- # Election Algorithms

Distributed system is a collection of independent computers that do not share their memory. Each processor has its own memory and they communicate via communication networks.

Communication in networks is implemented in a process on one machine communicating with a process on another machine. Many algorithms used in distributed systems require a coordinator that performs functions needed by other processes in the system. Election algorithms are designed to choose a coordinator.

Election algorithms choose a process from a group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected by another processor. Election algorithm basically determines where a new copy of coordinator should be restarted.

Election algorithm assumes that every active process in the system has a unique priority number. The process with highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has the highest priority number.

Then this number is sent to every active process in the distributed system. We have two election algorithms for two different configurations of distributed systems.

Distributed algorithms require one process to act as a coordinator or initiator. To decide which process becomes the coordinator different types of algorithms are used.

Election algorithms are meant for electing a coordinator process from among the currently running processes in such a manner that at any instance of time there is a single coordinator for all processes in the system.

The goal of an election algorithm is to ensure that when an election starts it concludes with all the processes agreeing on who the coordinator should be.

Therefore, whenever initiated, an election algorithm basically finds out which of the currently active processes has the highest priority number and then informs this to all other active processes.

1. The Bully Algorithm

2. The Ring Algorithm

1. Bully Algorithm :-

This algorithm was proposed by Garcia-Molina.

When the process notices that the coordinator is no longer responding to requests, it initiates an election. A process, P, holds an election as follows:

(I) P sends an ELECTION message to all processes with higher numbers.

(II) If no one responds, P wins the election and becomes the coordinator.

(III) If one of the higher-ups answers, it takes over. P's job is done.

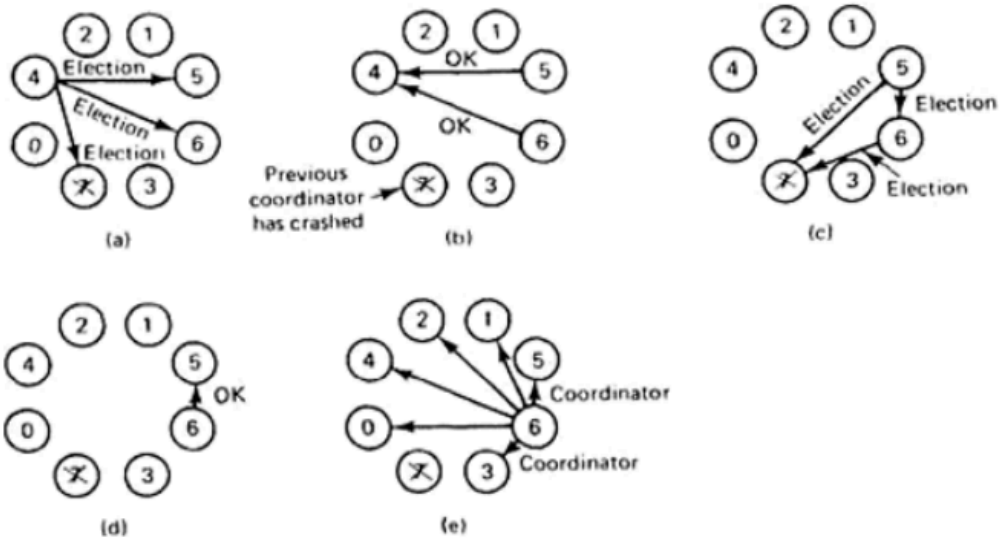
a. A process can get an ELECTION message at any time from one of its lower numbered colleagues.

b. 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.

c. All processes give up except one that is the new coordinator. It announces its victory by sending all processes a message telling them that starting immediately it is the new coordinator.

d. If a process that was previously down comes back up, it holds an election. If it happens to the highest numbered process currently running, it will win the election and take over the coordinator's job. Thus the biggest guy in town always wins, hence the name "bully algorithm".

e. Example:



In fig(a) a group of eight processes taken is numbered from 0 to 7. Assume that previously process 7 was the coordinator, but it has just crashed. Process 4 notices if first and sends ELECTION messages to all the processes higher than it that is 5, 6 and 7.

In fig (b) processes 5 and 6 both respond with OK. Upon getting the first of these responses, the process job is over. It knows that one of these will become the coordinator. It just sits back and waits for the winner.

In fig(c), both 5 and 6 hold elections by each sending messages to those processes higher than itself. In fig(d), process 6 tells 5 that it will take over with an OK message. At this point 6 knows that 7 is dead and that (6) it is the winner. If there is state information to be collected from disk or elsewhere to pick up where the old coordinator left off, 6 must now do what is needed. When it is ready to take over, 6 announce this by sending a COORDINATOR message to all running processes. When 4 gets this message, it can now continue with the operation it was trying to do when it discovered that 7 was dead, but using 6 as the coordinator this time. In this way the failure is handled and the work can continue.

