UNIT – 01

DISTRIBUTED SYSTEM

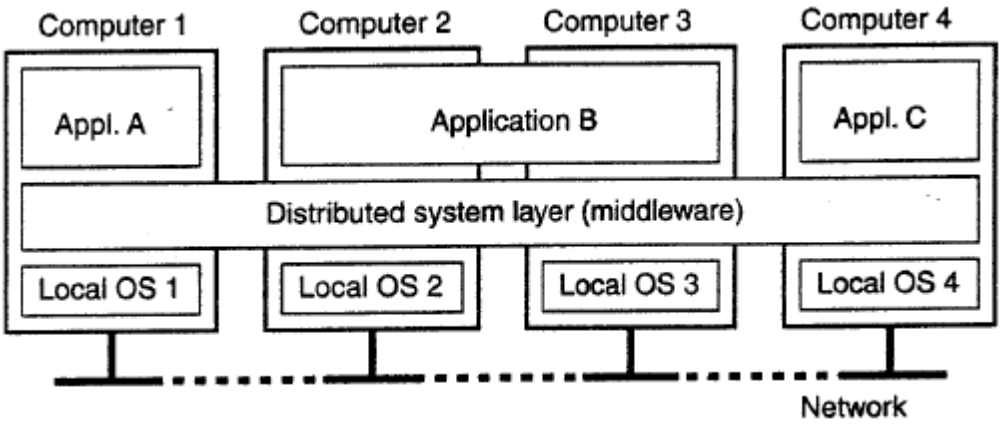
UNIT-01/LECTURE-01

Introduction ([RGPV/ Dec 2011 (7)]

The definition of distributed system is given below-

"A distributed system is a collection of independent computers that appears to its users as a single coherent system".

The definition has two aspects. The first one deals with hardware-the machine are autonomous. The second one deals with software-the users think they are dealing with a single system. Both are essential. The distributed system has two important characteristics, first is that the difference between the various computers and the ways in which they communicate are hidden from users and other important characteristic is that users and applications can interact with a distributed system in a consistent and inform way regardless of where and when interaction takes place



Distributed systems should also be relatively easy to expand or scale. This characteristic is a direct consequence of having independent computer but at the same time, hiding how these computers take part in the system as a whole. A distributed system will normally be continuously available although perhaps certain parts may be temporarily out of order. User and application should notice that parts are being replaced or fixed or that new parts are added to serve more users or applications.

To support heterogeneous computer and networks while offering a single system view, distributed systems are often organized by means of a layers of software that is logically placed between a higher-level layer consisting of users and applications and a underneath consisting of operating systems. Such a distributed system is sometimes called Middleware.
For example, consider a network of workstation in a university or company department. In addition to each user's personal workstation, there might be a pool of processors in the machine room that are not assigned to specific users but are allocated dynamically as needed. Such a system might have a single file system, with all files accessible from all machines in the same way and using the same path name. also when a user types a command the system could look for the best place to execute that command, possibly on the user's own workstation, possibly on an idle workstation belonging to someone else, and possible on one of the unassigned processors in the machine room. If the system as a whole looks and acts like a classical single-processor timesharing system, it qualifies as a distributed system.

A distributed system is a collection of processors that do not share memory or clock. Instead, each processor has its own local memory and the processor communicates with each other through communication lines such as local or wide-area networks. The processors in a distributed system vary in size and function. Such systems may include small handhold or relative devices, personal computers, workstations, and large mainframe computer systems.

The benefits of a distributed system include user access to the resources maintained by the system and therefore computation speedup and improved data availability and reliability. A distributed file system is a file-service system whose users, servers and storage devices are dispersed among the sites of a distributed system. Accordingly service activity has to carried out across the network, instead of a single centralized data repository, there are multiple and independent storage devices.

Because a system is distributed, however it must provide mechanisms for process synchronization and communication, for dealing with the deadlock problem and for dealing with failures that are not encountered in a centralized system.

Example of Distributed system ([RGPV/ Dec 2011 (7)]

A large bank with hundreds of branch offices all over the world. Each office has a master computer to store local accounts and handle local transactions. In addition, each computer has the ability to talk to all other branch computers and with a central computer at headquarters. If transactions can be done without regard to where a customer or account is, and the users do not notice any difference between this system and the old centralized mainframe that it replaced, it too would be considered a distributed system.

Autonomous – the machines are autonomous in distributed system

1. **Single system view-** the users think that they are dealing with single system
2. **Heterogeneous computer & network-** the workstations in distributed system may differ from each other & also in way they communicate (network)
3. **Interaction-** Interactions of computers are hidden from users
4. **Scalable-** Distributed system should be scalable
5. **Middleware-** Distributed system may be organized as a means of layer of software & placed between higher layer & underneath layer

Resource sharing and the Web Challenges

Advantages of Distributed System over Centralized System

The main purpose for the decentralization is economics. “The computing power of a CPU is proportional to the square of its price”. By paying twice as much, you could get four times the performance. This observation fit the mainframe technology of its time quite well and let most organization to buy the largest single machine they could afford. With microprocessor technology, Grosch’s law no longer holds. For a few hundred dollars you can

get a CPU chip that can execute more instructions per second that one of the largest 1980s mainframes. If you are willing to pay twice as much, you get the same CPU only running at a slightly higher clock speed. As a result the most cost effective solution is frequently to harness a large number of cheap CPUs together in a system. In effect, a distributed system gives more bang for the buck.

