LECTURE 5

Mutual Exclusion and Election

Shared resources

- Processes may need to access the same resources.
- Concurrent accesses will corrupt the resource.
  - Make it inconsistent (consistency later)
- Need for solutions that that facilitate coordination between different processes.
- Two different ways:
  - Token based solutions
  - Permission based solutions

Permission based approaches

- Process wanting to access a resource must first acquire permission from other processes.
- How to do so ?
  - Various ways we will see.

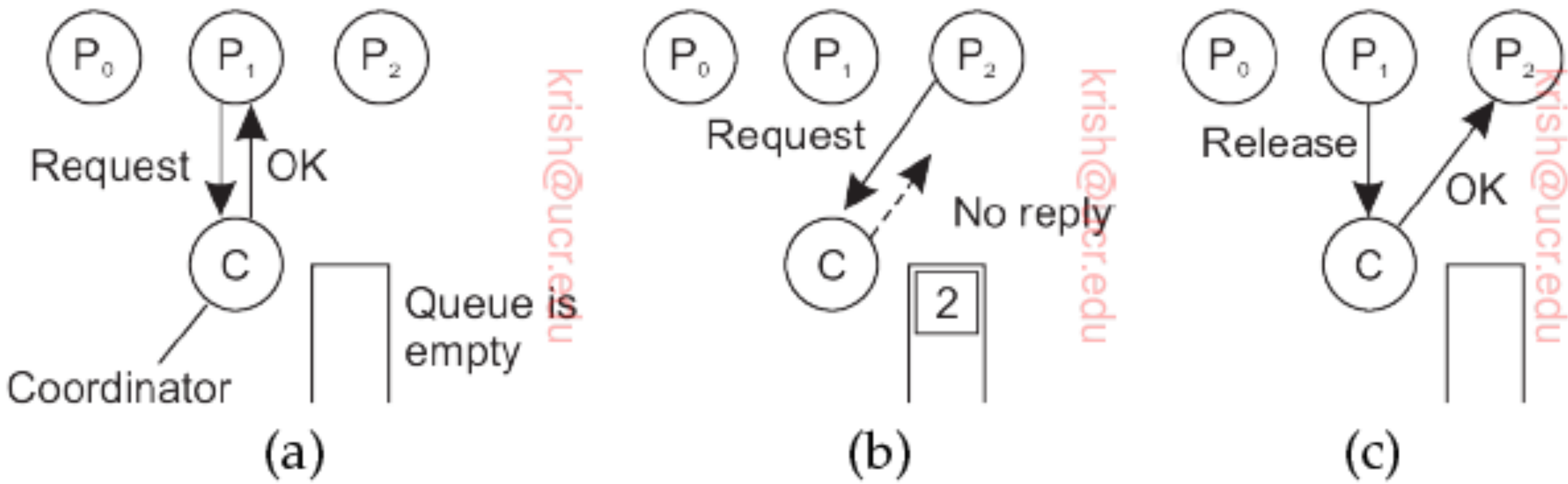
Token-based solutions

- Mutual exclusion achieved by using a special message called a token
- Only one token available → anyone who has the token can access the shared resource.
- Avoid deadlocks and starvation
- However, challenge when token is lost → process holding the token can crash

A centralized algorithm

- Choose a coordinator (election → later)
- A process that seeks to access a resource sends a message to the coordinator seeking permission.
- If no other process is accessing the resource, permission granted. (use a reply)
- If some other process is using the resource – permission cannot be granted.
  - How to handle is system dependent.
  - e.g., just don't reply so that the requesting process blocks.

Coordination functions



- When resource is released, coordinator is notified.
- Coordinator picks first item off a queue of waiting requests and assigns resource.
- Easy to see mutual exclusion guaranteed.
- No starvation -- process is fair.

