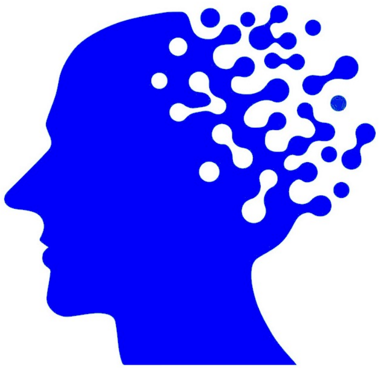


Spring 2023

Distributed Computing

- Token-Based Mutual Exclusion algos

Dr. Rajendra Prasath

Indian Institute of Information Technology Sri City, Chittoor

23rd 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?

- 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 – Token-Based MutEx Algorithms

Recap: The need for 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

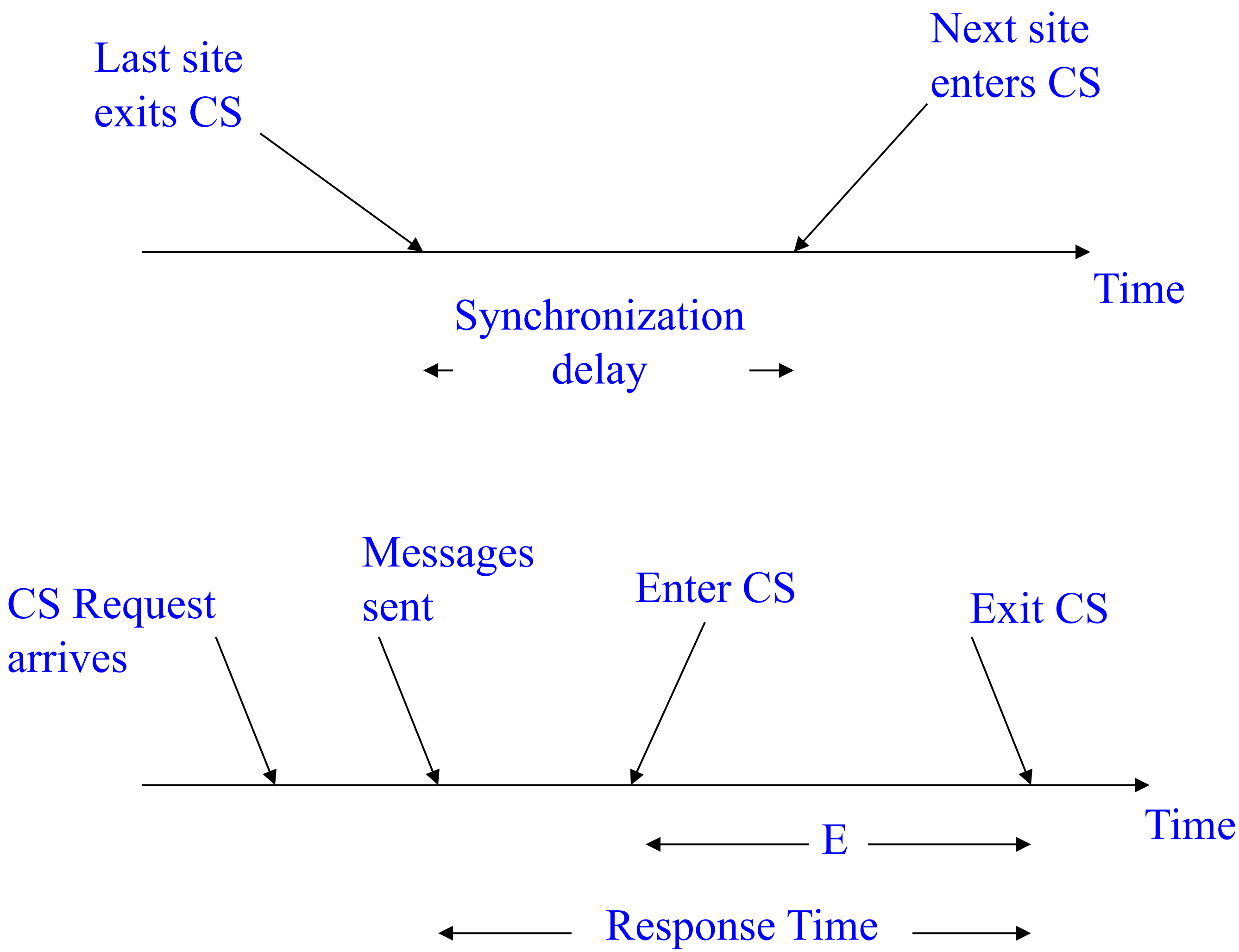
Characteristics

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

Recap: Distributed MutEx

- **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

Recap: Performance Metrics



Token-Based MutEx Algorithms