A slight variation on this theme is the observation that a collection of microprocessors cannot only give a better price/performance ratio than a single mainframe but may yield an absolute performance that no mainframe can achieve at any price. Another potential advantage of a distributed system over a centralized one is higher reliability. By distributing the workload over many machines, a single chip failure will bring down at most one machine, leaving the rest intact. Ideally if 5 percent of machine are down at any moment the system would be able to continue to work with a 5 percent loss is performance.

Finally increment growth is also potentially a big plus. Often a company will buy a mainframe with the intention of doing all its work on it. If the company prospers and the workload grows, at a certain point the mainframe will no longer be adequate. The only solutions are to either replace the mainframe with a larger one or add a second mainframe. Both of these can cause major havoc with the company's operations. In contrast with a distributed system, it may be possible to simply add more processor to the system, thus allowing it to expand gradually as the need arises.

Item	Description
Economics	Microprocessors offer a better price/performance than mainframe
Speed	A distributed system may have more total computing power than a mainframe
Inherent Distribution	Some applications involve spatially separated machines
Reliability	If one machine crashes, the system as a whole can still survive
Incremental Growth	Computing power can be added in small increments

Advantages of Distributed System over Centralized Ones

Advantages of Distributed System over Independent PC's

1.Data sharing- Allow many users access to a common data base

2.**Device Sharing** -Allow many users to share expensive peripherals like color printers

3.**Communication-** Make human-to-human communication easier, for example, by electronic mail.

4.**Flexibility-** Spread the workload over the available machines in the most cost effective way.

Disadvantages of Distributed Systems

Although distributed systems have their strength, they also have their weaknesses. The first problem with the distributed systems is software. With the current state of art, we do not have much experience in designing, implementing and using distributed software. As more research is done, this problem will diminish but for the moment it should not be underestimated.

A second potential problem is due to the communication network. It can lose message, which requires special software to handle, and it can become overloaded. When the network saturates it must either be replaced or a second one must be added. In both cases, some portion of one or more buildings may have to be rewired at great expense, or network interface boards may have to be replaced. Once the system comes to depend on the network, its loss or saturation can negate most of the advantages the distributed system was built to achieve.

Finally, the easy sharing of data which is the advantage of distributed systems, may turn out to be a two edged sword. If people can conveniently access data all over the system, they may be equally able to conveniently access data that they have no business looking at. In other words the security is often a problem.

The disadvantages of distributed systems are summarized in fig.

Item	Description
Software	Little software exists at present for distributed systems
Networking	The network can saturate or cause other problems
Security	Easy access also applies to secret data

Disadvantage of Distributed Systems

S.NO	RGPV QUESTIONS	Year	Marks
Q.1	What is distributed system? Give an example of distributed system.	Dec 2011	7

UNIT-01/LECTURE-02

System Models: Architectural models ([RGPV/ Dec 2013 (7)]

Client -Server

- **3-tier Architecture**
- **N-tier Architecture**
- **Tightly coupled**
- **Peer to Peer**

1. **Client-Server-** Small Clients code contacts the server for data then formats & displays it to the user. Input at the client is committed back to server when it represents a permanent change.
2. **3-tier Architecture-** 3-tier system move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are 3-tier.
3. **N-tier Architecture-** The web application which further forward their request to other enterprise services. This type of applications is the one most responsible for the success of applications servers.
4. **Tightly coupled-** It refers to a set of highly integrated machine that run the same process in parallel subdividing the task in parts that are made individually by each one & then put back together to make the final result.
5. **Peer to peer-**An architecture where there is no special machine that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines known as peers.

Limitation of Distributed system

1. Slower
2. More complicated more expensive & less robust than centralized one Inherent Limitation

3. No global clock

– Due to unpredictable message delay, it is difficult to synchronize physical clocks

– Distributed schedule is difficult to implement

4. No shared memory

– It is difficult to obtain a coherent global view, which includes local state and messages in transmission.

Distributed Shared Memory (DSM) ([RGPV/ Dec 2012 (7)], ([RGPV/ June2013 (7)]

• Traditionally, distributed computing is based on message passing.

