Process Synchronization

- Mutual Exclusion Algorithms
  - Permission Based
  - Centralized
  - DeCentralized
  - Distributed
  - Token Based
  - Token Ring

- Election Algorithms
  - Bully Algorithm
  - Ring Algorithm
  - Election in Wireless Networks
  - Election in Large Scale Systems
Process Synchronization

- **Concurrency** and **Collaboration** are fundamental to process synchronization in distributed systems
  - Processes may required coordination to carry out a global, collective task in a consistent manner
  - Processes may seek to access simultaneously the same resource
    - Access to a shared resource must – **Critical Section**
    - Exclusive access to the resource is required
- Algorithms are required to coordinate execution among processes
  - Grant **Mutual Exclusive Access** by processes
  - Leader Election, etc
Mutual Exclusion in Distributed Systems

- The lack of a common shared memory makes the problem of mutual exclusion more challenging in distributed systems as compared to centralized systems.

- Desirable properties for possible DME solutions:
  - Ensure Mutual Exclusion,
  - No deadlocks – Deadlock is a state of the system when no node is in its CS and yet no requesting node can proceed to its CS.
  - No starvation – Starvation occurs when some node waits indefinitely to enter its CS, while other nodes can enter and exit their CSs.

Mutual Exclusion – Classification

- Mutual Exclusion
  - Permission-Based
    - Decentralized
    - Centralized
  - Token-Based
    - Distributed
    - Token Ring
Centralized Algorithm

- Centralized Algorithm achieves DME by closely “mimicking” ME in single processor systems
  - One process, C, is the Coordinator – Coordinates access to resources
  - Other processes issue requests to access resource

1. **Request Resource**
2. **Wait for Response**
3. **Receive Grant**
4. **Access Resource**
5. **Release Resource**
Centralized Algorithm — Allocated Resource

- If resource is currently by another process, C does not reply until resource is released
- Maintain queue of pending requests, serviced in FIFO order

Centralized Algorithm

- Advantages
  - Easy to understand, verify and implement
  - Access Fairness – FIFO order of service is simple to implement
    - Note FIFO policy does not guarantee “Fair Share” access

- Limitations
  - Not scalable to large scale distributed systems – Coordinator is likely to become a bottleneck
  - A process requesting a resource cannot distinguish between being in a "blocked state", waiting for the resource to be released, from waiting for a failed coordinator
    - Timeout mechanisms or alive messages may be required – to be designed carefully to avoid early and late timeouts or unnecessary alive messages
Decentralized Permission Based DME

- Fully decentralized algorithms attempt to address the bottleneck problem of centralized algorithms
  - Basic tenet of the algorithm is coordination replication and majority voting
- Each resource is assumed to be replicated $n$ times
- Each replica has its own coordinator
  - Therefore, $n$ coordinators are needed
- Voting algorithm can be executed using DHT-based approach
DME Decentralized Voting Algorithm

- Unlike in the centralized scheme, if the resource cannot be granted, a decentralized DME coordinator informs the requesting process that the resource is unavailable.
- To gain access to a resource, a process may secure a majority vote \( m > n/2 \), where \( n \) is the number of coordinators.
- A failed coordinator resets at arbitrary moments, not having to remember any vote it gave before the crash.
  - Ignoring previously granted permissions, may lead a recovered coordinator to grant permission again to another process.

Decentralized Permissions – Analysis

- Benefits
  - No single point of failure
  - Implementable using DHT based structure
    - If permission to access a resource is denied, because the process cannot obtain a majority vote, the process backs off a random amount of time and tries again later.
- Limitations
  - Overhead in terms of the number of messages generated can be prohibitively high.
    - Messages to secure majority, which may require multiple attempts.
    - Failure may be difficult to handle.
PERMISSION BASED DME

DISTRIBUTED ALGORITHM

A Distributed ME Algorithm

- **Ricart and Agrawala Algorithm** assumes there is a mechanism for “**totally ordering of all events**” in the system and a **reliable** message system
  - **Lamport’s algorithm** can be used for total ordering
- A process wanting to enter it CS sends a message with (CS name, process id, current time) to all processes, including itself
- When a process receives a CS request from another process, it reacts based on its current state with respect to the CS requested.
  - **Three** possible cases must be considered
Ricart and Agrawala Algorithm – Basic Cases

- RA Algorithm distinguishes between 3 cases
  a. If the receiver is not in the CS and it does not want to enter the CS, it sends an OK message to the sender.
  b. If the receiver is in the CS, it does not reply and queues the request.
  c. If the receiver wants to enter the CS but has not yet, it compares the timestamp of the incoming message with the timestamp of its message sent to everyone – The lowest timestamp wins.
    - If the incoming timestamp is lower, the receiver sends an OK message to the sender.
    - If its own timestamp is lower, the receiver queues the request and sends nothing.

RA DME Algorithm – CS Entry and Exit

- After a process sends out a request to enter a CS, it waits for an OK from all the other processes.
  - Upon receiving all OK message, process enters the CS
- Upon exiting CS, it sends OK messages to all processes on its queue, which are waiting for that CS
  - Upon sending OK messages, processes are deleted from the queue.
RA Algorithm – Example

- $P_0$ and $P_2$ request access to the same resource, nearly at the same time.
  - Both send requests with timestamps, 8 and 12, respectively.
- $P_0$ has the lowest timestamp, 8.
  - $P_0$ wins access to the resource.
- Upon completion of its CS, $P_0$ sends OK message to $P_2$.
  - $P_2$ can enter its own CS.

