
  - No Single Global notion of the correct time
- No global state
  - No Process has a knowledge of the current global state of the system
- Units may fail independently
  - Network Faults may isolate computers that are still running
  - System Failures may not be immediately known

What did you learn so far?

- ➔ Goals / Challenges in Message Passing systems
- ➔ Distributed Sorting / Space-Time diagram
- ➔ Partial Ordering / Total Ordering
- ➔ Concurrent Events / Causal Ordering
- ➔ Logical Clocks vs Physical Clocks
- ➔ Global Snapshot Detection
- ➔ Termination Detection Algorithm
- ➔ Leader Election in Rings
- ➔ Topology Abstraction and Overlays
- ➔ Message Ordering and Group Communication

- ➔ Mutual Exclusion Algorithm

[Now] ➔ ➔ ➔

Recap

> About this Lecture

What do we learn today?

- ➤ Recap: Message Ordering and GC
  - ➤ Models of Communication
  - ➤ Design Issues / Good / Bad Ordering

- ➤ Mutual Exclusion Algorithms
  - ➤ Centralized Algorithm
  - ➤ Token-Based / Permission-Based Algorithms
  - ➤ Quorum-Based Algorithm
  - ➤ Tree-Based Algorithm

Let us explore these topics → → →

Today's Focus

Distributed Mutual Exclusion Algorithms

Why do we need MutEx?

- Mutual Exclusion
  - Operating systems: Semaphores
    - In a single machine, you could use semaphores to implement mutual exclusion
    - How to implement semaphores?
      - Inhibit interrupts
      - Use clever instructions (e.g. test-and-set)
    - On a multiprocessor shared memory machine, only the latter works

Characteristics

- Processes communicate only through messages – no shared memory or no global clocks
- Processes must expect unpredictable message delays
- Processes coordinate access to shared resources (printer, file, etc.) that should only be used in a mutually exclusive manner.

Race Conditions

- Consider Online systems – For example, Airline reservation systems maintain records of available seats
- Suppose two people buy the same seat, because each checks and finds the seat available, then each buys the seat
- Overlapped accesses generate different results than serial accesses – race condition

Distributed Mutual Exclusion

- Needs
  - Only one process should be in critical section at any point of time
  - What about resources?

Distributed Mutual Exclusion

- **No Deadlocks** – no set of sites should be permanently blocked, waiting for messages from other sites in that set
- **No starvation** – no site should have to wait indefinitely to enter its critical section, while other sites are executing the CS more than once
- **Fairness** - requests honored in the order they are made. This means processes have to be able to agree on the order of events. (Fairness prevents starvation.)
- **Fault Tolerance** – the algorithm is able to survive a failure at one or more sites

Distributed MutEx – An overview

Token-based solution: Processes share a special message known as a token

- ➔ Token holder has right to access shared resource
- ➔ Wait for/ask for (depending on algorithm) token; enter Critical Section (CS) when it is obtained, pass to another process on exit or hold until requested (depending on algorithm)
- ➔ If a process receives the token and doesn't need it, just pass it on

Distributed MutEx – A Few Issues

- Who can access the resource?
- When does a process to be privileged to access the resource?
- How long does a process access the resource? Any finite duration?
- How long can a process wait to be privileged?
- Computation complexity of the solution

Types of Distributed MutEx

- Token-based distributed mutual exclusion algorithms
  - Suzuki – Kasami's Algorithm
- Non-token based distributed mutual exclusion algorithms
  - Lamport's Algorithm
  - Ricart-Agartala's Algorithm

Token Based Methods

Advantages:

- Starvation can be avoided by efficient organization of the processes
- Deadlock is also avoidable

Disadvantage: Token Loss

- Must initiate a cooperative procedure to recreate the token
- Must ensure that only one token is created!

Permission-Based Methods

- **Non-Token Based solutions:** a process that wishes to access a shared resource must first get permission from one or more other processes.