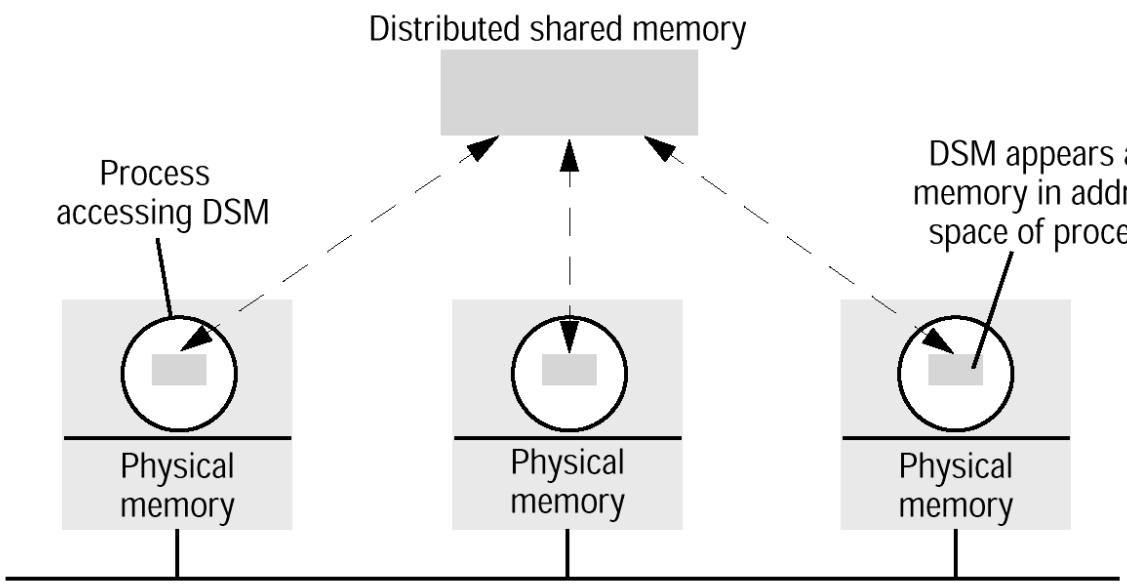
• DSM provides a virtual address space that is shared among all nodes in a distributed system.

• With DSM, programs access data in the shared address space just as they access data in traditional virtual memory.

• In DSM, each node can own data stored in the shared address space, and the ownership can change when data moves from one node to another.

• When a process accesses data in the shared address space, the DSM software layer, implemented in the kernel or as a runtime library routine, maps the shared memory address

to the physical memory.



The DSM

Advantages of DSM

• DSM hides the explicit message passing and provides a simpler abstraction for sharing data that programmers are used to.

• DSM allows complex structures (e.g., pointer, array) to be passed by reference, which is not supported by the message passing model.

• By moving the entire block or page containing the referenced data, make use of locality of reference.

• Cheaper to build than multiprocessor systems.

• Good scalability compared to multiprocessor systems.

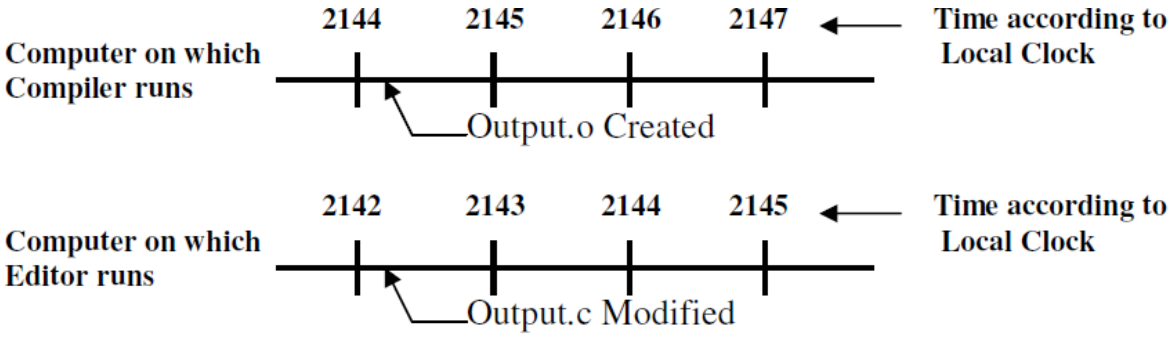
• Programs written for shared memory multiprocessors can run on DSM systems.

Clock Synchronization

In centralized system, time is unambiguous. When a process wants to know the time, it makes a system call and kernel tells it. If process **A** Asks fro the time, and then a little later process **B** asks for the time, the value that **B** gets will be higher than the value **A** got. It will certainly not be lower. In a distributed system, achieving agreement on time is not trivial. In

Unix, large programs are split into multiple source files, so that a change to one source file only requires one file to be recompiled, not all the files. If a program consists of 100 files, not having to recompile, everything because one file has been changed greatly increases the speed at which programmers can work. The way make normally works is simple. When the programmer has finished changing all the source files, he start make, which examines the times at which all the source and object files were last modified. If the source file input.c has time 2151 and the corresponding object file input.o has time 2150 make knows that input.c has been changed since input.o was created and thus input.c must be recompiled. On the other hand, if output.c has time 2144 and output.o hs tiem 2145, no compilation is needed there. Thus make goes through all the source file to find out which ones need to be recompiled and calls the compiler to recompile them.

