Distributed Algorithm

• Based on event ordering and time stamps
  – Assumes total ordering of events in the system (Lamport’s clock)
• Process k enters critical section as follows
  – Generate new time stamp $T_S = T_S + 1$
  – Send request $(k, T_S)$ to all other $n-1$ processes
  – Wait until reply $(j)$ received from all other processes
  – Enter critical section
• Upon receiving a request message, process $j$
  – Sends an OK message if no contention
  – If already in critical section, does not reply, queue request
  – If wants to enter, compare $T_S$ with $T_K$ and send an OK message if $T_S < T_K$, else queue the request
• When process $k$ is finished
  – Send OK messages to all processes on its queue and delete them from the queue

Properties

• Fully decentralized
• 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 send a reply granting or denying permission.
  – Enter critical section when the process has got permission from a simple majority of the other processes

Comparison

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Messages per entry</th>
<th>Delay before entry (in message times)</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td>2</td>
<td>2</td>
<td>Coordinator crash</td>
</tr>
<tr>
<td>Decentralized</td>
<td>2mk, k=1,2,...</td>
<td>2mk</td>
<td>Starvation</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>