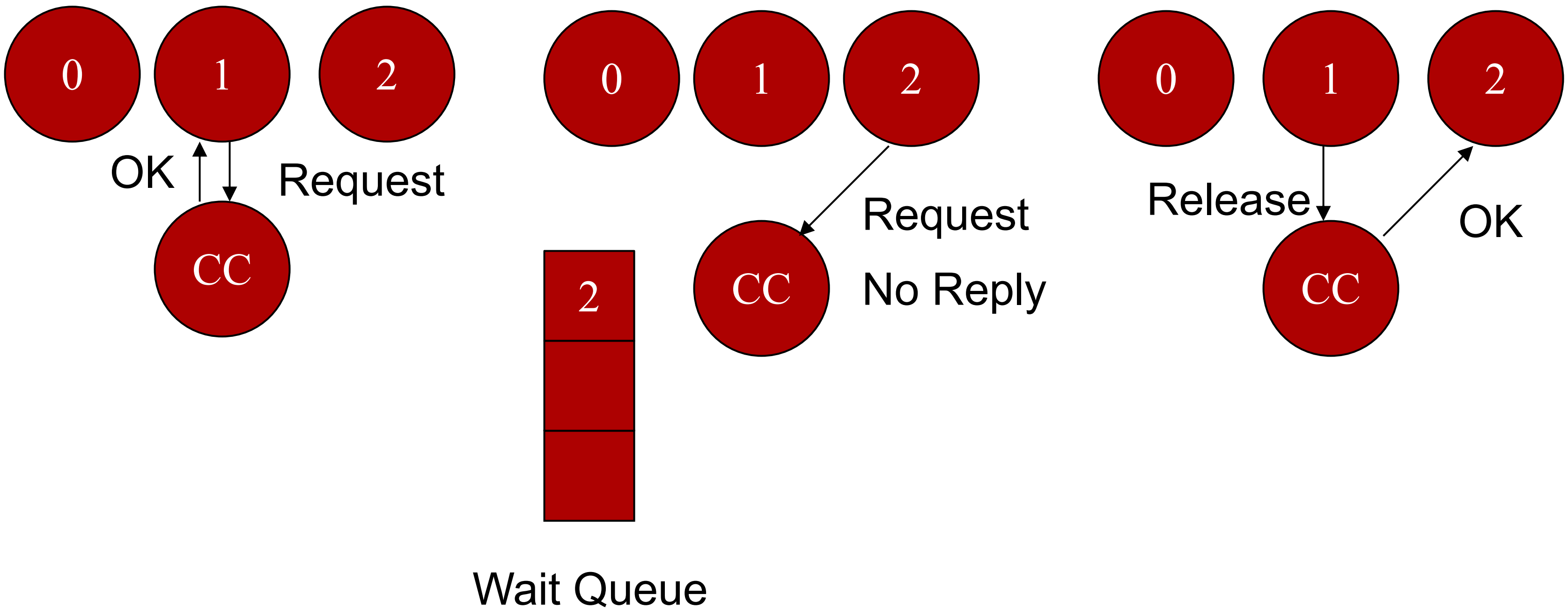
- Avoids the problems of token-based solutions, but is more complicated to implement

Basic Algorithms

- Centralized
- Decentralized
- Distributed
  - Distributed with “voting” – for increased fault tolerance
- Token Ring

Centralized Mutual Exclusion

➔ Central coordinator manages the FIFO queue of requests to guarantee "no starvation"



Centralized Control of MutEx

- A central coordinator (master or leader)
- Is elected (which algorithm?)
- Grants permission to enter CS & keeps a queue of requests to enter the CS.
- Ensures only one process at a time can access the CS
- Has a special token message, which it can give to any process to access CS

Centralized Control - Operations

- To enter a CS, send a request to the coordinator & wait for token.
- On exiting the CS, send a message to the coordinator to release the token.
- Upon receipt of a request, if no other process has the token, the coordinator replies with the token; otherwise, the coordinator queues the request
- Upon receipt of a release message, the coordinator removes the oldest entry in the queue (if any) and replies with a token