If process 7 is ever restarted, it will just send all the others a COORDINATOR message and bully them into submission.

2. Ring Algorithm :-

This algorithm uses a ring for its election but does not use any token. In this algorithm it is assumed that the processes are physically or logically ordered so each processor knows its successor.

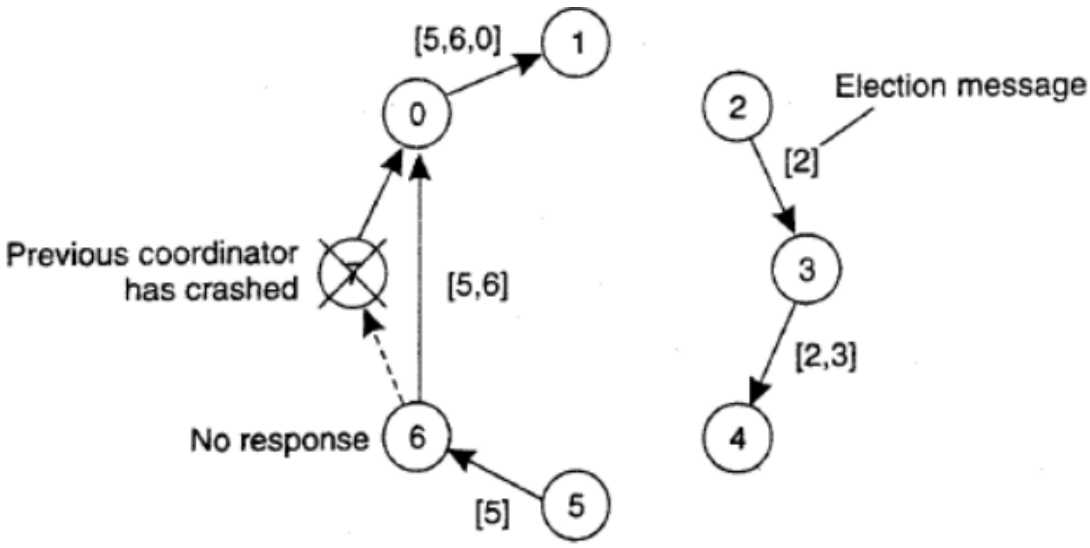
1. When any process notices that a coordinator is not functioning, it builds an ELECTION message containing its own process number and sends the message to its successor. If the successor is down the sender skips over the successor and goes to the next member along the ring until a process is located.

2. At each step the sender adds its own process number to the list in the message making itself a candidate to elected as coordinator

3. The message gets back to the process that started it and recognizes this event as the message consists of its own process number.

4. At that point the message type is changed to COORDINATOR and circulated once again to inform everyone who the coordinator is and who are the new members. The coordinator is selected with the process having the highest number.

5. When this message is circulated once it is removed and normal work is preceded.



- # Mutual Exclusion,Distributed Mutual Exclusion-Classification of Mutual Exclusion Algorithm, Requirements of Mutual Exclusion Algorithms, Performance measure

Mutual exclusion is a concurrency control property which is introduced to prevent race conditions.

It is the requirement that a process can not enter its critical section while another concurrent process is currently present or executing in its critical section i.e only one process is allowed to execute the critical section at any given instance of time.

Requirements of Mutual exclusion Algorithm:

