**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**
**DATA SCIENCE**
**UNIT TEST-I**

**Class: TE** **Semester:VI** **Subject: CSDLO6012-DC**

**Date:22/02/2024** **Time:02.00-03.30** **Max marks: 40**

**Note the following instructions**

1. **Attempt all questions.**
2. **Draw neat diagrams wherever necessary.**
3. **Write everything in Black ink (no pencil) only.**
4. **Assume data, if missing, with justification.**

| Q.N | Questions | MARKS | CO | Bloom's Taxonomy Level | PO2 |
|---|---|---|---|---|---|
| **Q.1.** | **Attempt any two.** | **[10]** | | | |
| i | Summarize characteristics, Advantages, disadvantages and applications of distributed systems. | [5] | CO1 | L2 | |
| ii | Illustrate various architectural models in a distributed system with neat diagrams. | [5] | CO1 | L2 | |
| iii | Differentiate multiprocessor and multicomputer systems. | [5] | CO1 | L2 | |
| iv | Compare Network OS, Distributed OS and middleware in the design of distributed systems. | [5] | CO1 | L2 | |
| **Q.2.** | **Attempt any one** | **[10]** | | | |
| i | Construct a distributed Communication model by selecting appropriate components of Remote Procedure Call (RPC)? | [10] | CO2 | L2 | |
| ii | Apply Stream Oriented communication principles to organize RSVP for resource reservation in a distributed system. | [10] | CO2 | L2 | |
| Q.3. | **Attempt any two.** | **[20]** | | | |
| i | Apply Happened-Before relationship for synchronizing logical clock with an example. | [10] | CO3 | L2 | |

| | | | | | |
|---|---|---|---|---|---|
| ii | Identify the requirements of the election algorithm in a distributed system. Articulate different forms of election algorithms in detail with an example. | [10] | CO3 | L2 | |
| iii | Apply Maekwa's algorithm to achieve mutual exclusion. | [10] | CO3 | L2 | |

**Q.1 Attempt any two.** **10M**

1. Summarize characteristics, Advantages, disadvantages and applications of distributed systems. 5M

   Answer:

   Characteristics of Distributed System

   - Resource Sharing: It is the ability to use any Hardware, Software, or Data anywhere in the System.
   - Openness: It is concerned with Extensions and improvements in the system (i.e., How openly the software is developed and shared with others)
   - Concurrency: It is naturally present in Distributed Systems, that deal with the same activity or functionality that can be performed by separate users who are in remote locations. Every local system has its independent Operating Systems and Resources.
   - Scalability: It increases the scale of the system as a number of processors communicate with more users by accommodating to improve the responsiveness of the system.
   - Fault tolerance: It cares about the reliability of the system if there is a failure in Hardware or Software, the system continues to operate properly without degrading the performance the system.
   - Transparency: It hides the complexity of the Distributed Systems to the Users and Application programs as there should be privacy in every system.
   - Heterogeneity: Networks, computer hardware, operating systems, programming languages, and developer implementations can all vary and differ among dispersed system components.

   Advantages of Distributed System

   - Applications in Distributed Systems are Inherently Distributed Applications.
   - Information in Distributed Systems is shared among geographically distributed users.
   - Resource Sharing (Autonomous systems can share resources from remote locations).
   - It has a better price performance ratio and flexibility.

- It has shorter response time and higher throughput.
- It has higher reliability and availability against component failure.
- It has extensibility so that systems can be extended in more remote locations and also incremental growth.

Disadvantages of Distributed System

- Relevant Software for Distributed systems does not exist currently.
- Security possess a problem due to easy access to data as the resources are shared to multiple systems.
- Networking Saturation may cause a hurdle in data transfer i.e., if there is a lag in the network then the user will face a problem accessing data.
- In comparison to a single user system, the database associated with distributed systems is much more complex and challenging to manage.
- If every node in a distributed system tries to send data at once, the network may become overloaded.