Centralized Control - Features

- Safety, Liveness are guaranteed
- Ordering also guaranteed (what kind?)
- Requires 2 messages for entry + 1 messages for exit operation.
- Client delay: one round trip time (request + grant)
- Synchronization delay: 2 message latencies (release + grant)
- The coordinator becomes performance bottleneck and single point of failure

Decentralized MutEx

- More fault-tolerant than centralized approach
- Uses the Distributed Hash Table (DHT) approach to locate objects/replicas
  - Object names are hashed to find the node where they are stored (succ function)
- n replicas of each object are placed on n successive nodes
  - Hash object name to get addresses
- Now every replica has a coordinator that controls access

The Decentralized Algorithm

- Coordinators respond to requests at once:

  Yes OR No

- Majority: To use the resource, a process must receive permission from m > n/2 coordinators
  - If the requester gets fewer than m votes, it will wait for a random time and then ask again
- If a request is denied, or when the CS is completed, notify the coordinators who have sent OK messages, so they can respond again to another request. (Why is this important?)

The Decentralized Algo - Analysis

- More robust than the central coordinator approach. If one coordinator goes down others are available.
  - If a coordinator fails and resets then it will not remember having granted access to one requestor, and may then give access to another.
  - It is highly unlikely that this will lead to a violation of mutual exclusion

