CT30A3401
Distributed Systems
Lecture 8

Bilal Naqvi, PhD.
syed.naqvi@lut.fi

LAPPEENRANTA UNIVERSITY OF TECHNOLOGY

Mutual exclusion

- To enable concurrency and collaboration the processes will need to simultaneously access the same resources
  - such concurrent accesses may corrupt the resource, or make it inconsistent
- Mutual exclusion is a concurrency control property which is introduced to prevent race conditions.
  - requirement that a process can not enter its critical section while another concurrent process is currently present or executing in its critical section
  - only one process is allowed to execute the critical section at any given instance of time

Mutual exclusion in distributed systems

- In single processor systems mutual exclusion is often solved by maintaining status of shared resources in a shared variable
  - semaphore
- In distributed systems, it is more complicated due to: (1) no centralized clock to lack of shared memory, (2) a common physical clock
- To eliminate the mutual exclusion problem in distributed system approach based on message passing is used
  - mutual exclusion algorithms for determining right to access

Requirements for mutual exclusion algorithms

- No deadlock: Two or more processes should not endlessly wait for any message that will never arrive
- No starvation: Each process that wants to execute in critical section should get an opportunity to execute it in finite time
- Fairness: Each site should get a fair chance to execute critical section. Any request to execute critical section must be executed in the order they are made
- Fault tolerance: In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption

Mutual exclusion algorithms

- Classified into two categories
  - Token-based approach
  - Permission-based approach

Token-based approach

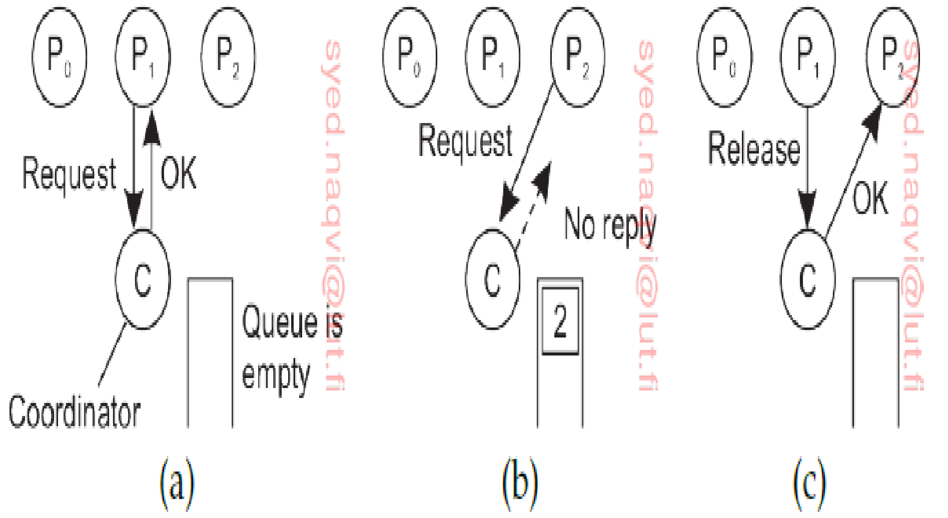
- Mutual exclusion is achieved by passing a special message between the processes, known as a token
- There is only one token available and who ever has that token is allowed to access the shared resource
- When finished, the token is passed on to a next process. If a process having the token is not interested in accessing the resource, it passes it on
- Drawback: When the token is lost (e.g., because the process holding it crashed), an intricate distributed procedure needs to be started to ensure that a new token is created, but above all, that it is also the only token

Permission-based approach

- A process wanting to access the resource first requires the permission from other processes
- Two ways of getting permission, (1) centralized, (2) de-centralized/ distributed

Centralized approach

- One process is elected as the coordinator
- Whenever a process wants to access a shared resource, it sends a request message to the coordinator stating which resource it wants to access and asking for permission.
- If no other process is currently accessing that resource, the coordinator sends back a reply granting permission