Applications of Distributed System

- Finance and Commerce: Amazon, eBay, Online Banking, E-Commerce websites.
- Information Society: Search Engines, Wikipedia, Social Networking, Cloud Computing.
- Cloud Technologies: AWS, Salesforce, Microsoft Azure, SAP.
- Entertainment: Online Gaming, Music, youtube.
- Healthcare: Online patient records, Health Informatics.
- Education: E-learning.
- Transport and logistics: GPS, Google Maps.
- Environment Management: Sensor technologies.

2. Illustrate various architectural models in a distributed system with neat diagrams. 5M

Answer:

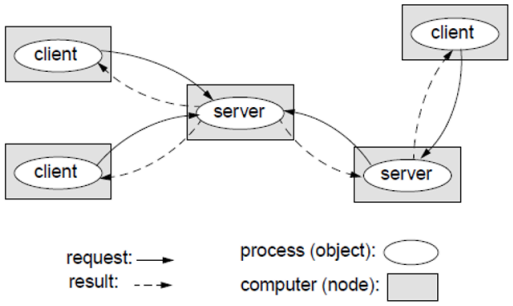
Architectural model describes responsibilities distributed between system components and how these components are placed.

a)Client-server model

☞ The system is structured as a set of processes, called servers, that offer services to the users, called clients.

The client-server model is usually based on a simple request/reply protocol, implemented with send/receive primitives or using remote procedure calls (RPC) or remote method invocation (RMI):

The client sends a request (invocation) message to the server asking for some service; The server does the work and returns a result (e.g. the data requested) or an error code if the work could not be performed.

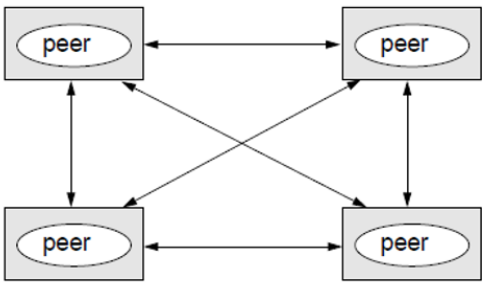



A server can itself request services from other servers; thus, in this new relation, the server itself acts like a client.

b)Peer-to-peer

☞ All processes (objects) play similar role.

Processes (objects) interact without particular distinction between clients and servers.
The pattern of communication depends on the particular application.
A large number of data objects are shared; any individual computer holds only a small part of the application database.
Processing and communication loads for access to objects are distributed across many computers and access links.
This is the most general and flexible model.




Peer-to-Peer tries to solve some of the above
It distributes shared resources widely -> share computing and communication loads.
☞ Problems with peer-to-peer:
High complexity due to

- Cleverly place individual objects
- retrieve the objects

- maintain a potentially large number of replicas.

3. Differentiate multiprocessor and multicomputer systems. 5M
   Answer:

| Multiprocessor | Multicomputer |
|---|---|
| Multiprocessor consists of multiple processors within a single computer. | Multicomputer is an interlinked multiple autonomous computer. |
| Multiprocessor is a singly shared memory that is attached to the elements being processed. | The memory attached to the processing elements are distributed in multiples. |
| Multiprocessor is necessary for the processing elements to communicate with each other. | Multicomputer is not required for elements being processed to communicate. |
| Multiprocessor is a dynamic network. | Multicomputer is a type of static network. |
| Example of multiprocessor is a sequent symmetry S-81. | Example of a multicomputer is a message passing multicomputer. |

4. Compare Network OS, Distributed OS and middleware in the design of distributed systems. 5M
   Answer:
   Network OS:
   A Network OS is a type of operating system that is used to run a system software on a server and allow the server to manage the users, groups, data, security, applications and other networking operations. It is considered as the primary form of an operating system for the distributed computer architectures.
   A network operating system permits resource sharing among two or more computers that are operating under their own operating systems. However, it cannot control the utilization of resources and hence it causes the improper distribution of resources. Also, there is no provision of fault tolerance.
   A network operating system provides the necessary functions for managing the network, such as managing network resources, controlling access to the network, and establishing communication between devices on the network.
   Distributed OS:
   A Distributed operating system is an operating system that runs on multiple machines and provides the appearance of a single system to the users. It is designed to allow multiple computers to work together as a single system, with each machine in the system running its own instance of the operating system and contributing its own resources to the system.
   The primary objective of a distributed operating system is to introduce transparency in the system. In a distributed OS, the utilization of multiple hardware resources of computers is hidden from users. It is less autonomous because the operating system has complete control on the resources.