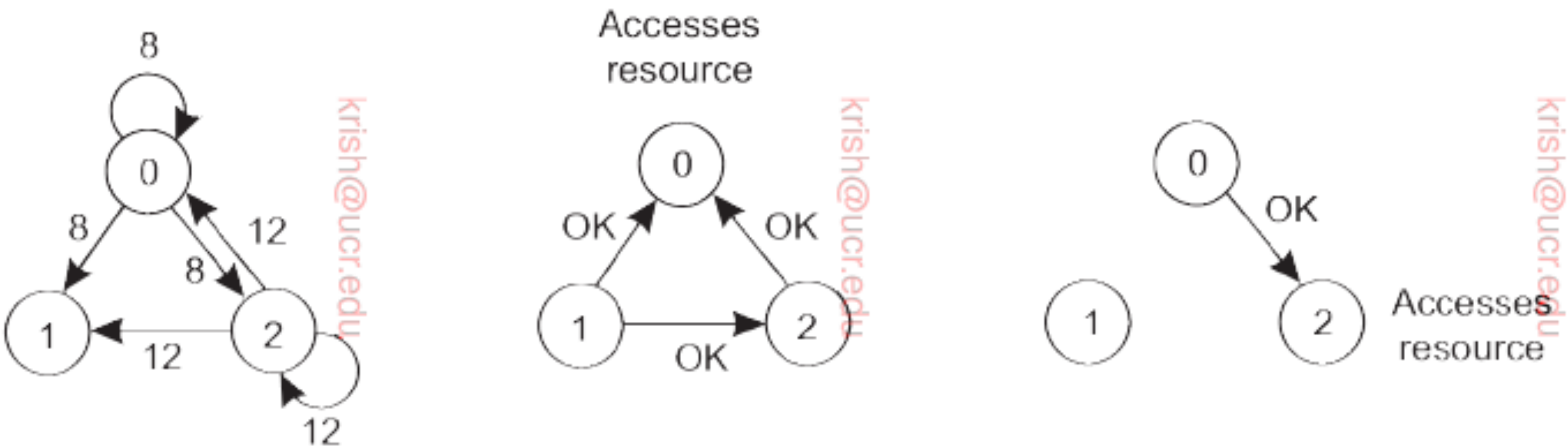
How to make it distributed?

- Use of Lamport's clocks
- Need total ordering of evens
  - Unambiguous which happened first
- When a process wants to access a resource, it builds a message which includes:
  - Name of resource
  - Process number
  - current logical time.
- Send message to everyone else (broadcast)
  - Assume reliable transmissions

Receiver functions

- If not accessing the resource and does not want to access it send "OK"
- Don't reply if accessing resource – just queue request.
- If receiver wants to access resource, but has not done so:
  - Compare time stamp with its own message (which it has sent everyone).
  - If time stamp lower send OK (lowest timestamp wins)
  - Else, queue request and send nothing.

Example of distributed algorithm



(a) (b) (c)

- Receiver waits until everyone gives permission.
- Once it gets, it accesses resource.
- Upon completion of usage, send OK to everyone in queue.

Message complexity

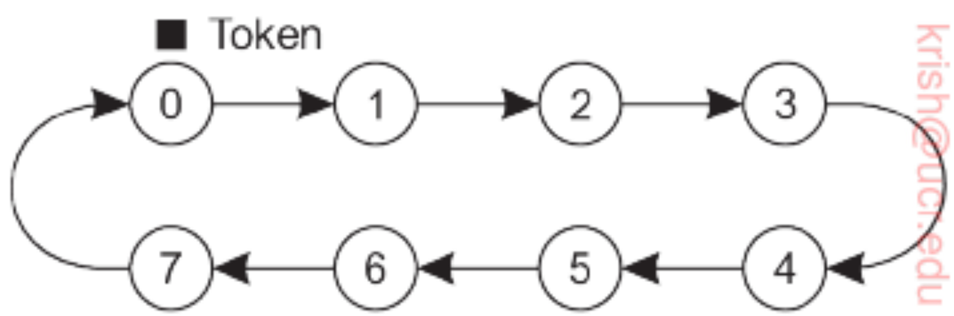
- The distributed algorithm guarantees mutual exclusion without starvation or deadlocks.
- Number of messages before resource acquisition:
  - (N-1) requests to all other processses
  - (N-1) OK messages from all other processes

Failures

- N points of failure
  - Any process crash is wrongly interpreted as denial of permission.
  - Blocks all subsequent attempts by processes to acquire resource.
- Patch:
  - When a request arrives, receiver always grants or denies permission.
  - If nothing is got within a time-out, keep trying until a reply is obtained, or the receiver is deemed dead.

Token ring

- A logical overlay (application level) ring is formed.
- Each process is assigned a position on the ring.
  - Each one needs to know who is next in line after itself.



