Mutual Exclusion

• Processes in a distributed system may need to simultaneously access the same resource
• Mutual exclusion is required to prevent interference and ensure consistency
• We will study three algorithms for mutual exclusion:
  – A centralized algorithm
  – A distributed algorithm
  – A token ring algorithm

A Centralized Algorithm

• One process is elected as the coordinator
• Every process needs to get permission from the coordinator before entering the critical section
  – To obtain exclusive access: sends request message, awaits OK message
  – To release: sends release message
• Coordinator maintains a queue of waiting processes
  – Upon receiving request message: if resource is available, sends OK message; if not, queues the request
  – Upon receiving release message: removes the oldest request in the queue and sends OK message
An Example

(a) Process 1 asks the coordinator for permission to access a shared resource. Permission is granted. b) Process 2 then asks permission to access the same resource. The coordinator does not reply. c) When process 1 releases the resource, it tells the coordinator, which then replies to 2.

Properties of the Centralized Algorithm (1)

- Fair – requests are granted in the order in which they are received
- No starvation – no process ever waits forever
- Simple – three messages per use of a resource (request, OK, release)
- Drawbacks
  - The coordinator is a single point of failure – must elect a new coordinator if the current coordinator fails
  - The coordinator can be a performance bottleneck in a large system
An Improvement to the Centralized Algorithm

• Processes can not distinguish a dead coordinator from “permission denied” – No response from the coordinator in either case
• Solution: Upon receiving a request, the coordinator always sends a reply, either granting or denying permission
  – This enables the requester to detect dead coordinator
  – After a request is denied, the sender should block waiting for a subsequent OK message

A Distributed Algorithm (1)

• Developed by Ricart and Agrawala [1981]
• The algorithm implements mutual exclusion between a set of peer processes
  – A process can enter the critical section only when it has got permission from all other processes
• Each process maintains a logical clock
• Process $i$ enters critical section as follows
  1. Multicasts a request message $(L_i, i)$ to all other processes
     $(L_i$ is process $i$’s current logical time)
  2. Waits until a reply is received from every other process
  3. Enters critical section
A Distributed Algorithm (2)

- When process $j$ receives a request message
  - If outside the critical section, sends an OK message
  - If in the critical section, does not reply and queues the request
  - If has sent a request to enter the critical section, sends an OK message if $(L_i, i) < (L_j, j)$, else queues the request
- When a process exits the critical section, it sends OK messages to all processes in its queue and remove these processes from the queue
- If two or more processes request entry at the same time, the process whose request bears the lowest timestamp (process ID used for tie breaking) will be the first to enter the critical section

An Example

a) Two processes want to access a shared resource at the same time.
b) Process 0 has the lowest timestamp, so it wins.
c) When process 0 is done, it sends OK to process 2, so process 2 can now go ahead.
Properties of the Distributed Algorithm

- Fully distributed: no coordinator needed
- No starvation
- All processes are involved in all decisions
  - N points of failure!
  - Any overloaded process can become a bottleneck
- Improvements
  - Upon receiving a request, a process always sends a reply, either granting or denying permission – this enables the detection of a dead process
  - A process can enter the critical section when it has got permission from a simple majority of the other processes

A Token Ring Algorithm

- Processes are arranged in a logical ring
- A token is passed from process to process around the ring
- A process must obtain the token before entering the critical section; it passes the token to its neighbor when it exits the critical section
- A process passes the token to its neighbor if it does not require to enter the critical section

(a) An unordered group of processes on a network. (b) a logical ring constructed in software.
Properties of the Token Ring Algorithm

- Fully distributed: no coordinator needed
- No starvation
- The algorithm continuously consumes network bandwidth
  - The token circulates around the ring even when no process requires entry to the critical section
- If the token is lost, it must be regenerated
  - How to detect token loss?
- A dead process can break the ring
  - How to detect a dead process?

Comparison of Mutual Exclusion Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Messages per entry/exit</th>
<th>Delay before entry (in message times)</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td>3</td>
<td>2</td>
<td>Coordinator crash</td>
</tr>
<tr>
<td>Distributed</td>
<td>2(n-1)</td>
<td>2(n-1)</td>
<td>Crash of any process</td>
</tr>
<tr>
<td>Token ring</td>
<td>1 to infinity</td>
<td>0 to n-1</td>
<td>Lost token, process crash</td>
</tr>
</tbody>
</table>