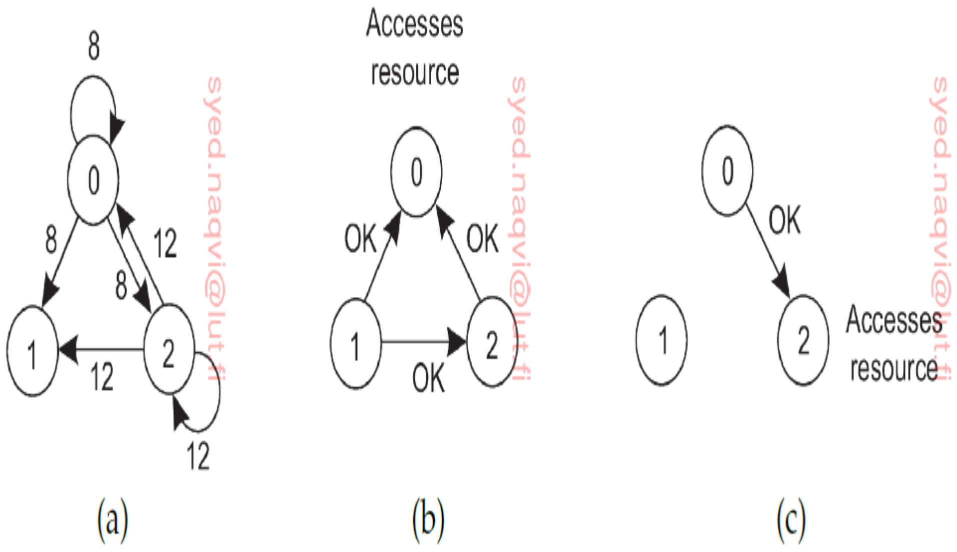


Distributed approach

- When a process wants to access a shared resource, it builds and sends a message containing the name of the resource, its process number, and the current (logical) time to all other processes, conceptually including itself.
- The sending of messages is assumed to be reliable; that is, no message is lost.
- When a process receives a request message from another process, there can be three different cases

Distributed approach

- If the receiver is not accessing the resource and does not want to access it, it sends back an OK message to the sender
- If the receiver already has access to the resource, it simply does not reply and queues the request
- If the receiver wants to access the resource as well but has not yet done so, it compares the timestamp of the incoming message with the one contained in the message that it has sent everyone. The lowest one wins

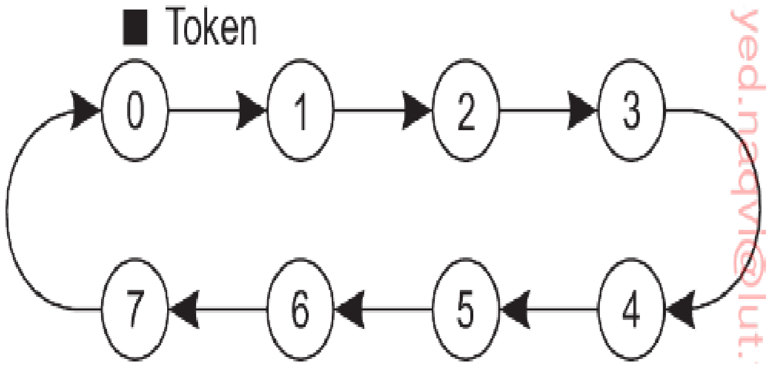


Distributed approach

- This algorithm has N (number of processes) points of failure
  - what if any process crashes and fails to respond to requests?
  - silence will be interpreted (incorrectly) as denial of permission
- Patch: make each process send both (OK and denial), silence will not be considered as denial

Token-ring algorithm

- An overlay network in the form of a logical ring is formed in which each process is assigned a position in the ring.
  - each process knows who is next in line after itself
- When the ring is initialized, process P_0 is given a token
- The token circulates around the ring in point-to-point messages



Token-ring algorithm

- When a process acquires the token from its neighbor, it checks to see if it needs to access the shared resource. If so, the process goes ahead, does all the work it needs to and releases the resources
- If a process is handed the token by its neighbor and is not interested in the resource, it just passes the token along
- Only one process has the token at any instant, so only one process can get to the resource
- If holder of the token crashes it must be regenerated
- If the token has not been spotted for an hour does not mean that it has been lost; somebody may still be using it
  - how to distinguish between token loss or being used?