Token ring algorithm

- P0 is given a token → which allows the process to access the resource.
- Upon completion, it passes it to P1 and the process continues.
- In general if there are N processes, P(k) → P(k+1) mod N
- If a node that receives the token has no interest in the resource, it simply passes on the token on the ring.
- Nodes cannot immediately access the resource for a second time using the same token.

Issues

- Process with token might crash
  - Hard to detect (process may still be accessing resource)
  - Time bounds?
- ACKs
  - If a process has to ACK token receipt – lack of ACKs could help detect a dead process
    - Remove dead process from the group.

Decentralized algorithm

- Each resource replicated N times and each has its own coordinator.
- When a process wants to access the resource, it needs OKs from $m > n/2$ coordinators for that resource.
  - Majority vote
- When a permission has already been granted to a different process, coordinator tells requester.

Fault model

- When a coordinator crashes, it recovers quickly but forgets its vote (before crash)
  - Thus, it may incorrectly grant permission again to another process after recovery.
- Recall: m coordinators had granted permission to the process accessing resource
- Let $p = \Delta t/T$ be the probability of a coordinator reset. Then, probability k out of m coordinators reset is

$$\mathbb{P}[k] = \binom{m}{k} p^k (1-p)^{m-k}$$

Condition for correctness

- Let f coordinators fail; the remaining will be m-f.
- In order for this algorithm to work correctly, the remaining must still constitute the majority.
  - That is → $m - f > N/2$ or $f < m - N/2$
- In order for a violation $f \geq m\text{-}N/2$ and this occurs with a probability

$$\sum_{k=m-N/2}^{m} \mathbb{P}[k]$$

- For typical values of N, m, T and Δt, these are quite small (e.g., for N= 16, m=9 T= 1 hour, Δt = 30 seconds), the violation probability is less than 10^{-18}.
  - Thus it can obe often neglected.

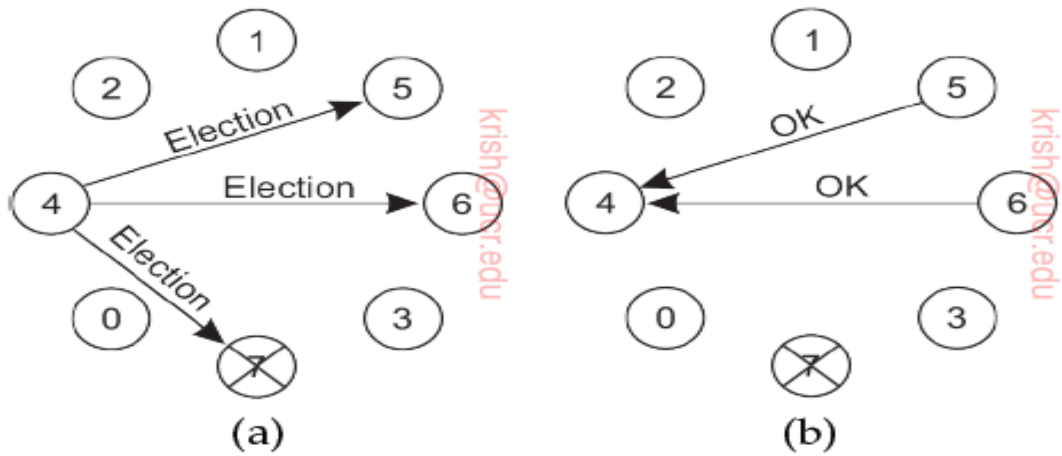
Election algorithms

- As discussed, many algorithms require one process to act as a coordinator, initiator or perform some special role.
- If all processes are the same, how do we select this coordinator ?
- Assume that each process P has a unique identifier id (P).
  - Election is to locate the process with the highest ID and designate it as the coordinator.
  - Algorithms differ in terms of how they locate this highest ID Process.