A distributed operating system dynamically allocates processes to random CPUs and the storage of files is also managed by the operating system which means the user does not know which hardware is being utilized for the processing and for storing the files.

Middleware:

In distributed systems, middleware is a software component that provides services between two or more applications and can be used by them. Middleware can be thought of as an application that sits between two separate applications and provides service to both. In this article, we will see a role of middleware in distributed systems. With all the miscommunication going around these days, it is vital for enterprises to start using software solutions that streamline communications across departments. One such product that fits this description, is known as Middleware, which allows organizations to implement their processes seamlessly by integrating all components of the enterprise. In this article, we will examine the role of middleware in distributed systems and how it can help businesses to evolve.

However, there are numerous middlewares around (for a list of some well-known middlewares see Related topics), each having its own set of responsibilities. This can be attributed to the fact that different types of middleware are designed for different scenarios and each type has its own set of requirements and goals.

**Q.2 Attempt any one. 10M**

1. Construct a distributed Communication model by selecting appropriate components of Remote Procedure Call (RPC)? 10M

   Answer:

   RPC is a powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional, or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers of distributed applications avoid the details of the interface with the network. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports.

   RPC makes the client/server model of computing more powerful and easier to program.

   Working:

   Each RPC occurs in the context of a thread. A thread is a single sequential flow of control with one point of execution at any instant. A thread created and managed by application code is an application thread.

RPC applications use application threads to issue both RPCs and RPC run-time calls. An RPC client contains one or more client application threads, each of which may perform one or more RPCs.

In addition, for executing called remote procedures, an RPC server uses one or more call threads that the RPC run-time system provides. When beginning to listen, the server application thread specifies the maximum number of concurrent calls it will execute.

Single-threaded applications have a maximum of one call thread. The maximum number of call threads in multi-threaded applications depends on the design of the application and RPC implementation policy. The RPC run-time system creates the call threads in the server execution context.

An RPC extends across client and server execution contexts. Therefore, when a client application thread calls a remote procedure, it becomes part of a logical thread of execution known as an RPC thread.

An RPC thread is a logical construct that encompasses the various phases of an RPC as it extends across actual threads of execution and the network.

After making an RPC, the calling client application thread becomes part of the RPC thread. Usually, the RPC thread maintains execution control until the call returns.

The RPC thread of a successful RPC moves through the execution phases illustrated in Execution Phases of an RPC Thread.

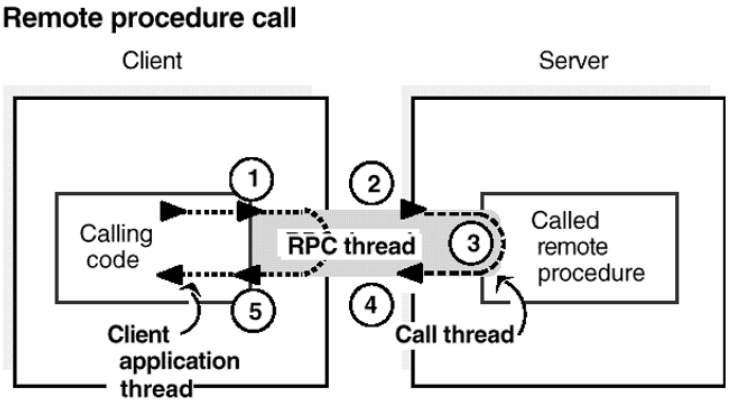



Figure 6-1 Execution Phases of an RPC Thread

The execution phases of an RPC thread, as shown in Execution Phases of an RPC Thread include the following:

The RPC thread begins in the client process, as a client application thread makes an RPC to its stub; at this point, the client thread becomes part of the RPC thread.