Now imagine that there is no global agreement on time. Suppose that output.o has time 2144 as above and shortly thereafter output.c is modified but is assigned time 2143 because the clock on its machine is slightly behind, as shown in fig. make will not call the compiler. The resulting executable binary program will then contain a mixture of object files from the old sources and new sources. It will probably crash and the programmer will go crazy trying to understand what is wrong with this code.



S.NO	RGPV QUESTIONS	Year	Marks
Q.2	Why architectural model is important in the distributed system design?	Dec 2013	7
Q.1	Explain the different architecture of distributed shared memory.	June 2013	7
Q.2	What are the three main approches for designing distributed shared memory?	Dec 2012	7

UNIT-01/LECTURE-03

Summary on time and clocks in distributed systems

- The accurate timekeeping is important for distributed systems.
- algorithms (e.g. Cristian's and NTP) synchronize clocks in spite of their drift and the variability of message delays.
- for ordering of an arbitrary pair of events at different computers, clock synchronization is not always practical.
- the happened-before relation is a partial order on events that reflects a flow of information between them.
- Lamport clocks are counters that are updated according to the happened-before relationship between events.
- vector clocks are an improvement on Lamport clocks,
- we can tell whether two events are ordered by happened-before or are concurrent by comparing their vector timestamps.

Computer clocks and timing events

- Each computer in a DS has its own internal clock
  - used by local processes to obtain the value of the current time
  - processes on different computers can timestamp their events
  - but clocks on different computers may give different times
  - computer clocks drift from perfect time and their drift rates differ from one another.
  - clock drift rate: the relative amount that a computer clock differs from a perfect clock

Even if clocks on all computers in a DS are set to the same time, their clocks will eventually vary quite significantly unless corrections are applied.

The 'happened before' relation is essential to understanding logical clocks

- A distributed system is defined as a collection P of N processes p_i, i = 1,2,… N
- Each process p_i has a state s_i consisting of its variables (which it transforms as it executes)
- Processes communicate only by messages (via a network)
- Actions of processes:
  - Send, Receive, change own state
- Event: the occurrence of a single action that a process carries out as it executes e.g. Send, Receive, change state

Events at a single process p_i, can be placed in a total ordering denoted by the relation $®_i$ between the events.

Lamport's Logical Clocks([RGPV/ Dec 2011 (7)], ([RGPV/ June2013 (5)],([RGPV/ June2014 (5)]

• Goal: implement the happens before relation in a distributed system without global clock

• Assume each process has a clock (counter)

• A system of clocks is correct if a → b implies

C(a) < C(b)

• Happens before can be realized if the following clock conditions are met:

[C1] For any two events a and b in a process Pi, if a → b then Ci(a) < Ci(b).

[C2] If a is the event of sending a message in process Pi, a and b is the event of receiving the same message at Pj, then, Ci(a) < Cj(b).

Implement the Logical Clock

[IR1] Clock Ci is incremented between any two successive events in process Pi,

$$Ci := Ci(a) + d \quad (d>0)$$

– It's easy to see if a and b are two successive events in Pi and a → b, then Ci(b) = Ci(a) + d.

[IR2] If a is the event of sending a message m, with timestamp tm := Ci(a), by process Pi. On receiving m by process Pj, Cj is set to a value greater than or equal to its present value and greater than tm.

$$Cj := \max(Cj , tm +d) \quad (d>0)$$

Lamport's Logical Clock

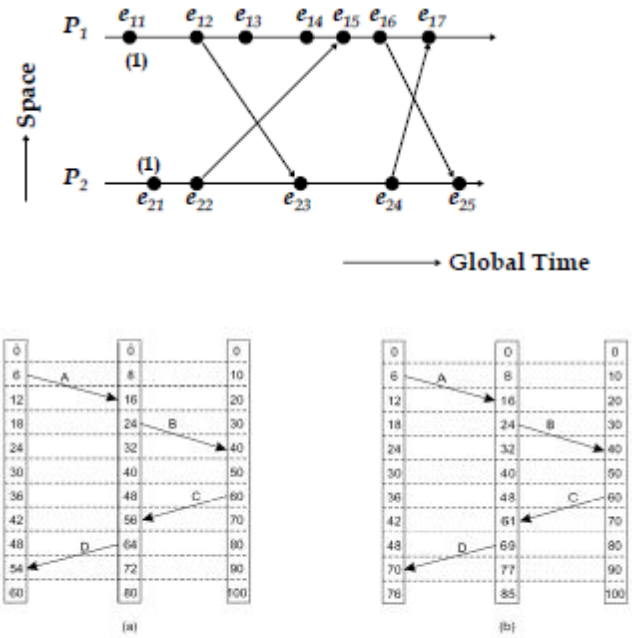


Fig. 5-7. (a) Three processes, each with its own clock. The clocks run at different rates. (b) Lamport's algorithm corrects the clocks.