A decentralized algorithm (voting algorithm)

- Each resource is assumed to be replicated N times
  - each replica has its own coordinator for controlling the access by concurrent processes
- Whenever a process wants to access the resource, it will simply need to get a majority vote from $m > N/2$ coordinators
- Limitation
  - When multiple nodes want to access the same resource, there are so many nodes competing to get access that eventually no one is able to get enough votes leaving the resource unused

Election algorithms

- Many distributed algorithms require one process to act as coordinator, initiator, or otherwise perform some special role
- Different algorithms exist for such election

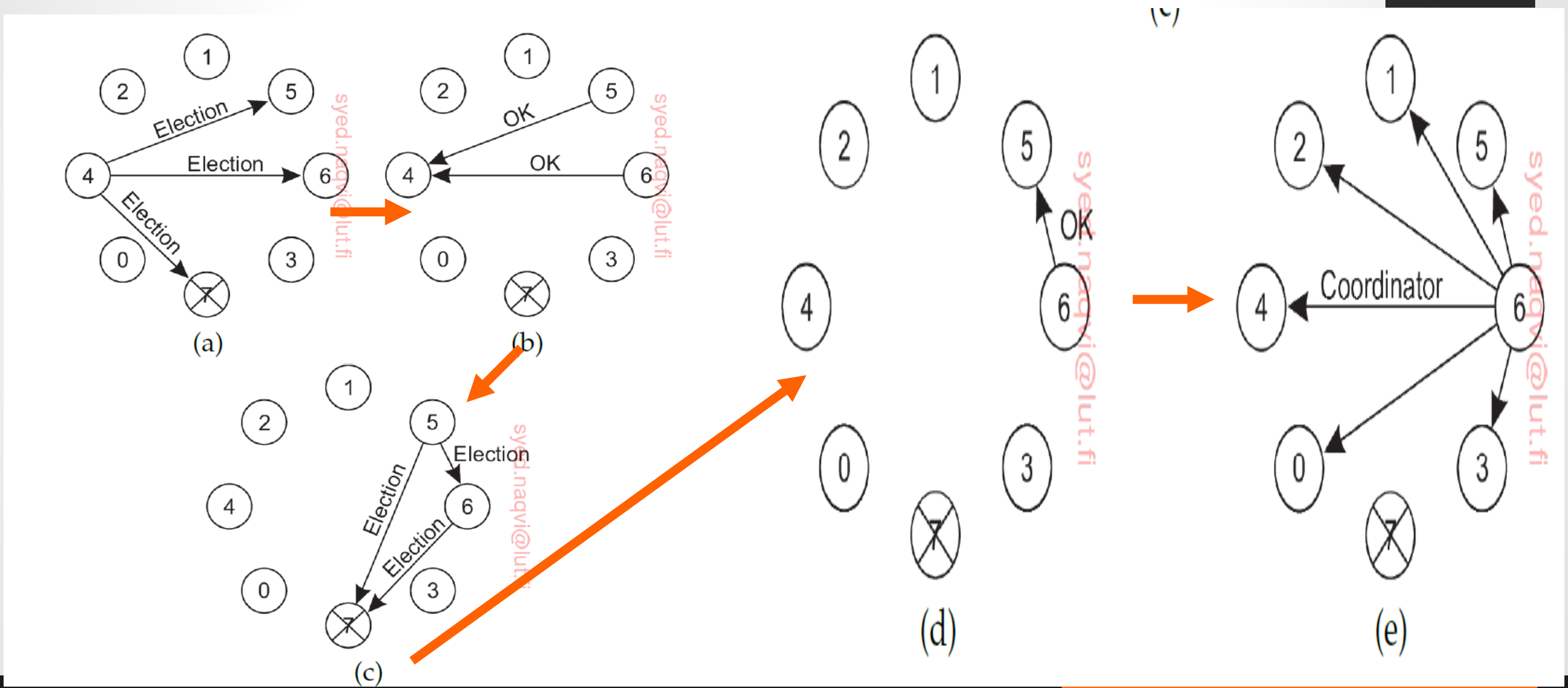
Bully algorithm

- When any process notices that the coordinator is no longer responding to requests, it initiates an election
- A process, holds an election as follows:
  - P_k sends an ELECTION message to all processes with higher identifiers P_{k+1}, P_{k+2} … P_{k+N}
  - If no one responds, wins the election and becomes coordinator
  - If one of the higher-ups answers, it takes over and k’s job is done

