Distributed Mutual Exclusion Algorithms

Course: Distributed Computing

Faculty: Dr. Rajendra Prasath

About this topic

This course covers various concepts in **Mutual Exclusion in Distributed Systems.** We will also focus on different types of distributed mutual exclusion algorithms in distributed contexts and their analysis

What did you learn so far?

Recap

- → Challenges in Message Passing systems
- → Distributed Sorting
- → Space-Time Diagram
- → Partial Ordering / Causal Ordering
- → Concurrent Events
- → Local Clocks and Vector Clocks
- → Distributed Snapshots
- → Termination Detection
- → Topology Abstraction and Overlays
- → Leader Election Problem in Rings
- → Message Ordering / Group Communications

Recent Topic …

- Communication Models
- Design Issues
  - Process Failures
- Message Ordering
  - Good / Bad ordering
  - Various Types of Ordering of messages
- Group Communication
  - Causal ordering based approach

- Many more to come up … stay tuned in !!

Topics to focus on …

- Leader Election in Distributed Systems
- Topology Abstraction and Overlays
- Message Ordering
- Group Communication

- **Distributed Mutual Exclusion**

- Deadlock Detection
- Check pointing and rollback recovery

For End Semester

Mutual Exclusion in Distributed Systems

Let us explore MutEx algorithms proposed for various interconnection networks

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-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!

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

Token Ring Approach

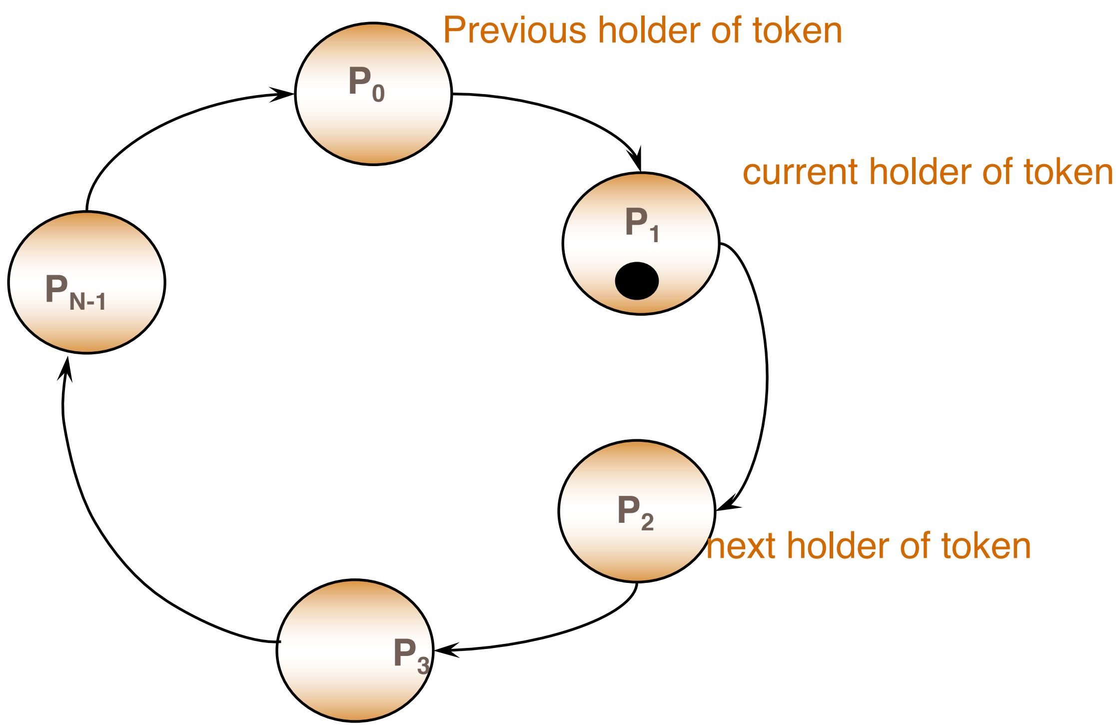
- Processes are organized in a logical ring: P_i has a communication channel to $P_{(i+1)} mod\ N$

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 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 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 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, ..., 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

- ➔ Mutual Exclusion Problem
- ➔ Basics of MutEx algorithms

- ➔ Various Types of MutEx algorithms
  - ➔ Token-based
    - ➔ Token rings
  - ➔ Non-Token based algorithm
    - ➔ Lamport's Algorithm
    - ➔ Ricart – Agrawala's Algorithm
- ➔ Performance Metrics

➔ Many more to come up … stay tuned in !!