Distributed Permissions Analysis

- Benefits
  - No central bottleneck
    - This results in improved performance
  - Fewer messages than the decentralized algorithm
- Limitations
  - The system is exposed to $n$ points of failure
    - If a node fails to respond, the entire system locks up
Token-Based DME

Token Ring Algorithm

A Token Ring Algorithm

- An unordered group of processes on a network.
- A logical ring constructed in software
- Token circulates at high speed on the network
- A process must have a token to enter its CS
### DME Algorithm 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^{1})</td>
<td>Coordinator Crash</td>
</tr>
<tr>
<td>Decentralized</td>
<td>(2mk+m, k=1,2,…)</td>
<td>(2mk)</td>
<td>Starvation, Low Efficiency</td>
</tr>
<tr>
<td>Distributed</td>
<td>(2(n-1))</td>
<td>(2(n-1))</td>
<td>Process Crash</td>
</tr>
<tr>
<td>Token ring</td>
<td>1 to (\infty)</td>
<td>0 to (n-1)</td>
<td>Lost Token, Process Crash</td>
</tr>
</tbody>
</table>

- Centralized is the most efficient.
- Token ring efficient when a large of processes seek to use critical region.

---

**ELECTION**

**BACKGROUND**
Election Algorithms

- Many distributed algorithms such as mutual exclusion and deadlock detection require a coordinator process.
- When the coordinator process fails, the distributed group of processes must execute an election algorithm to determine a new coordinator process.
  - Different criteria can be used to select the new coordinator.

Assumptions and Requirements

- Any process can call for an election.
  - A process can call for at most one election at a time.
  - Multiple processes can call an election simultaneously.
  - Election results should not depend on which process calls for it.
- Each process maintains the following data structure
  - Variable called elected
  - An attribute value called attr, representing ID, MAC address, fastest CPU, etc.
- The non-faulty process with the best (highest) election attribute value (e.g., highest ID or MAC address, or fastest CPU, etc.) is elected.
ELECTION ALGORITHMS

BULLY ALGORITHM

Bully Algorithm Main Characteristics

- Operating Assumptions
  - Synchronous system
    - All messages arrive within $T_M$ units transmission of time.
    - A reply is dispatched within $P_p$ units of processing time after the receipt of a message.
    - If no response is received in $2 \times T_M + P_p$, the node is assumed to be faulty
      - Node crashed
  - Attribute = Process ID
  - Each process knows all the other processes in the system
    - Therefore, processes know each others’ IDs
The Bully Algorithm

When any process, P, notices that the coordinator is no longer responding it initiates an election:

1. P sends an election message to all processes with higher id numbers.
2. If no one responds, P wins the election and becomes coordinator.
3. If a higher process responds, it takes over.
   - Process P’s job is done.

The Bully Algorithm

- At any moment, a process can receive an election message from one of its lower-numbered colleagues.
- The receiver sends an OK back to the sender and conducts its own election.
- Eventually only the bully process remains.
  - Bully process becomes the new coordinator
  - The bully announces victory to all processes in the distributed group.
Bully Algorithm Example

- Process 4 notices 7 down.
- Process 4 holds an election.
- Process 5 and 6 respond, telling 4 to stop.
- Now 5 and 6 each hold an election.

- Process 6 tells process 5 to stop
- Process 6 (the bully) wins and tells everyone
- If processes 7 recovers, it restarts election process
Performance of Bully Algorithm

- **Best case scenario:** The process with the second highest id notices the failure of the coordinator and elects itself.
  - $N-2$ coordinator messages are sent.
  - Turnaround time is one message transmission time.
- **Worst case scenario:** When the process with the least id detects the failure.
  - $N-1$ processes altogether begin elections, each sending messages to processes with higher ids.
  - The message overhead is $O(N^2)$.
  - Turnaround time is approximately 5 message transmission times if there are no failures during the run: election, answer, election, answer, coordinator.
Ring Algorithm – Basic Operation

- RA assumes that the processes are logically ordered in a ring (implies a successor pointer and an active process list) that is unidirectional.
- When any process, P, notices that the coordinator is no longer responding it initiates an election:

1. P sends message containing P’s process ID to the next available successor.

Ring Algorithm – Basic Operation

2. At each active process, the receiving process adds its process number to the list of processes in the message and forwards it to its successor.
   - Eventually, the message gets back to the sender.

3. The initial sender sends out a second message letting everyone know who the coordinator is (the process with the highest number) and indicating the current members of the active list of processes.
Ring Algorithm – Example

- Even if two ELECTIONS start at once, every node will pick the same leader.
Very Large Scale Networks

- To deal with scale, more than one node should be selected
- Nodes organized as peers and super-peers
  - Normal nodes should have low-latency access to super-peers
  - Super-peers should be evenly distributed across the network
    - There should be a predefined portion of super-peers relative to the total number of nodes in the overlay network
    - Each super-peer should not need to serve more than a fixed number of normal nodes
- Elections held within each peer group
- Super-peers coordinate among themselves

Conclusion

- Mutual Exclusion Algorithms
  - Permission Based
    - Centralized
    - DeCentralized
    - Distributed
  - Token Based
    - Token Ring
- Election Algorithms
  - Bully Algorithm
  - Ring Algorithm
  - Election in Large Scale Systems