Bully algorithm

- A process can get an ELECTION message from one of its lower-numbered colleagues
- When such a message arrives, the receiver sends an OK message back to the sender to indicate that he is alive and will take over
- The receiver then holds an election, unless it is already holding one
- Eventually, all processes give up but one, and that one is the new coordinator
- It announces its victory by sending all processes a message telling them that starting it is the new coordinator

Bully algorithm



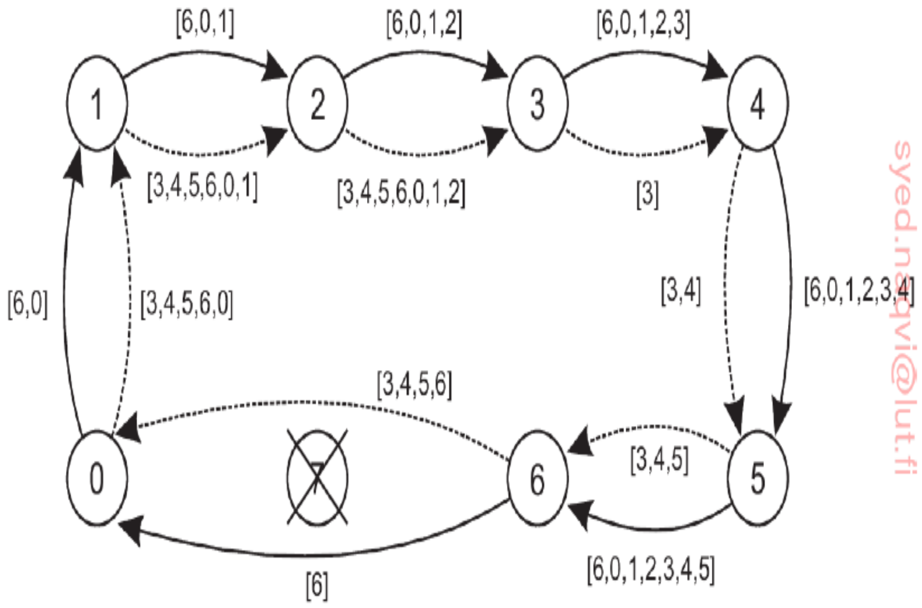
- If a process that was previously down comes back up, it holds an election
- If it happens to be the highest-numbered process currently running, it will win the election and take over the coordinator’s job
- The biggest guy in town always wins, hence the name “bully algorithm”

A ring algorithm

- Election algorithm that is based on the use of a (logical) ring
- When any process notices that the coordinator is not functioning, it builds an ELECTION message containing its own process identifier and sends the message to its successor
- If the successor is down, the sender skips over the successor and goes to the next member along the ring
- At each step along the way, the sender adds its own identifier to the list in the message effectively making itself a candidate to be elected as coordinator

A ring algorithm

- Eventually, the message gets back to the process that started it all. That process recognizes this event when it receives an incoming message containing its own identifier
- At that point, the message type is changed to COORDINATOR and circulated once again to inform everyone else who the coordinator is (the list member with the highest identifier) and who the members of the new ring are


[6,0,1]
[6,0,1,2]
[6,0,1,2,3]
1
2
3
4
[3,4,5,6,0,1]
[3,4,5,6,0,1,2]
[3]
[6,0]
[3,4,5,6,0]
[3,4]
[6,0,1,2,3,4]
[3,4,5,6]
0
7
6
5
[3,4,5]
[6]
[6,0,1,2,3,4,5]