Coordination and Agreement

➢ Introduction

- ***How processes coordinate their actions?***
- Main assumptions in coordination:
  - ✓ Each pair of processes is connected by reliable channels
  - ✓ Processes independent from each other
  - ✓ Processes fail only by crashing

Coordination and Agreement

Distributed Mutual Exclusion

- Distributed processes require a mechanism that can coordinate their activities because they share a resource or collection of resources
- Mutual exclusion is required to
  - ✓ prevent interference
  - ✓ ensure consistency when accessing the resources

Coordination and Agreement

➢ Distributed Mutual Exclusion

- Distributed processes require a mechanism that can coordinate their activities because they share a resource or collection of resources
- Mutual exclusion is required to
  - ✓ prevent interference
  - ✓ ensure consistency when accessing the resources

Coordination and Agreement

➢ Distributed Mutual Exclusion

- Algorithms for mutual exclusion
  - ✓ Requirements for mutual exclusion are:
    - Safety - At most one process may execute in the critical section (CS) at a time.
    - Liveness - Requests to enter and exit the critical section eventually succeed.
    - Ordering - If one request to enter the CS happened-before another, then entry to the CS is granted in that order.

Coordination and Agreement

Distributed Mutual Exclusion

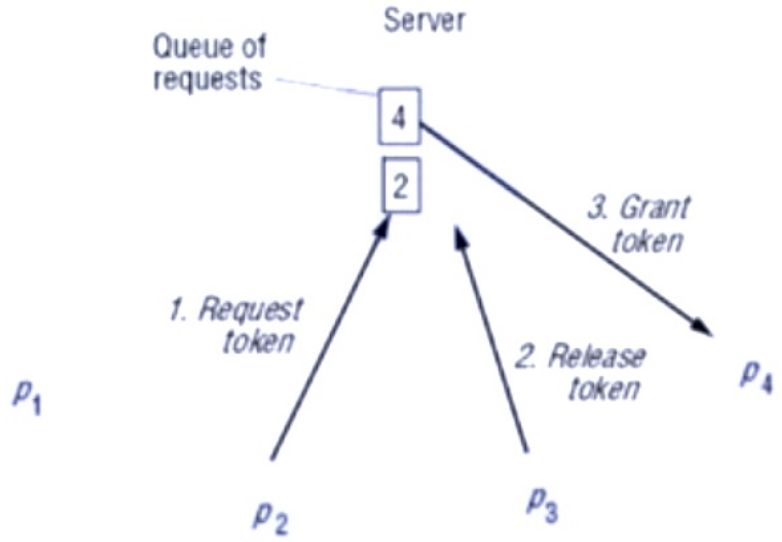
- Algorithms for mutual exclusion
  - ✓ The criteria:
    - the bandwidth consumed, which is proportional to the number of messages sent in each entry and exit operation;
    - the client delay incurred by a process at each entry and exit operation;
    - the algorithm's effect upon the throughput of the system.
  - ✓ Some examples of algorithms:
    - The central server algorithm
    - Ring-Based Algorithm
    - Multicast and Logical Clocks
    - Maekawa's Voting Algorithm

Coordination and Agreement

Distributed Mutual Exclusion

- The central server algorithm
  - ✓ Employs a server that grants permission to enter the critical section.