Total Order

• Order events by their local time, but Lamport's logical clock only defines a partial order.

• Break ties by a total ordering on processes

• Total ordering of events (a → b):

• If a is any event at process Pi and b is any event at process Pj, then a→ b if either

Ci(a) < Cj(b) or

Ci(a) = Cj(b) and Pi << Pj

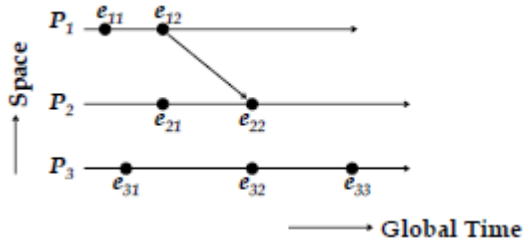
– << is any arbitrary relation that totally orders the processes to break ties. A simple way is to implement

<< by using unique identification numbers.

Limitations of Lamport's Clocks

• If a →b then C(a) < C(b). However the reverse is not necessary true, but we know if C(a) < C(b) then what?

• See figure, C(e11)< C(e22) and C(e11)< C(e32), however, e11 is causally related to event e22,but not to event e32



S.NO	RGPV QUESTIONS	Year	Marks
Q.1	Describe Lamport's Logical Clocks and their limitations.	June 2013 June 2014	5
Q.2	What is logical clock and how do we synchronize logical clock in Lamport's algorithm?	Dec 2011	7

UNIT-01/LECTURE-04

Vector Clock ([RGPV/ June 2013 (5)], ([RGPV/ June2014 (5)]

• Vector clock is proposed to decide whether two events are causally related or not by simply looking at their timestamps

• Vector clock is an array of integers instead of just one integer. In a vector, each entry is used to

represent the knowledge about the clock of other processes

• Ci[j] denotes Pi's knowledge about the logical time at Pj

Implementation Rules

[IR1] Clock Ci is incremented between any two successive events in process Pi,

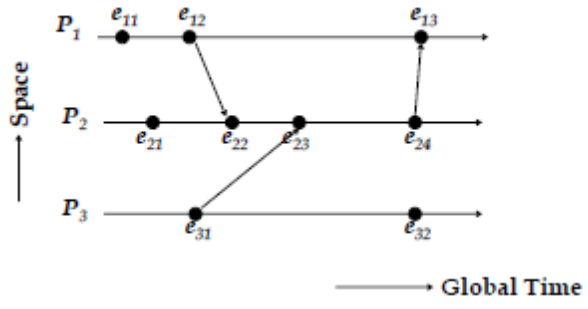
Ci := Ci(a) + d (d>0)

[IR2] If a is the event of sending a message m, with timestamp tm := Ci(a), by process Pi. On receiving m by process Pj, Cj is updated as follows:

k, Cj[k] := max(Cj[k], tm[k])

Assertion: At any time, ∀i,∀j: Ci[i] >= Cj[i]

Vector Clock



Equal: $t^a = t^b$ *iff* $\forall i,\ t^a[i] = t^b[i]$

Not Equal: $t^a \neq t^b$ *iff* $\exists i,\ t^a[i] \neq t^b[i]$

Less Than or Equal: $t^a \leq t^b$ *iff* $\forall i,\ t^a[i] \leq t^b[i]$

Less Than: $t^a < t^b$ *iff* $(t^a \leq t^b \wedge t^a \neq t^b)$

Not Less Than: $t^a \not< t^b$ *iff* $\neg(t^a \leq t^b \wedge t^a \neq t^b)$

Concurrent: $t^a \parallel t^b$ *iff* $(t^a \not< t^b \wedge t^b \not< t^a)$

Assertion: $a \rightarrow b$ *iff* $t^a < t^b$

S.NO	RGPV QUESTIONS	Year	Marks
Q.1	Explain vector clock.	June 2013 June 2014	5

UNIT-01/LECTURE-05

Distributed Mutual Exclusion

Mutual Exclusion

Mutual exclusion refers to the problem of ensuring that no two processes can be in their critical section at the same time. There are several resources in a system that must not be used simultaneously by multiple processes, if program operation to be correct. For example, a file must not be simultaneously updated by multiple processes. Similarly, use of unit record peripherals such as tape drives or printers must be restricted to a single process at a time. Therefore, exclusive access to such a shared resource by a process must be ensured. This exclusiveness of access is called **mutual exclusion** between processes. The section of a program that need exclusive access to shared resources are referred to as critical sections.
For mutual exclusion, means are introduced to prevent processes from executing concurrently within their associated critical sections.

An algorithm for implementing mutual exclusion must satisfy the following requirements –

a) **Mutual Exclusion –** Given a shared resources accessed by multiple concurrent processes, at any time only one process should access the resource. That is, a process that has been granted the resource must release it before it can be granted to another process.

b) **No Starvation –** If every process that is granted the resource eventually releases it, every request must be eventually granted. In single-processor systems, mutual exclusion is implemented using semaphores, monitors and similar constructs.

