Distributed Mutual Exclusion

Mutual Exclusion

- Very well-understood in shared memory systems

- Requirements:
  - at most one process in critical section (safety)
  - if more than one requesting process, someone enters (liveness)
  - a requesting process enters within a finite time (no starvation)
  - requests are granted in order (fairness)

Types of Dist. Mutual Exclusion Algorithms

- Non-token based / Permission based
  - Permission from all processes: e.g. Lamport, Ricart-Agarwala, Raicourol-Carvalho etc.
  - Permission from a subset: ex. Maekawa

- Token based ex. Suzuki-Kasami
  - Single token in the system
  - Node enters critical section if it has the token
  - Algorithms differ in how the token is circulated

Some Complexity Measures

- No. of messages/critical section entry
- Synchronization delay
- Response time
- Throughput

Lamport's Algorithm

- Every node i has a request queue q_i
  - keeps requests sorted by logical timestamps (total ordering enforced by including process id in the timestamps)

- To request critical section:
  - send timestamped REQUEST(tsi, i) to all other nodes
  - put (tsi, i) in its own queue

- On receiving a request (tsi, i):
  - send timestamped REPLY to the requesting node i
  - put request (tsi, i) in the queue

Lamport's Algorithm contd..

- To enter critical section:
  - Process i enters critical section if:
    - (tsi, i) is at the top if its own queue, and
    - Process i has received a message (any message) with timestamp larger than (tsi, i) from ALL other nodes.
- To release critical section:
    - Process i removes its request from its own queue and sends a timestamped RELEASE message to all other nodes
    - On receiving a RELEASE message from i, i's request is removed from the local request queue

Some notable points

- Purpose of REPLY messages from node i to j is to ensure that j knows of all requests of i prior to sending the REPLY (and therefore, 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 mesg transmission time
- Requests are granted in order of increasing timestamps

The Ricart-Agrawala Algorithm

- Improvement over Lamport's
- Main Idea:
  - node j need not send a REPLY to node i if j has a request with timestamp lower than the request of i (since i cannot enter before j anyway in this case)
- Does not require FIFO
- 2(n – 1) messages per critical section invocation
- Synchronization delay = max. message transmission time
- Requests granted in order of increasing timestamps

The Ricart-Agrawala Algorithm

- To request critical section:
  - send timestamped REQUEST message (tsi, i)

- On receiving request (tsi, i) at j:
  - send REPLY to i if j is neither requesting nor executing critical section or
  - if j is requesting and i's request timestamp is smaller than j's request timestamp. Otherwise, defer the request.

- To enter critical section:
  - i enters critical section on receiving REPLY from all nodes

- To release critical section:
  - send REPLY to all deferred requests

Roucairol-Carvalho Algorithm

- Improvement over Ricart-Agarwala
- Main idea
  - Once i has received a REPLY from j, it does not need to send a REQUEST to j again unless it sends a REPLY to j (in response to a REQUEST from j)
  - Message complexity varies between 0 and $2(n - 1)$ depending on the request pattern
  - worst case message complexity still the same