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
  - Feuerstein et al's Algorithm

- Non-token based distributed mutual exclusion algorithms
  - Lamport's Algorithm
  - Ricart-Agrawala'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!

Token Ring Approach

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

Operations:

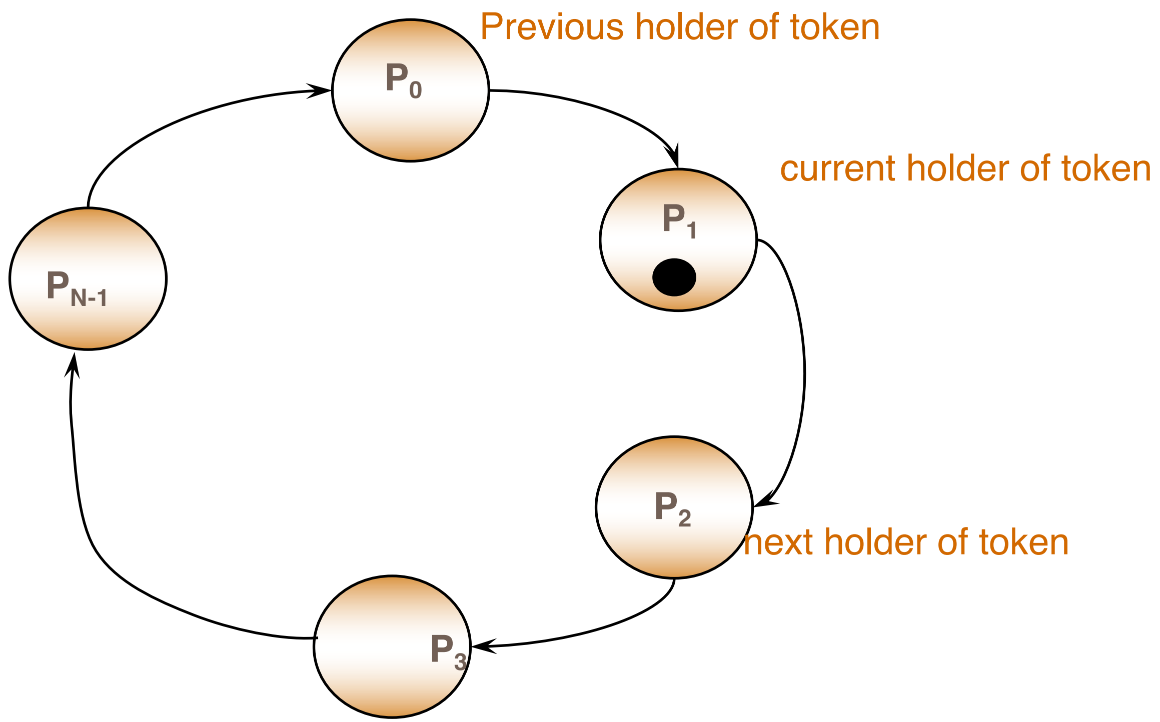
- Only the process holding the token T can enter the CS
- To enter the critical section, wait passively for T When in CS, hold on to T and don't release it
- To exit the CS, send T onto your neighbor
- If a process does not want to enter the CS when it receives T, it simply forwards T to the next neighbor