The bully algorithm

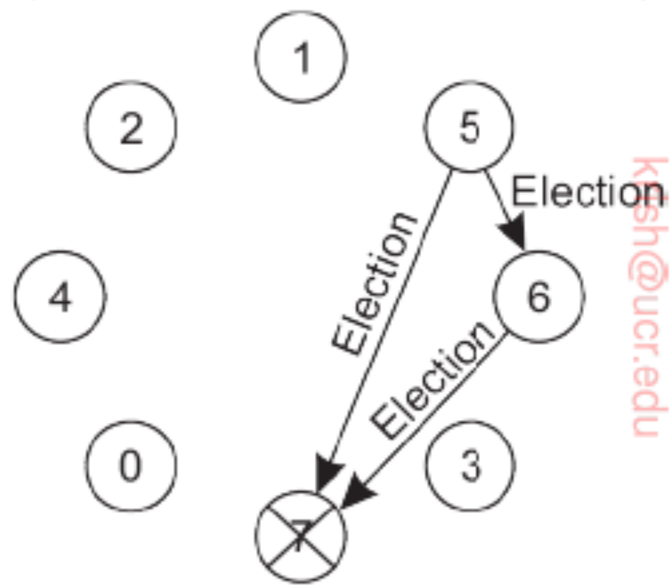
- There are n processes [P0 … Pn-1].
- Let ID of Pk = k.
- An election is invoked when a process notices that the current coordinator is no longer responding to requests.
- The process Pk sends an ELECTION message to all processes with higher identifiers (Pk+1 and so on until Pn-1)
- If no one wins Pk wins and it becomes coordinator.
- Else, if one of the higher ups answers, it takes over.
  - Pk’s job is done.

Example



- Process 4 notices that the coordinator is not responding.
- It sends messages to processes 5, 6 and 7.
- 5 and 6 respond → this is a cue for process 4 to withdraw.

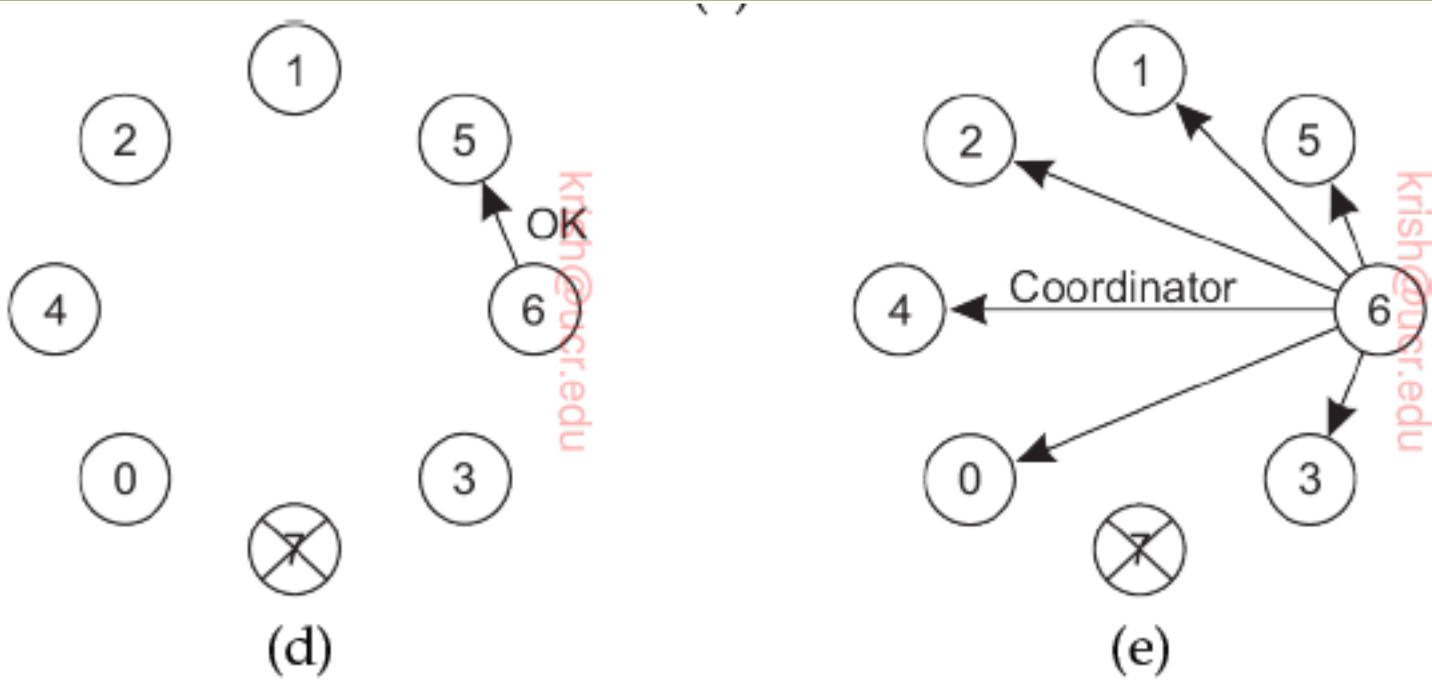
Example (continued)