Requirements of Mutual Exclusion Algorithm

1.Freedom from Deadlocks

2.Freedom from starvation

3.Fairness

Performance of mutual exclusion algorithm is measured by

1.No. of messages

2.Synchronization delay

3.Response time

4.System throughput

There are three algorithms which are used to achieve the mutual exclusion in distributed system-

Mutual Exclusion Algorithm

1.A Centralized Algorithm

2.A Distributed Algorithm

3.A Token Ring Algorithm

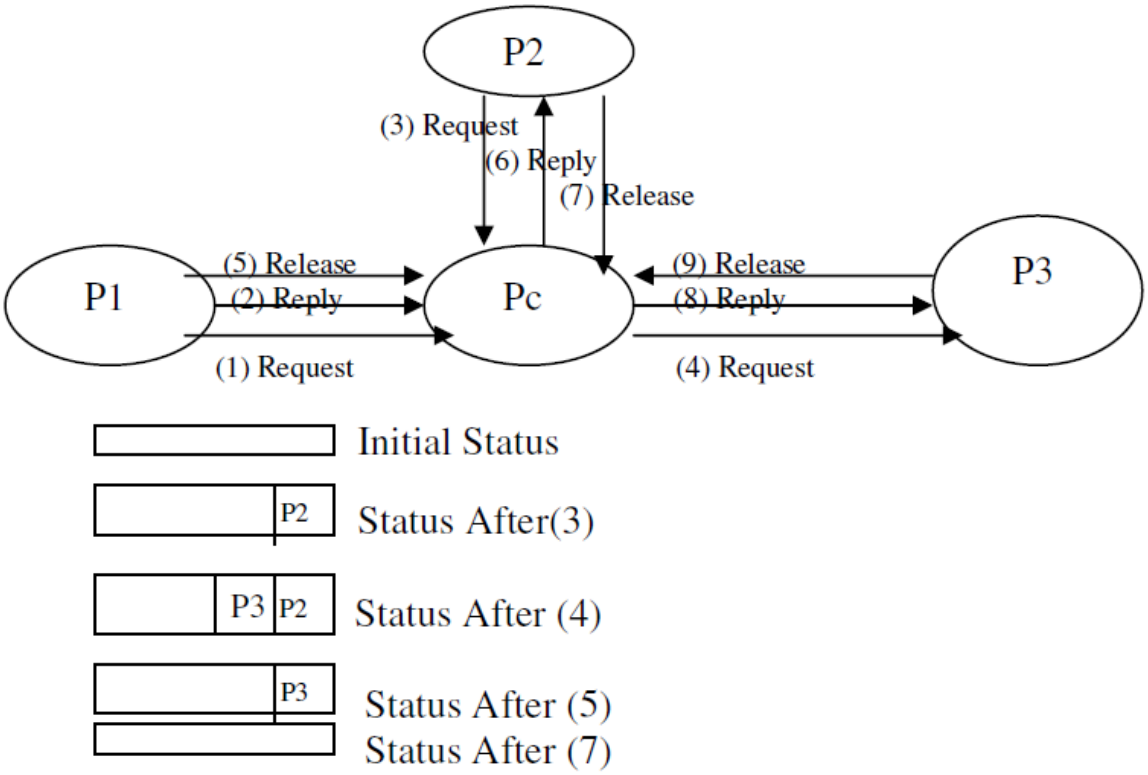
Centralized Algorithm ([RGPV/ June 2014 (7)]

One of the processes in the system is elected as the coordinator to coordinates the entry to the critical sections. Each process that wants to enter a critical section must first seek permission from the coordinator. If no other process is currently in that critical section, the coordinator can immediately grant permission to the requesting process. However if two or more processes concurrently ask for permission
to enter the same critical section. The coordinator knows that a different process is already in the critical section, so it cannot grant permission. After executing a critical section, when a process exits the critical section, it must notify the coordinator so that the coordinator can grant permission to another process that has also asked for permission to enter the same critical section.

Let us suppose that there is a coordinator process (Pc), and three other processes P1, P2, P3

in the system. Also it is assumed that the requests are granted in the first-come, first-served order for which the coordinator maintains a request queue. Suppose P1 wants to enter a critical section for which it sends a request message to

Pc. On receiving the request message, Pc checks to see whether some other process is currently in that critical section. Since no other process is in the critical section Pc immediately sends back a reply message granting permission to P1.when the reply arrives, P1 enters the critical section.



Now suppose if P1 is in critical section P2 asks for permission to enter the same critical section by sending a request message to Pc. Since P1 is already in the critical section, P2 cannot be granted permission. Now let us assume that the coordinator does not return any reply and the process that made the request remains blocked until it receive the reply from the coordinator. Therefore, Pc does not send a reply to P2 immediately and enters its

request in the request queue.