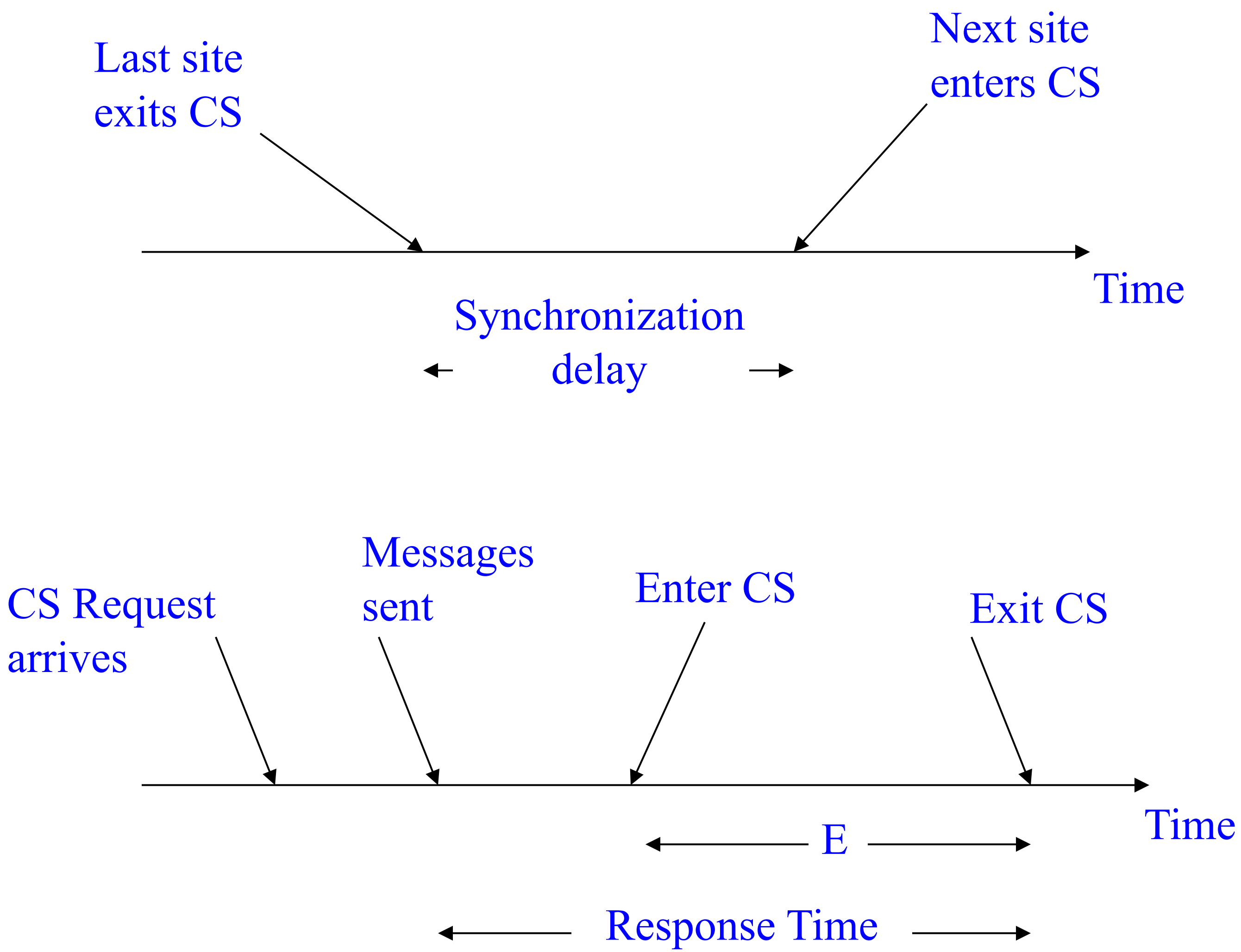
The Decentralized Algorithm - Issues

- If a resource is in high demand, multiple requests will be generated by different processes.
- High level of contention
- Processes may wait a long time to get permission - Possibility of starvation exists
- Resource usage drops.

Performance Analysis

- Guarantees mutual exclusion
- No starvation: Only if requests served in order
- No deadlock
- Fault tolerant?
  - Single point of failure
  - Blocking requests mean client processes have difficulty distinguishing crashed coordinator from long wait
  - Bottlenecks
- The solution is simple and ease

Performance Metrics



Performance - Analysis

- Number of messages per CS invocation: should be minimized
- Synchronization delay, i.e., time between the leaving of CS by a site and the entry of CS by the next one: should be minimized
- Response time: time interval between request messages transmissions and the exit of CS
- System throughput, i.e., rate at which system executes the requests for CS: should be maximized
- If d is the synchronization delay, e the average CS execution time:

$$\text{System Throughput} = 1 / (d + e)$$

Performance – Analysis (contd)

- Low and High Load:
  - Low load: No more than one request at a given point in time
  - High load: Always a pending mutual exclusion request at a site
- Best and Worst Case:
  - Best Case (low loads): Round-trip message delay + Execution time - 2T + E
  - Worst case (high loads)
- Message traffic: low at low loads, high at high loads
- Average performance: when load conditions fluctuate widely