Token Ring - Idea

- Single token circulates, enter CS when token is present
- Mutual exclusion obvious
- Algorithms differ in how to find and get the token
- Uses sequence numbers rather than timestamps to differentiate between old and current requests

Token Rings – Illustration

- Request movements in an unidirectional ring network



Suzuki – Kasami's Algorithm

- **Broadcast** a request for the token
- Process with the **token sends** it to the requestor if it does **not need** it

- Issues:
  - Current versus outdated requests
  - Determining sites with pending requests
  - Deciding which site to give the token to

Data Structures

The token:

- ➔ Queue (FIFO) Q of requesting processes
- ➔ LN[1.. n] : sequence number of request that j executed most recently

The request message:

- ➔ REQUEST(i, k): request message from node i for its k^{th} critical section execution

Other data structures:

- ➔ RN_i [1.. n] for each node i, where , RN_i [j] is the largest sequence number received so far by i in a REQUEST message from j

Suzuki-Kasami's algorithm

To request critical section:

- If i does not have token, increment RN_i[i] and send REQUEST(i, RN_i[i]) to all nodes
- If i has token already, enter critical section if the token is idle (no pending requests), else follow rule to release critical section

On receiving REQUEST(i, s_n) at) j:

- Set RN_j[i] = max(RN_j[i], s_n)
- If j has the token and the token is idle then
  - send it to i if RN_j [i] = LN[i] + 1
  - If token is not idle, follow rule to release critical section

Suzuki-Kasami's algorithm

To enter critical section:

- ➔ Enter CS if token is present

To release critical section:

- ➔ Set $LN[\ i\ ] = RN_i[\ i\ ]$
- ➔ For every node j which is not in Q (in token), add node j to Q if $RN_i[\ j\ ] = LN[\ j\ ] + 1$
- ➔ If Q is non empty after the above, delete first node from Q and send the token to that node

Complexity

- No. of messages:
  - 0 if node holds the token already,
  - n otherwise

- Synchronization delay:
  - 0 (node has the token) or
  - max. message delay (token is elsewhere)

- No starvation

Token Based Control in Rings

- Requests move in either Clockwise or Anticlockwise
- Proposed by Feuerstein et al. (1996)
- There are 3 steps:
  - P needs the resource
    - P has T: enter CS
    - P has no T: send the request to the next P
  - P receives a request
    - P has T: increase TC by 1 and send T to the next P
    - P has no T: send request to the next P
  - P receives Token
    - P has a pending request: enter CS and decrease TC by 1 and send T to next P if TC > 0
    - P has no pending request: send T to the next P

Refer to: Feuerstein et al., Efficient token-based control in rings, Information Processing Letters, 66 (4) (1998) pp. 175-180

Token Based Control In Rings

- When p needs the resource then
  - If p has T then it enters the critical section.
  - If p has not T then it sends a request message to the next processor
- When p receives a request message then:
  - If p has T then it increases the counter and passes T to the next processor.
  - If p has not T then the request message is passed to the next processor.
- When p receives T then:
  - If p has a pending request then it enters the critical section in which the token counter is decreased by 1; at the exit of the critical section p passes T to the next processor if the token counter is greater than 0
  - If p has not a pending request then T is passed to the next processor in the ring.

