Today
- Distributed Mutual Exclusion
- Leader Election

Distributed Mutual Exclusion
- Multiple processes on different machines may need to access a critical section
- Shared-memory systems:
  - Semaphores, mutexes, etc.
  - Typically implemented in shared memory
  - Processes share same blocking queues
- How to implement mutual exclusion in distributed systems?

Centralized Algorithm
- A coordinator grants access to critical section
  - Maintains a local queue
  - Can be elected using an election algorithm
- A process sends request to coordinator
  - If nobody in critical section, grant access
  - Otherwise, put process in queue
- When process done:
  - Send release to coordinator
  - Coordinator grants access to next process in queue
Centralized Algorithm

- Simple and efficient:
  - Requires only 3 messages per request grant
- No starvation or deadlock
- Problem:
  - What happens when coordinator crashes?

Replicated Coordinator

- Have $n$ replicas of the coordinator
  - A coordinator grants only one request at a time
- Need to get a majority of permissions
- Problems:
  - What if a coordinator fails and resets its state?
  - What if there is a lot of resource contention?

Timestamp-based Algorithm

- All events are totally ordered
- To gain access:
  - Send a request to all processes with timestamp
- On receipt of request:
  - If don’t care, send OK
  - If already in critical section, queue the request
  - If want to enter the critical section, compare timestamp of request to own request: Send OK or queue based on timestamp value
- Access granted: When all processes send OK
**Timestamp-based Algorithm**

- Requires $2(n-1)$ messages per access
- Any node becomes point of failure/bottleneck
- Dependent on all nodes
- Higher probability of failure than central algorithm
- Requires group communication
- Modifications:
  - Get permission from majority of processes
  - Get permission from overlapping subsets ($\sim \sqrt{n}$ size)

**Token Ring Algorithm**

- Processes arranged in a ring
- Token passes around the ring
  - Token holder has access to critical section
- If process wants to enter critical section:
  - Wait for the token
  - Enter the critical section while holding the token
  - Pass on the token when done
- If process does not want to enter critical section:
  - Pass the token to neighbour

**Token Ring Algorithm: Properties**

- Fairness: Each process gets chance in turn
- Worst-case wait: $O(n)$
- Problems:
  - How to detect a lost token?
  - What if a process crashes?
Mutual Exclusion Algorithms: Comparison

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Messages per entry/exit</th>
<th>Delay before entry (no. of messages)</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td>3</td>
<td>2</td>
<td>Coordinator crash</td>
</tr>
<tr>
<td>Replicated</td>
<td>3mk, k=1,2,...</td>
<td>2m</td>
<td>Starvation, inefficiency</td>
</tr>
<tr>
<td>Timestamp</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 ∞</td>
<td>0 to n−1</td>
<td>Lost token, Process crash</td>
</tr>
</tbody>
</table>

Leader Election

- Why do we need it?
  - Many systems require a coordinator, monitor, initiator, central server, etc.
  - It may not matter who the leader is
- Examples?

Election Algorithms

- Goal: All processes must agree on the leader after the election
- Choice of leader
  - Process with the highest ID
  - Process with desired properties, e.g.: resource capacity, location, etc.
- Question: How do we determine the leader?

Bully Algorithm

- Process with highest ID "bullies" everyone into accepting it as a leader
- Initiation:
  - A process P sends ELECTION message to all processes with higher ID's
  - If no one responds, P wins the election
  - If someone responds, it takes over the election
- Last process remaining becomes the leader
  - Sends a Victory message to everyone
**Bully Algorithm: Initiation**

- Assume n processes initially
- Worst Case:
  - Smallest process initiates election
  - Requires $O(n^2)$ messages
- Best Case:
  - Eventual leader initiates election
  - Requires $(n-1)$ messages

**Bully Algorithm: Leader Election**

- Processes arranged in a ring
- Each process has a successor
- Initiation:
  - A process sends an ELECTION message to its successor (or next alive process) with its ID
  - Each process adds its own ID and forwards the ELECTION message
- Leader Election:
  - Message comes back to initiator
  - Initiator announces the winner by sending another message around the ring

**Ring Algorithm**
**Ring Algorithm: Initiation**

![Ring Algorithm Diagram]

**Properties**
- If only 1 process initiates election:
  - Requires 2n messages
- Two or more processes might simultaneously initiate elections
  - Still ensures election of the same leader
  - Results in extra messages

---

**Election in Wireless Networks**
- Restricted information
  - Nodes do not know everyone's identity
  - Overall topology may not be known
  - Want "best" node to be leader
    - E.g.: most battery life, capacity, etc.

---

**Election Tree**
- Initiation:
  - One node starts election
  - Send ELECTION message to all neighbors
- On receiving ELECTION message:
  - If first message, assign sender as parent
  - Forward to all other neighbors
  - Otherwise, ACK to sender
- Responding to parent:
  - After getting ACKs from all neighbors
  - Also pass on info on "best" downstream node
Election in P2P Systems

- Electing Superpeers
- Goals:
  - Fixed proportion of total no. of nodes
  - Load balanced
- DHT networks:
  - Use top k-bits to identify superpeers
  - Superpeer for p = Node responsible for
    p&1...10...0 (first k bits 1)
  - No. of superpeers \( \approx 2^{k-m} N \)