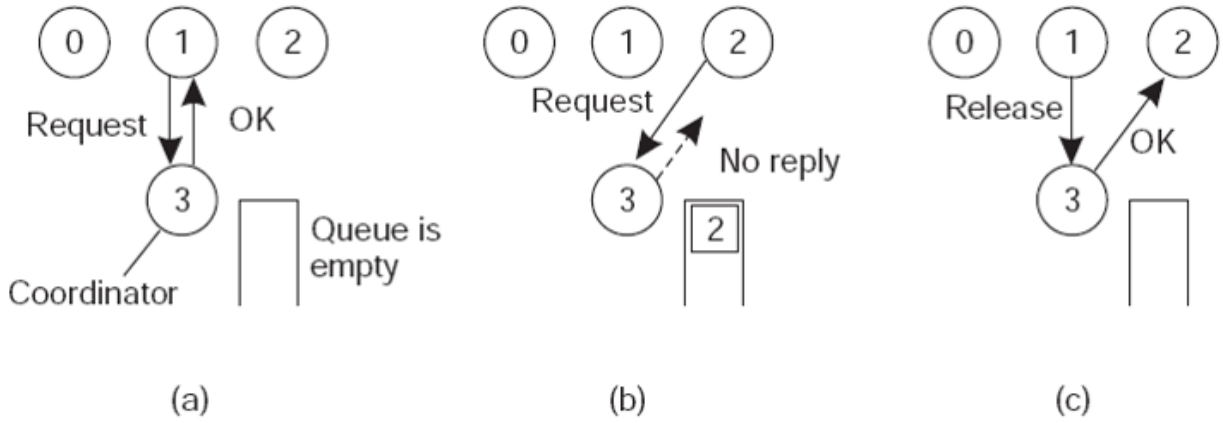
- No Deadlock: Two or more sites should not endlessly wait for any message that will never arrive.
- No Starvation: Every site who wants to execute a critical section should get an opportunity to execute it in finite time. Any site should not wait indefinitely to execute critical section while other site are repeatedly executing critical section
- Fairness: Each site should get a fair chance to execute a critical section. Any request to execute a critical section must be executed in the order they are made i.e Critical section execution requests should be executed in the order of their arrival in the system.
- 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

- A Centralized Algorithm
- A Decentralized Algorithm
- A Distributed Algorithm
- A Token Ring Algorithm

A Centralized Algorithm

In a centralized algorithm one process is elected as the coordinator which may be the machine with the highest network address.



- Whenever a process wants to enter a critical region, it sends a request message to the coordinator stating which critical region it wants to enter and asking for permission. If no other process is currently in that critical region, the coordinator sends back a reply granting permission, as shown in Fig (a). When the reply arrives, the requesting process enters the critical region.
- Suppose another process 2 shown in Fig (b), asks for permission to enter the same critical region. Now the coordinator knows that a different process is already in the critical region, so it cannot grant permission. The coordinator just refrains from replying, thus blocking process 2, which is waiting for a reply or it could send a reply saying 'permission denied.'
- When process 1 exits the critical region, it sends a message to the coordinator releasing its exclusive access as shown in Fig (c).
- The coordinator takes the first item off the queue of deferred requests and sends that process a grant message. If the process is still blocked it unlocks and enters the critical region.
- If an explicit message has already been sent denying permission, the process will have to poll for incoming traffic or block later. When it sees the grant, it can enter the critical region.
- Advantages:

- Algorithms guarantee mutual exclusion by letting one process at a time into each critical region.
- It is also fair as requests are granted in the order in which they are received.
- No process ever waits forever so no starvation.
- Easy to implement so it requires only three messages per use of a critical region (request, grant, release).
- Used for more general resource allocation rather than just managing critical regions.

- Disadvantages:
  - The coordinator is a single point of failure, the entire system may go down if it crashes.
  - If processes normally block after making a request, they cannot distinguish a dead coordinator from "permission denied" since no message comes back.
  - In a large system a single coordinator can become a performance bottleneck.

A **Decentralized Algorithm**

Each resource is assumed to be replicated n times. Every replica has its own coordinator for controlling the access by concurrent processes.

However, whenever a process wants to access the resource, it will simply need to get a majority vote.

This scheme essentially makes the original centralized solution less vulnerable to failures of a single coordinator.

A Distributed Algorithm

When a process wants to access a shared resource, it builds a message containing the name of the resource, its process number, and the current (logical) time. It then sends the message 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, the action it takes depends on its own state with respect to the resource named in the message**.**

Three different cases have to be clearly distinguished:

- 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. Instead, it 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 me. one contained in the message that it has sent everyone. The lowest one wins. If the incoming message has a lower timestamp, the receiver sends back an OK message. If its own message has a lower timestamp, the receiver queues the incoming request and sends nothing.