How to reach me?

- Please leave me an email:
  rajendra [DOT] prasath [AT] iiits [DOT] in

- Visit my homepage @

  - http://www.iiits.ac.in/FacPages/index-rajendra.html

  OR

  - http://rajendra.2power3.com

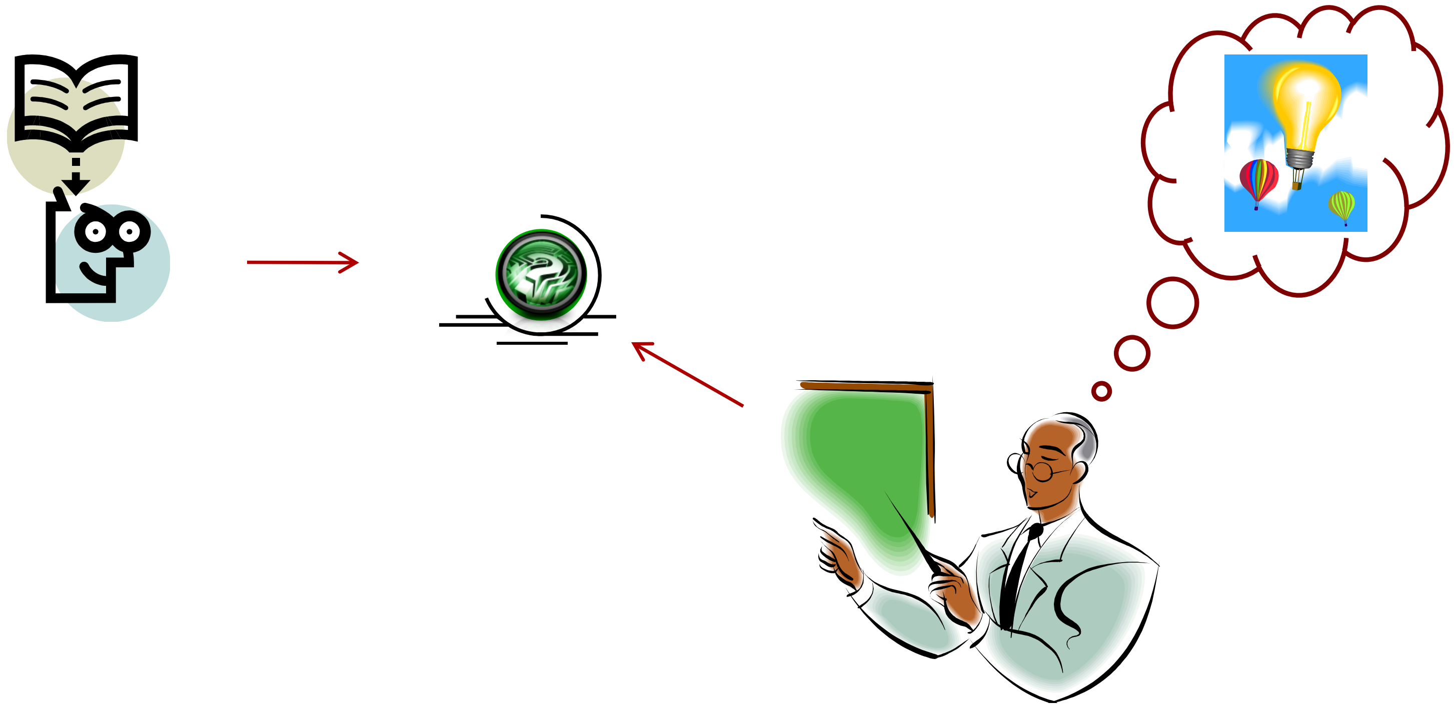
Help among Yourselves?

- **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

Thanks …



… Questions ???