Token Ring Approach

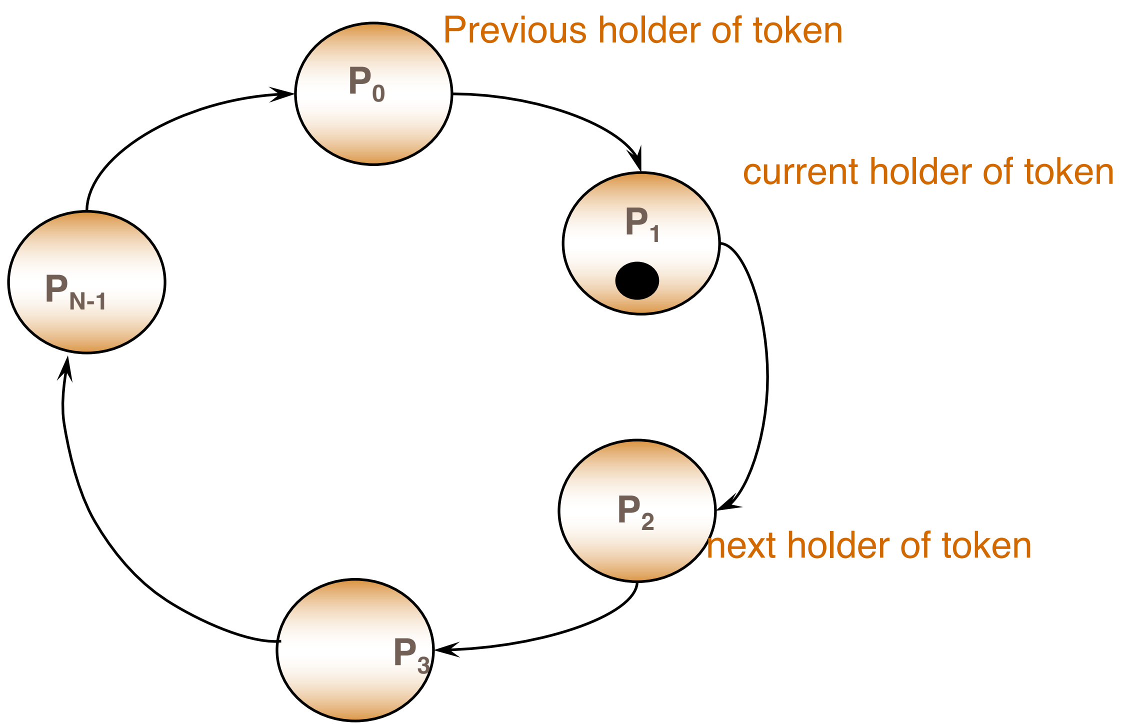
- Processes are organized in a logical ring: P_i has a communication channel to $P_{(i+1)} mod\ N$

Operations:

- Only the process holding the token T can enter the CS
- To enter the critical section, wait passively for T When in CS, hold on to T and don't release it
- To exit the CS, send T onto your neighbor
- If a process does not want to enter the CS when it receives T, it simply forwards T to the next neighbor

Token Rings – Illustration

- Request movements in an unidirectional ring network



Token Rings - Features

- Safety & Liveness are guaranteed
- Ordering is not guaranteed
- Bandwidth: 1 message per exit
- Client delay: 0 to N message transmissions
- Synchronization delay between one process's exit from the CS and the next process's entry is between 1 and N-1 message transmissions

Summary

- Racap: Message Ordering & Group Communications
  - Good / Bad Ordering
  - Causal Ordering

- Distributed Mutual Exclusion Algorithms
  - Mutual Exclusion Problem
    - Basics of MutEx algorithms
    - Types of MutEx algorithms
    - Centralized Approach
    - Token-based / Permission-based algorithms
    - Token RingsPerformance Metrics

Many more to come up … ! Stay tuned in !!

Penalties

- Every Student is expected to strictly follow a fair Academic Code of Conduct to avoid penalties

- Penalties is heavy for those who involve in:
  - Copy and Pasting the code
  - Plagiarism (copied from your neighbor or friend – in this case, both will get "0" marks for that specific take home assignments)
  - If the candidate is unable to explain his own solution, it would be considered as a "copied case"!!
  - Any other unfair means of completing the assignments

Help among Yourselves?

- **Perspective Students** (having CGPA above 8.5 and above)
- **Promising Students** (having CGPA above 6.5 and less than 8.5)

- **Needy Students** (having CGPA less than 6.5)
  - Can the above group help these students? (Your work will also be rewarded)

- You may grow a culture of **collaborative learning** by helping the needy students

How to reach me?

- Please leave me an email:

  rajendra [DOT] prasath [AT] iiits [DOT] in

- Visit my homepage @
  - https://www.iiits.ac.in/people/regular-faculty/dr-rajendra-prasath/

(OR)

  - http://rajendra.2power3.com

Assistance

- You may post your questions to me at any time
- You may meet me in person on available time or with an appointment
- You may ask for one-to-one meeting

Best Approach

- You may leave me an email any time
  (email is the best way to reach me faster)

Questions
It's Your Time

Contact Information

Dr. Rajendra Prasath
IIIT Sri City, Chittoor