Distributed Mutual Exclusion

- Distributed mutual exclusion (DME) coordinates software on different computers so that they agree upon assigning a resource or section of code to one particular task.

Requirements

- Mutual Exclusion
- Freedom from deadlock
- Eventual entry (freedom from starvation)
- *All processes must participate equally*.
  Only interested processes participate.

Assumptions

- N nodes randomly request access.
- Messages are not lost or corrupted.
- Message transmissions take a finite, variable, non-zero time.
- Messages arrive in order.
- Transmission time might not be transitive.
- Network is fully connected.

Mutual Exclusion Goals

- Minimize the number of messages sent.
- Grant permission in order of request.
- Fault tolerant
- Fair to all systems
- Scalable

Distributed Mutual Exclusion Algorithms

- Assertion Based
  - Lamport algorithm
  - Ricart-Agrawala algorithm
  - Maekawa algorithm
- Token Based
  - Raymond's Tree-based algorithm
  - Simple 2 process algorithm

Lamport's DME Algorithm

- Processes are granted mutual exclusion in the order in which they make the request. Each process maintains a request queue sorted in timestamp order.
- Assertion based algorithm.
- Uses the Lamport time numbers.

Lamport Algorithm (cont)

1. To request a resource, process *Pi* sends a timestamped request message to every other process and puts the request on its own request queue.
2. When process *Pk* receives a request message, it sends a timestamped reply and puts the request on its request queue.

Lamport Algorithm (cont)

3. Process *Pi* can access the resource when:

- Its own request is at the top of the request queue.
- It has received a reply from every other process.

4. To release a resource, *Pi* removes its request from its queue and sends a release message to all other processes.

Lamport Algorithm (cont)

5. When process *Pk* receives a release message, it removes *Pi*”s request from its queue.

Ricart and Agrawala's Algorithm

- Similar to Lamport's Algorithm but slightly more efficient.

1. To request a resource, process *Pi* sends a time stamped request message to every other process.

Ricart and Agrawala's Algorithm *(cont.)*

2. When process *Pk* receives a request msg

if *Pk* is not currently requesting the resource

    it sends a time stamped reply

else

    if timestamp of *Pi* request is <

                the timestamp of *Pk*'s request

            it sends a timestamped reply

    else

            it keeps *Pi*'s request.

Ricart and Agrawala's Algorithm *(cont.)*

3. Process *Pi* has received a reply from every other process, it can access the resource.
4. To release a resource, *Pi* returns a reply message to all pending processes.

Simple Two Process Algorithm

- Assume only two processes are competing for a single resource.
- The two processes communicate by message passing.