Read from: E. Feuerstein et al. Efficient token-based control in rings, Information Processing Letters, 66 (1998) pp. 175-180, doi: 10.1016/S0020-0190(98)00054-4

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

Non-Token Based Methods

- **Permission-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

Non-Token Based Algorithms

- **Notations:**
  - P_i: i th Process
  - R_i: Request set, containing IDs of all P_i s from which permission must be received before accessing CS
  - Non-token based approaches use time stamps to order requests for CS
  - Smaller time stamps get priority over larger ones
- **Lamport's Algorithm**
  - R_i = $\{P_1, P_2, \ldots, P_n\}$, i.e., all processes.
  - Request queue: maintained at each P_i ordered by time stamps.
  - Assumption: message delivered in FIFO

Lamport's Algorithm

- **Requesting CS:**
  - Send REQUEST(ts_i, i) where (ts_i, i) - Request time stamp; Place REQUEST in $request\_queue_i$
  - On receiving the message; P_j sends time-stamped REPLY message to P_i; Pi 's request placed in $request\_queue_j$
- **Executing CS:**
  - P_i has received a message with time stamp larger than (ts_i, i) from all other sites
  - P_i's request is the top most one in $request\_queue_i$
- **Releasing CS:**
  - Exiting CS: send a time stamped RELEASE message to all sites in its request set
  - Receiving RELEASE message: P_j removes P_i 's request from its queue

Notable Points

- Purpose of REPLY messages from i to j is to ensure that j knows of all requests of i prior to sending the REPLY (possibly any request of i with timestamp lower than j 's request)
- Requires FIFO channels
- 3(n – 1) messages per critical section invocation
- Synchronization delay = max msg transmission time
- Requests are granted in order of increasing timestamps

Performance Improvements

- 3(n-1) messages per Critical Section invocation
  - (n - 1) REQUEST messages
  - (n - 1) REPLY messages
  - (n - 1) RELEASE messages
- Synchronization delay: T
- Optimization:
  - Suppress reply messages: For example, P_j receives a REQUEST message from P_i after sending its own REQUEST message with time stamp higher than that of P_i's then Do NOT send a REPLY message
  - Messages reduced to between 2(n-1) and 3(n-1)

Ricart & Agrawala's Algorithm

- A time-stamp based approach
- Originally proposed by Lamport using logical clocks
- Modified by Ricart & Agrawala

Ricart & Agrawala's Algorithm

Main Idea:

- Process j need not send a REPLY to Process i if j has a request with timestamp lower than the request of i (since i cannot enter before j here)
- Does not require FIFO
- 2(n - 1) messages per critical section invocation
- Synchronization delay = maximum message transmission time
- Requests granted in order of increasing timestamps

Ricart & Agrawala (contd)

- Processes need entry to critical section multicast a request, and can enter it only when all other processes have replied positively
- Messages requesting entry are of the form $<T, P_i>$
  - T - sender's timestamp (Lamport clock)
  - P_i the sender's identity

Ricart & Agrawala – Algorithm

To enter the Critical Section (CS):

- Set state = wanted
- multicast "request" to all processes (including timestamp)
- wait until all processes send back "reply"
- change state to held and enter the CS

On receipt of a request <T_j, P_j> at P_i:

- if (state == held) or (state == wanted & $(T_i, P_i) < (T_j, P_j)$) then enqueue the request
- else "reply" to P_j

On exiting the CS:

- change state to release and "reply" to all queued requests

Ricart & Agrawala – Simplified

To request Critical Section:

- ➔ send timestamped REQUEST message (ts_i, i)

On receiving request (ts_i, i) at j:

- ➔ if j is neither requesting nor executing critical section then send REPLY to i
- ➔ if j is requesting and i's request timestamp is smaller than j's request timestamp then
  - ➔ enqueue the request; Otherwise, defer the request

To enter Critical Section:

- ➔ Process i enters critical section on receiving REPLY messages from all processes

To release Critical Section:

- ➔ send REPLY to all deferred requests

Summary

- Recap: Distributed Mutual Exclusion Algorithms
  - Mutual Exclusion Problem
    - Basics of MutEx algorithms
    - Types of MutEx algorithms
      - Token-based Algorithm
        - Suzuki-Kasami's Algorithm
        - Feuerstein et al's Algorithm
      - Non-Token based Algorithm
        - Lamport's Algorithm
        - Ricart – Agrawala's Algorithm
    - Performance 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