The RPC thread extends across the network to the server.

The RPC thread extends into a call thread, where the remote procedure executes. While a called remote procedure is executing, the call thread becomes part of the RPC

thread. When the call finishes executing, the call thread ceases being part of the RPC thread.

The RPC thread then retracts across the network to the client.

When the RPC thread arrives at the calling client application thread, the RPC returns any call results and the client application +thread ceases to be part of the RPC thread.

Concurrent Call Threads Executing in Shared Execution Context shows a server executing remote procedures in its two call threads, while the server application thread listens.

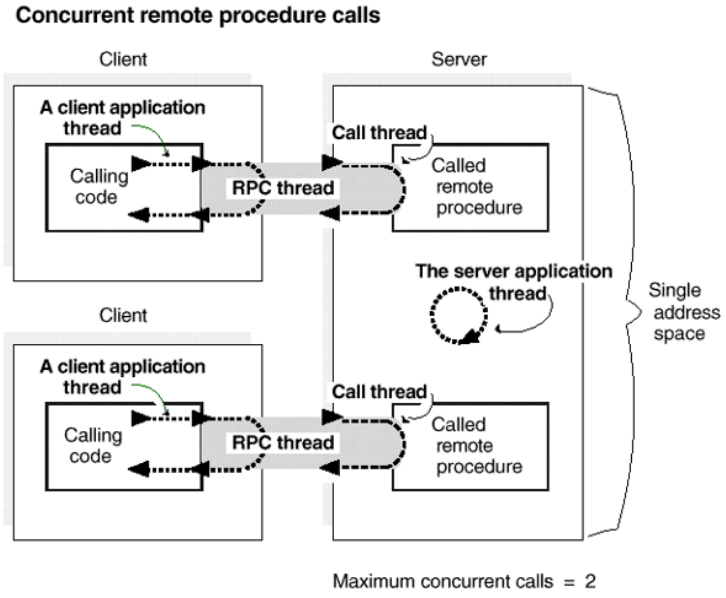



Figure 6-2 Concurrent Call Threads Executing in Shared Execution Context

2. Apply Stream Oriented communication principles to organize RSVP for resource reservation in a distributed system. 10M

Answer:

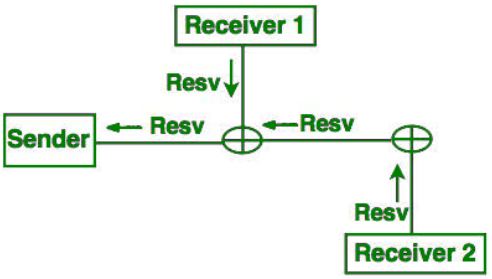
Resource Reservation Protocol (RSVP) is used in real-time systems for an efficient quality band transmission to a particular receiver. It is generally used by the receiver side for the fast delivery of the transmission packets from the sender to the receiver.

Flowspec is used in RSVP to determine the different parameters like bandwidth, link strength, congestion, etc for smooth and collisionless communication and data transfer. Filterspec is used in RSVP for filtering of the packets. It is used to route the packets according to its destination type as a fixed or shared receiver.

Types of messages used in RSVP connection establishment:

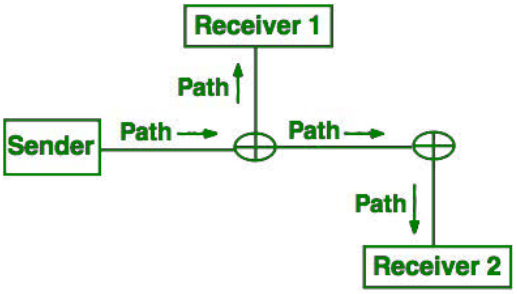
- Reservation Message (resv): The receiver sends the Reservation Message (resv) to the sender, which specifies all the required resources and parameters for the reservation to establish.
- Path Message (path): Upon receiving the reservation message from the receiver, the sender records all the necessary resources to be reserved and records the path. The sender multicasts a Path Message (path) to all the receivers, which specifies the routing details of the packet. It also contains all the necessary specifications about the reservation to be made for the receiver.