(c)

- Now, 5 and 6 hold an election
  - 5 sends ELECTION messages to 6 and 7
  - 6 sends ELECTION message to 7

Example (contd)



- Process 6 tells 5 to stop.
- There is no response from 7. This means process 6 has won the election.
- It tellls all other processes (bully them into submission ☺)

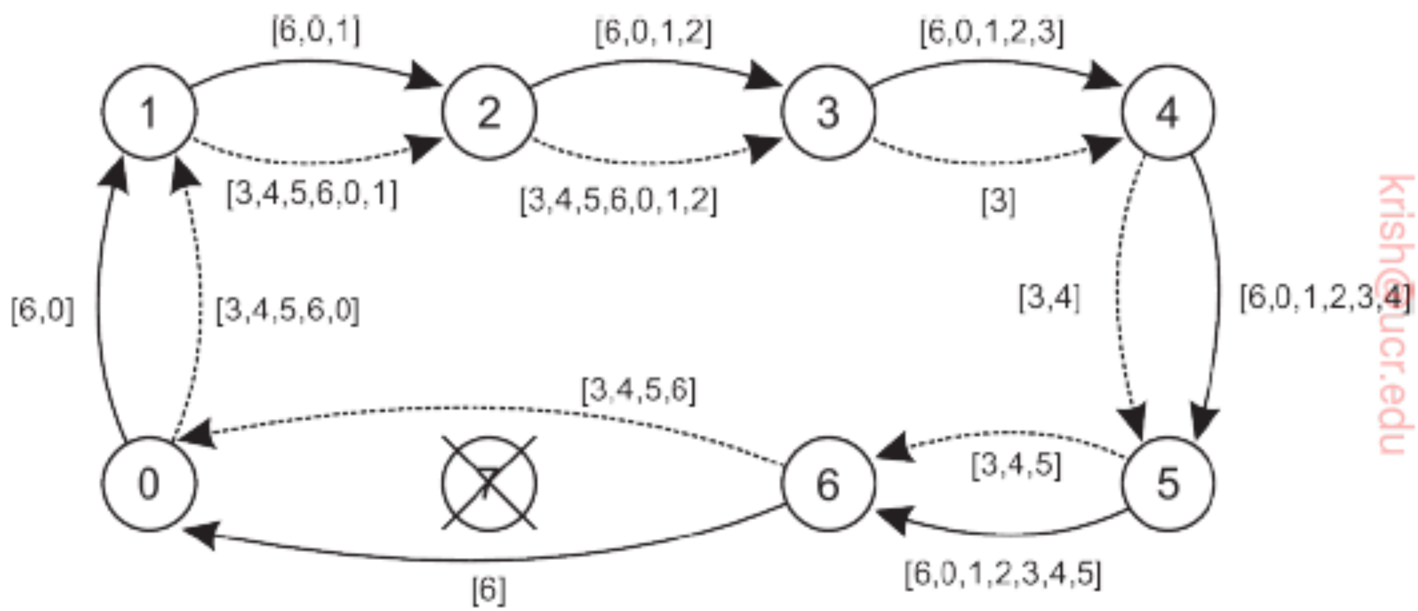
Ring algorithm

- Election also could be based on a logical ring
  - Note the physical topology is not a ring but they are logically organized that way.
- Does not use a token (like in token ring)
- Each process knows who is its successor.
- When a process notices that the coordinator is not functioning, it begins an election.

Election process

- The process that discovers the failed coordinator builds an ELECTION message
  - contains its own ID (creates a list).
- Sends to successor.
  - If successor is down, sender skips and goes to next member along the ring or one after that and so on – until a running process is located.
- At each step, the process that sends adds its ID to the list in the message.
- When message returns to the process that initiated the election, it identifies the highest ID and chooses that process as coordinator.
  - A new coordinator message is sent to everyone.

Example



- The example illustrates what happens when P3 and P6 discover simultaneously that the previous coordinator P7 has crashed.
- Note they converge to the same new coordinator (P6).