Again, while P1 is still in the critical section P3 sends a request message to Pc asking for permission to enter the same critical section. Obviously, P3 cannot be granted permission, so no reply is sent immediately to P3 by Pc and its request in the request queue.

If P1 exits the critical section and sends a **release** message to Pc releasing its exclusive access to the critical section. On receiving the release message Pc takes the first request from the queue of deferred requests and sends reply message to the corresponding process, granting it permission to enter the critical section. Thus Pc sends a reply message to P2. After receiving the reply message P2 enters the critical section and when it exits the critical section, it sends a release message to Pc. Again Pc takes the first request from the request queue and sends a reply message to the corresponding process(P3). P3 enters the critical section and when it exits the critical section, it sends a release message to Pc. Now since there are no more requests Pc keeps waiting for the next request message.

This algorithm ensures the mutual exclusion because at a time , the coordinator allows only one process to enter a critical section. The algorithm also ensures that no starvation will occur because of the use of first-come, first served scheduling policy. Main advantage of this algorithm is that it is simple to implement and requires only three message per critical section entry – a request, a reply, and a release. The disadvantage of this scheme is that a single coordinator is subject to a single point of failure and can become a performance bottleneck in a large system. Distributed algorithm overcome these drawbacks.

Advantages of Centralized algorithm

1.Guarantees mutual exclusion

2.Easy to implement & requires only three messages per use of a critical section(request, grant, release)

• **Disadvantages of Centralized algorithm**

1.The coordinator is a single point of failure

2.Confusion between no reply & permission denied

3.A single coordinator can become a performance bottleneck

S.NO	RGPV QUESTIONS	Year	Marks
Q.1	Explain the following 1.difference between distributed and centralized system.	June 2014	7

UNIT -01/LECTURE -06

Distributed algorithm([RGPV/ June2014 (7)]

• When a process wants to enter a critical region, it builds a message containing the name of the critical region it wants to enter, its process number, and the current time. It then sends the message to all other processes, conceptually including itself.

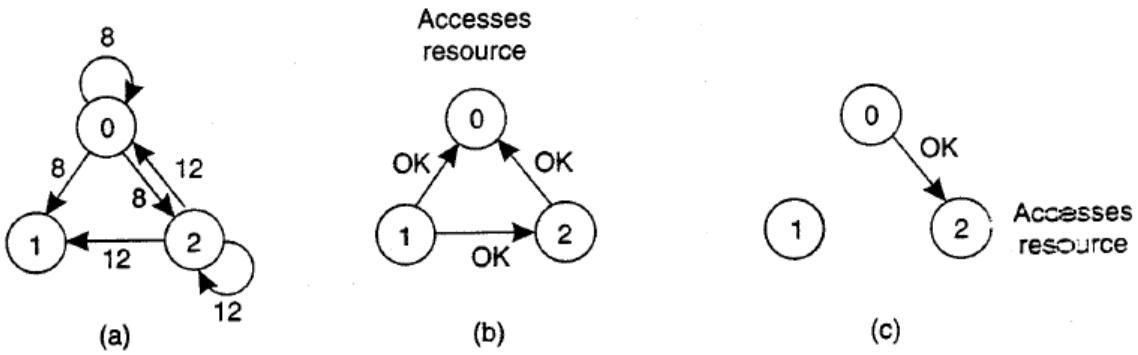
• When a process receives a request message from another process, the action it takes depends on its state with respect to the critical region named in the message. Three cases have to be distinguished:

1.If the receiver is not in the critical region and does not want to enter it, it sends back an *OK* message to the sender.

2.If the receiver is already in the critical region, it does not reply. Instead, it queues the request.

3.If the receiver wants to enter the critical region but has not yet done so, it compares the timestamp in the incoming message with the one contained in the message that it has sent everyone. The lowest one wins. If the incoming message is lower, the receiver sends back an *OK* message. If its own message has a lower timestamp, the receiver queues the incoming request and sends nothing.

4.After sending out requests asking permission to enter a critical region, a process sits back and waits until everyone else has given permission. As soon as all the permissions are in, it may enter the critical region. When it exits the critical region, it sends *OK* messages to all processes on its queue and deletes them all from the queue.



a) Two processes want to enter the same critical region 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 enter the critical region.

• **Advantages of Distributed algorithm**

1. No single point of failure exits

2.Number of messages are 2(n-1)messages (n-1) request & (n-1) reply messages

3.No starvation, total ordering of messages

4.Deadlock free

5. Mutual exclusion guaranteed

• **Disadvantages of Distributed algorithm**

1.Slower

2. More complicated more expensive & less robust than centralized one.

Token Ring Algorithm

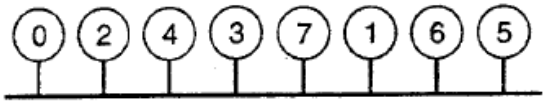
Mutual exclusion is achieved by using a single token that is circulated among the processes in the system. A token is special type of message that entitles its holder to enter a critical section. The processes in the system are logically organized in a ring structure and the token

is circulated from one process to another around the ring always in the same direction (clockwise or anticlockwise).