Reservation Messages Transfer:




Resv Messages

Path Messages Transfer:




Path messages

The data sent by the sender in Resource Reservation Protocol is encrypted to prevent the data sent to the receiver from breach. Error reporting is done at the sender side in Resource Reservation Protocol for making the necessary changes in the communication strategy. In case of failure in RSVP, the admission state of the hop is sent to the requester for handling the necessary packets. In RARP the routers record the necessary forward and reverse routing state. The router may also make the necessary changes in the path message sent by the sender to the receiver to indicate the actual resource availability.

Advantages of Resource Reservation Protocol (RRP) in Real-Time Systems:

- Predictable Resource Allocation: RRP allows for the reservation of system resources in advance, ensuring predictable and guaranteed resource

availability for real-time tasks. This enables the system to meet timing constraints and ensure timely execution of critical tasks.

- Deterministic Behavior: RRP provides deterministic behavior in real-time systems by allocating fixed resources to specific tasks. This determinism allows for precise timing analysis and ensures that tasks can meet their deadlines with high accuracy.
- Quality of Service Guarantees: RRP enables the system to provide quality of service guarantees by reserving resources based on the requirements of real-time tasks. It ensures that tasks receive the necessary resources and can operate within specified timing constraints, leading to reliable system behavior.
- Resource Optimization: RRP allows for efficient resource utilization by reserving resources only when they are needed. This prevents resource wastage and maximizes resource availability for other tasks, leading to overall improved system performance.

Disadvantages of Resource Reservation Protocol (RRP) in Real-Time Systems:

- Increased Resource Overhead: Implementing RRP requires additional resources for reservation and management, including memory for storing reservation information, computational power for scheduling and resource allocation decisions, and communication overhead for maintaining the reservation protocol. This can increase the overall resource overhead of the system.
- Limited Flexibility: RRP relies on static resource reservations, which may limit the flexibility and adaptability of the system. Changes in task requirements or dynamic workload variations may require manual reconfiguration of reservations, leading to inflexibility in resource allocation.
- Scalability Challenges: RRP may face scalability challenges in large-scale real-time systems with a high number of tasks and complex resource dependencies. As the number of tasks and resource requirements increase, managing and coordinating the reservations can become more challenging and may impact system performance.
- Reservation Conflicts and Deadlock Risks: In systems where multiple tasks contend for the same resources, conflicts, and deadlocks may arise if the reservation protocol is not carefully designed and managed. Resolving conflicts and avoiding deadlocks requires careful coordination and scheduling decisions, which can add complexity to the system design.

**Q.3 Attempt any two.** **20M**

1. Apply Happened-Before relationship for synchronizing logical clock with an example. 10M

   Answer:

   Lamport's Logical Clock was created by Leslie Lamport. It is a procedure to determine the order of events occurring. It provides a basis for the more advanced Vector Clock Algorithm. Due to the absence of a Global Clock in a Distributed Operating System Lamport Logical Clock is needed.

   Algorithm:

   - Happened before relation(->): a -> b, means 'a' happened before 'b'.
   - Logical Clock: The criteria for the logical clocks are:
     - [C1]: Ci (a) < Ci(b), [ Ci -> Logical Clock, If 'a' happened before 'b', then time of 'a' will be less than 'b' in a particular process. ]
     - [C2]: Ci(a) < Cj(b), [ Clock value of Ci(a) is less than Cj(b) ]

