Mutual Exclusion

- Processes in a distributed system may need to simultaneously access the same resource
- Need to grant mutual exclusive access to shared resources by processes
- Solutions:
  - Via a centralized server
  - Decentralized, using a peer-to-peer system
  - Distributed, with no topology imposed
  - Distributed, along a logical ring

Centralized Mutual Exclusion

- Assume processes are numbered
- One process is elected coordinator
- Every process needs to check with coordinator before entering the critical section
- To obtain exclusive access: send request, await reply
- To release: send release message
- Coordinator:
  - Receive request: if resource is available and queue empty, send OK; if not, queue request
  - Receive release: remove next request from queue and send OK
Centralized Mutual Exclusion

(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 Centralized Mutual Exclusion

- Simulate centralized locking with blocking calls
- Advantages
  - Fair: requests are granted in the order they were received
  - Simple: three messages per use of a resource (request, OK, release)
  - No starvation
- Drawbacks:
  - Single point of failure
  - Performance bottleneck in large distributed systems
  - How do you detect a dead coordinator?
    - A process can not distinguish between “permission denied” from a dead coordinator – No response from coordinator in either case
A Decentralized Algorithm

• Each resource is replicated n times. Each replica has its own coordinator
• Access requires majority vote from m > n/2 coordinators.
  – Nonblocking: coordinators return OK or “no”
• Coordinator crashes => forgets previous votes (i.e., resets itself)
  – Probability that k coordinators reset p(k) = C(m,k)p^k (1-p)^m-k
  • p = Δt/T is the probability that a coordinator crashes and recovers in a period Δt while having an average lifetime T
  – At least 2m-n coordinators need to reset to violate correctness
    • Probability of such a violation is
      \[ \sum_{k=2m-n}^{m} p(k) \]
    • With p=0.001, n=32, m=0.75n, probability of violation is less than 10^{-40}

• If request is denied, process will back off for a randomly-chosen time, and try again
• Drawbacks
  – Low resource utilization when many nodes want to access the same resource
  – Starvation can occur
A Distributed Algorithm

- Based on event ordering and time stamps
  - Each process maintains a logical clock
- Process $k$ enters critical section as follows
  - Increment logical clock: $L_k = L_k + 1$
  - Multicast a message $(L_k, k)$ to all other processes
  - Wait until a reply is received from every other process
  - Enter critical section
- Upon receiving a request message, process $j$
  - Sends an OK message if outside of critical section
  - If already in critical section, does not reply, queue the request
  - If wants to enter critical section, sends an OK message if $(L_i, k) < (L_j, j)$, else queue the request
- When a process is finished with the critical section
  - Send OK messages to all processes on its queue and delete them from the queue

---

A Distributed Algorithm

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

- Fully distributed
- \( N \) points of failure!
- All processes are involved in all decisions
  - Any overloaded process can become a bottleneck
- Improvements
  - When a request comes in, always sends a reply granting or denying permission.
    - This helps detect dead processes
  - Enter critical section when the process has got permission from a simple majority of the other processes

A Token Ring Algorithm

(a) An unordered group of processes on a network. (b) A logical ring constructed in software.

- Use a token to arbitrate access to critical section
- Must wait for token before entering CS
- Pass the token to neighbor once done or if not interested
- No starvation!
- Detecting token loss is non-trivial
- Failing nodes can break the ring
# Comparison

<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>Decentralized</td>
<td>3mk, k=1,2,...</td>
<td>2mk, k=1,2,...</td>
<td>Starvation, low efficiency</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>