Server managing a mutual exclusion token for a set of processes

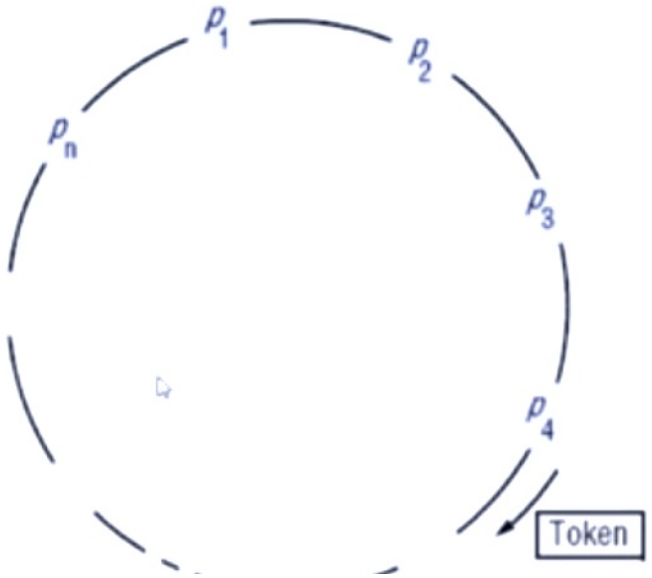


Coordination and Agreement

➢ Distributed Mutual Exclusion

- Ring-Based Algorithm
  - ✓ Arrange the processes in a logical ring

A ring of processes transferring a mutual exclusion token

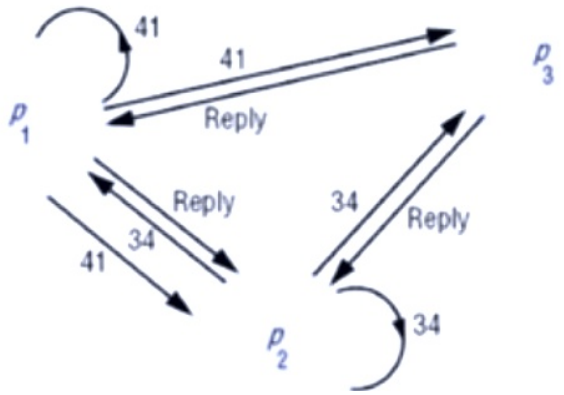


Coordination and Agreement

➢ Distributed Mutual Exclusion

- Multicast and Logical Clocks
  - ✓ processes that require entry to a critical section multicast a request message, and can enter it only when all the other processes have replied to this message

Multicast synchronization



Coordination and Agreement

Election

- An algorithm for choosing a unique process to play a particular role
  - ✓ a process p_i *can be*
    - *a participant : is engaged in* some run of the election algorithm
    - a *non-participant : is not* currently engaged in any election
  - ✓ Some examples of election algorithms
    - A ring-based election algorithm
    - The bully algorithm

Coordination and Agreement

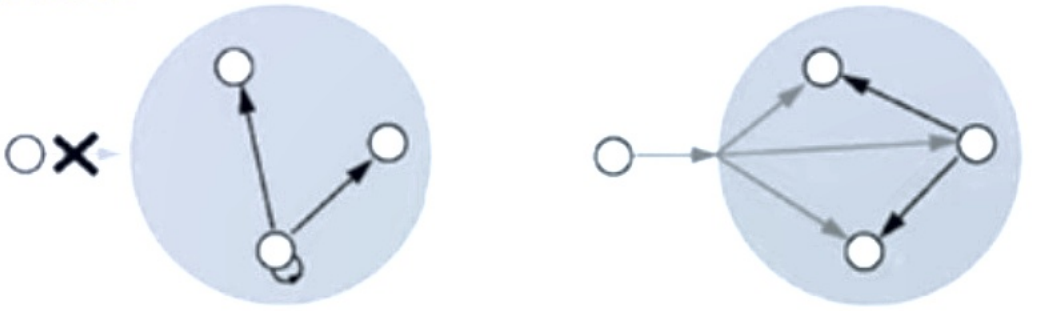
➢ Election

- The Bully algorithm
  - ✓ Allows processes to crash during an election, although it assumes that message delivery between processes is reliable
  - ✓ It assumes that the system is synchronous (uses timeouts to detect a process failure)
  - ✓ It assumes that each process knows which processes have higher identifiers, and that it can communicate with all such processes
  - ✓ 3 types of message used in this algorithm
    - an *election message - is sent to* announce an election;
    - an *answer message - is sent in response to an election message*
    - a *coordinator message - is sent to announce the identity of the elected process*

Coordination and Agreement

Coordination and Agreement in Group Communication

- *System:* contains a collection of processes, which can communicate reliably over one-to-one channels
- *Processes:* members of groups, may fail only by crashing



Closed group

Open group

Coordination and Agreement

➢ Coordination and Agreement in Group Communication

- Primitives:
  - ✓ ***multicast(g, m)*:** sends the message *m* to all members of group g
  - ✓ ***deliver(m)* :** delivers the message *m* to the calling process
  - ✓ ***sender(m)* :** unique identifier of the process that sent the message *m*
  - ✓ ***group(m)*:** unique identifier of the group to which the message *m* was sent

Coordination and Agreement

➢ Coordination and Agreement in Group Communication

- Basic Multicast
  - ✓ Guarantee that a correct process will eventually deliver the message as long as the multicaster does not crash
  - ✓ **Primitives:** B_multicast, B_deliver
  - ✓ **Implementation:** Use a reliable one-to-one communication
  - ✓ **Unreliable:** Acknowledgments may be dropped

To *B-multicast*(g, m): for each process $p \in g$, *send*(p, m);

On *receive*(m) at p: *B-deliver*(m) at p.

Coordination and Agreement

➢ Coordination and Agreement in Group Communication

- **Reliable Multicast**
- **Properties to satisfy:**
  - **Integrity:** A correct process P delivers the message m at most once
  - **Validity:** If a correct process multicasts a message m, then it will eventually deliver m