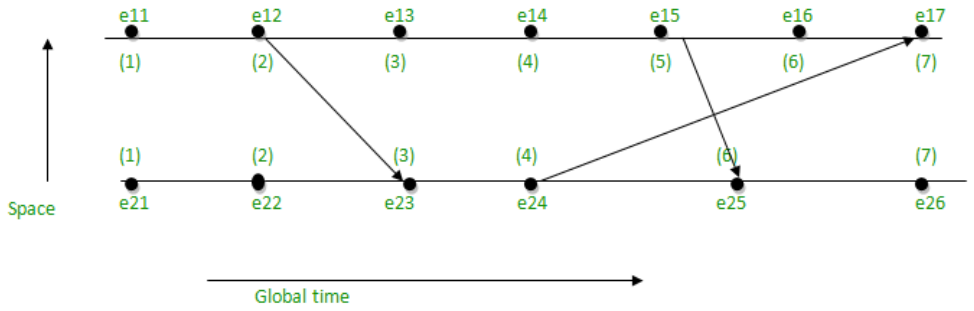
   Reference:

   - Process: Pi
   - Event: Eij, where i is the process in number and j: jth event in the ith process.
   - tm: vector time span for message m.
   - Ci vector clock associated with process Pi, the jth element is Ci[j] and contains Pi's latest value for the current time in process Pj.
   - d: drift time, generally d is 1.

   Implementation Rules[IR]:

   - [IR1]: If a -> b ['a' happened before 'b' within the same process] then, Ci(b) =Ci(a) + d
   - [IR2]: Cj = max(Cj, tm + d) [If there's more number of processes, then tm = value of Ci(a), Cj = max value between Cj and tm + d]

   For Example:




   Take the starting value as 1, since it is the 1st event and there is no incoming value at the starting point:

   - e11 = 1

- e21 = 1

The value of the next point will go on increasing by d (d = 1), if there is no incoming value i.e., to follow [IR1]

- e12 = e11 + d = 1 + 1 = 2
- e13 = e12 + d = 2 + 1 = 3
- e14 = e13 + d = 3 + 1 = 4
- e15 = e14 + d = 4 + 1 = 5
- e16 = e15 + d = 5 + 1 = 6
- e22 = e21 + d = 1 + 1 = 2
- e24 = e23 + d = 3 + 1 = 4
- e26 = e25 + d = 6 + 1 = 7

When there will be incoming value, then follow [IR2] i.e., take the maximum value between Cj and Tm + d

- e17 = max(7, 5) = 7, [e16 + d = 6 + 1 = 7, e24 + d = 4 + 1 = 5, maximum among 7 and 5 is 7]
- e23 = max(3, 3) = 3, [e22 + d = 2 + 1 = 3, e12 + d = 2 + 1 = 3, maximum among 3 and 3 is 3]
- e25 = max(5, 6) = 6, [e24 + 1 = 4 + 1 = 5, e15 + d = 5 + 1 = 6, maximum among 5 and 6 is 6]

2. Identify the requirements of the election algorithm in a distributed system. Articulate different forms of election algorithms in detail with an example. 10M

Answer:

Distributed algorithms require one process to act as a coordinator or initiator. To decide which process becomes the coordinator 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.

<u>i.Bully Algorithm</u>

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:

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

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

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

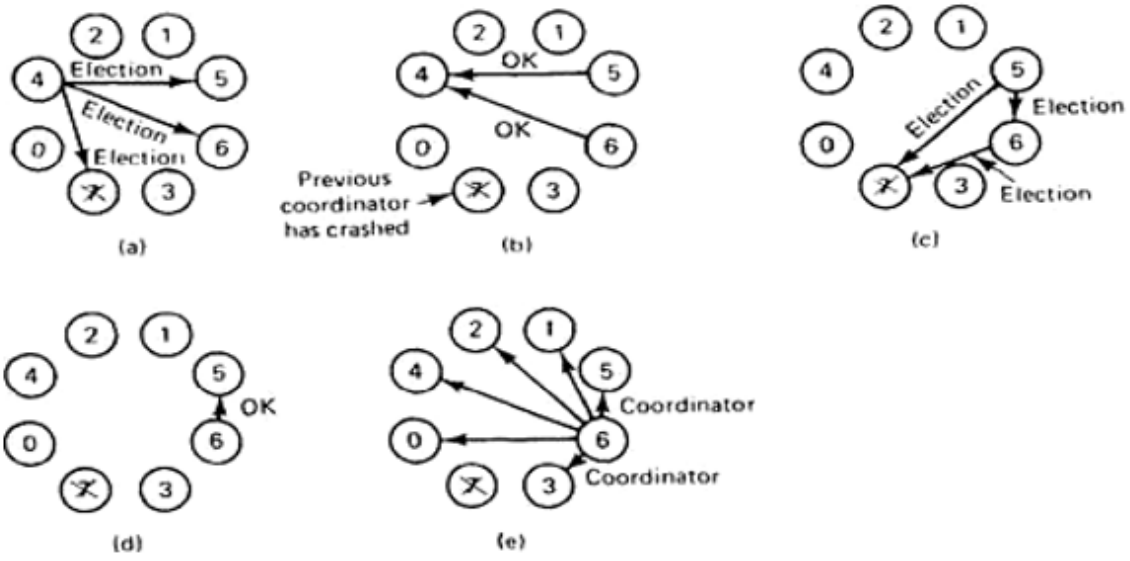
A process can get an ELECTION message at any time from one of its lower numbered colleagues.