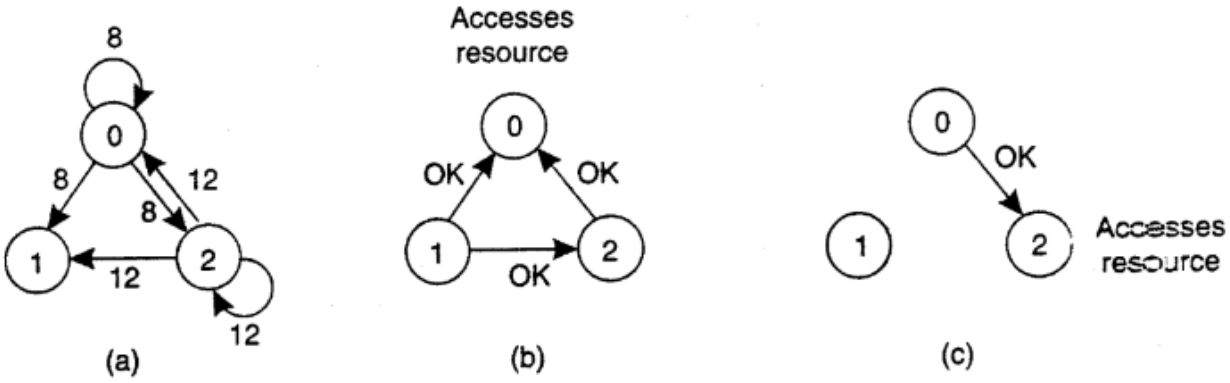


Figure 6-15. (a) Two processes want to access a shared resource at the same moment. (b) Process 0 has the lowest timestamp. so it wins. (c) When process 0 is done, it sends an *OK* also, so 2 can now go ahead.

A Token Ring Algorithm

Here we have a bus network, as shown in Fig. 6-16(a), (e.g., Ethernet), with no inherent ordering of the processes.

In software, a logical ring is constructed in which each process is assigned a position in the ring, as shown in Fig. 6-16(b).

The ring positions may be allocated in numerical order of network addresses or some other means.

It does not matter what the ordering is. All that matters is that each process knows who is next in line after itself.

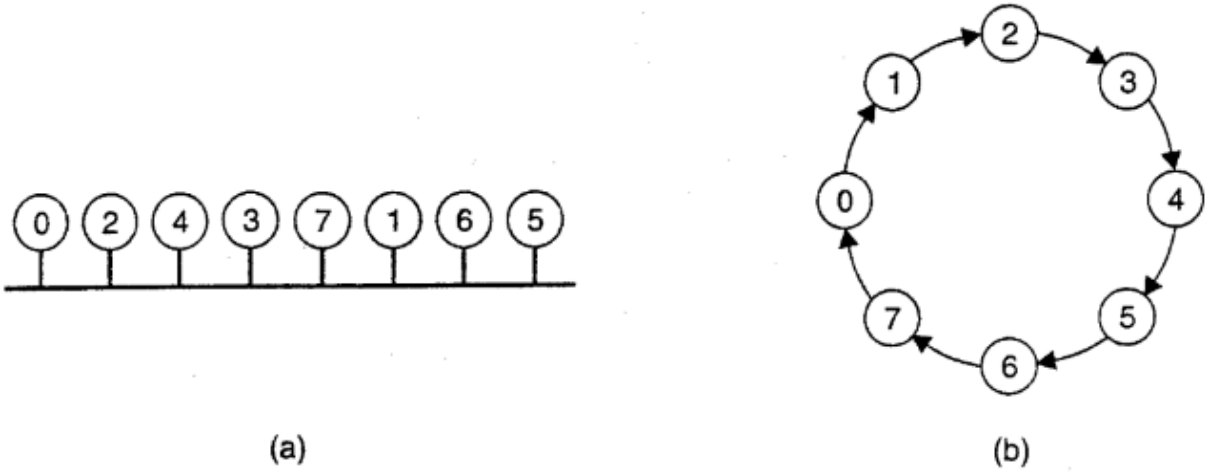


Figure 6-16. (a) An unordered group of processes on a network.. (b) A logical ring constructed in software.

In this algorithm it is assumed that all the processes in the system are organized in a logical ring.

The ring positions may be allocated in numerical order of network addresses and is unidirectional in the sense that all messages are passed only in clockwise or anti-clockwise direction.

When a process sends a request message to the current coordinator and does not receive a reply within a fixed timeout, it assumes the coordinator has crashed. It then initializes the ring and process Pi is given a token.

The token circulates around the ring. It is passed from process k to k+1 in point to point messages. When a process acquires the token from its neighbor it checks to see if it is attempting to enter a critical region. If so, the process enters the region, does all the execution and leaves the region. After it has exited it passes the token along the ring. It is not permitted to enter a second critical region using the same token.

If a process is handed the token by its neighbor and is not interested in entering a critical region it just passes along. When no processes want to enter any critical regions the token just circulates at high speed around the ring.

Only one process has the token at any instant so only one process can actually be in a critical region. Since the token circulates among the process in a well-defined order, starvation cannot occur.

Once a process decides it wants to enter a critical region, at worst it will have to wait for every other process to enter and leave one critical region.

The disadvantage is that if the token is lost it must be regenerated. But the detection of lost tokens is difficult. If the token is not received for a long time it might not be lost but is in use.