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.

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.

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".

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, process4job 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 6knows that 7 is dead and that (6) it is the winner. It 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 of 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.

ii.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.

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.

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

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

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 highest number.

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

3. Apply Maekwa's algorithm to achieve mutual exclusion. 10M

Answer:

Maekawa's Algorithm is a quorum based approach to ensure mutual exclusion in distributed systems. As we know, In permission based algorithms like Lamport's Algorithm, Ricart-Agrawala Algorithm etc. a site request permission from every other site but in quorum based approach, A site does not request permission from every other site but from a subset of sites which is called quorum. In this algorithm:

Three type of messages ( REQUEST, REPLY and RELEASE) are used.

A site send a REQUEST message to all other site in its request set or quorum to get their permission to enter critical section.

A site send a REPLY message to requesting site to give its permission to enter the critical section.

A site send a RELEASE message to all other site in its request set or quorum upon exiting the critical section.

Algorithm:

To enter Critical section:

- When a site Si wants to enter the critical section, it sends a request message REQUEST(i) to all other sites in the request set Ri.
- When a site Sj receives the request message REQUEST(i) from site Si, it returns a REPLY message to site Si if it has not sent a REPLY message to the site from the time it received the last RELEASE message. Otherwise, it queues up the request.

To execute the critical section:

- A site Si can enter the critical section if it has received the REPLY message from all the site in request set Ri
- To release the critical section:
- When a site Si exits the critical section, it sends RELEASE(i) message to all other sites in request set Ri
- When a site Sj receives the RELEASE(i) message from site Si, it send REPLY message to the next site waiting in the queue and deletes that entry from the queue
- In case queue is empty, site Sj update its status to show that it has not sent any REPLY message since the receipt of the last RELEASE message

Drawbacks of Maekawa's Algorithm:

This algorithm is deadlock prone because a site is exclusively locked by other sites and requests are not prioritized by their timestamp.

Performance:

- Synchronization delay is equal to twice the message propagation delay time
- It requires $3\sqrt{n}$ messages per critical section execution.

Advantages of Maekawa's Algorithm:

- Low Message Complexity: Maekawa's algorithm requires only O(sqrt(N)) messages per critical section invocation, where N is the total number of processes in the system. This makes it more efficient than many other distributed mutual exclusion algorithms.
- Scalability: The algorithm is highly scalable and can be used in large-scale distributed systems without any problems.
- Fairness: The algorithm provides fairness by allowing only one process per group to enter the critical section. It guarantees that every process will eventually get a chance to enter the critical section.

Disadvantages of Maekawa's Algorithm:

- High Overhead: The algorithm requires a lot of overhead to maintain the group membership information and to update it when a process joins or

leaves the system. This can lead to increased network traffic and latency.

- Limited Flexibility: The algorithm requires a fixed number of groups to be defined before the system starts running. This makes it less flexible than other algorithms that can dynamically adjust to changes in the system.
- Delayed Execution: The algorithm can lead to delays in the execution of critical sections because a process may have to wait for messages to arrive from other groups before entering the critical section.