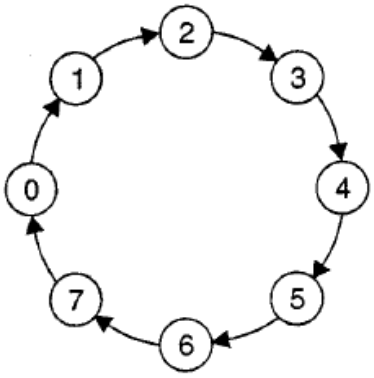
The algorithm works as follows. When a process receive the token, it checks if it wants to enter a critical section and act as follows –

a) If it wants to enter a critical section and exists from the critical section after finishing its work in the critical section. It then passes the token along the ring to its neighbour process. The process can enter only the critical section, it must wait until it gets the token again.

b) If it does not want enter a critical section, it just passes the token along the ring to its neighbour process.



(a)



(b)

Therefore, if none of the processes is interested in entering a critical section, the token simply keeps circulating around the ring. Mutual exclusion is guaranteed by this algorithm because at any instance of time only one process, can be in a critical section, since there is only a single token. Furthermore, since the ring is unidirectional and a process is permitted to enter only one critical section each time it gets the token starvation cannot occur. In this algorithm the number of message per critical section entry may vary from 1 (when every process always want to enter a critical section) to an unbounded value. Moreover, for a

total of n processes in the system, the waiting time form the moment a process wants to enter a critical section until its actual entry may vary from the time needed to exchange 0 to n-1 token passing messages Zero token passing messages are needed when the process receives the token just when it wants to enter the critical section, whereas n-1 messages are needed when the process wants to enter the critical section just after it has passed the token to its neighbour process.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain the following 1.difference between distributed and centralized system.	June 2014	7

UNIT -01/LECTURE -07

Mutual exclusion in single-computer systems vs. distributed systems

In single-computer systems, the status of a shared resource and the status of users is readily available in the shared memory, and solutions to the mutual exclusion problem can be easily implemented using shared variables (e.g., semaphores). However, in distributed systems, both the shared resources and the users may be distributed and shared memory does not exist. Consequently, approaches based on shared variables are not applicable to distributed systems and approaches based on message passing must be used.

The problem of mutual exclusion becomes much more complex in distributed systems (as compared to single-computer systems) because of the lack of both shared memory and a common physical clock and because of unpredictable message delays. Owing to these factors, it is virtually impossible for a site in a distributed system to have current and complete knowledge of the state of the system.

Requirement of mutual exclusion theorem ([RGPV/ June 2013 (7)]

In the problem of mutual exclusion, concurrent access to a shared resource by several uncoordinated user-requests is serialized tosecure the integrity of the shared resource. It requires that the actions performed by a user on a shared resource must be atomic. That is, if several users concurrently access a shared resource then the actions performed by a user, as far as the other users are concerned, must be instantaneous and indivisible such that the net effect on the shared resource is the same as if user actions were executed serially, as opposed to in an interleaved manner. The problem of mutual exclusion frequently arises in distributed systems whenever concurrent access to shared resources by several sites is involved. For correctness, it is necessary that the shared resource be accessed by a single site (or process) at a time. A typical example is directory management, where an update to

a directory must be done atomically because if updates and reads to a directory proceed concurrently, reads may obtain inconsistent information. If an entry contains several fields, a read operation may read some fields before the update and some after the update. Mutual exclusion is a fundamental issue in the design of distributed systems and an efficient and robust technique for mutual exclusion is essential to the viable design of distributed systems.

The primary objective of a mutual exclusion algorithm is to maintain mutual exclusion; that is, to guarantee that only one request accesses the es at a time. **In addition, the following characteristics are considered important in a mutual exclusion algorithm:**

Freedom from Deadlocks. Two or more sites should not endlessly wait for messages that will never arrive.

Freedom from Starvation. A site should not be forced to wait indefinitely to execute es while other sites are repeatedly executing es. That is, every requesting site should get an opportunity to execute es in a finite time.

Fairness. Fairness dictates that requests must be executed in the order they are made (or the order in which they arrive in the system). Since a physical global clock does not exist, time is determined by logical clocks. Note that fairness implies freedom from starvation, but not vice-versa.

Fault Tolerance. A mutual exclusion algorithm is fault-tolerant if in the wake of a failure, it can reorganize itself so that it continues to function without any (prolonged) disruptions.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain Requirement of distributes mutual exclusion. How performance of distributed mutual exclusion are measured.	June 2013	7

UNIT -01/LECTURE -08

Non-Token-Based Algorithms ([RGPV/ June 2013 (7)]

In non-taken-based mutual exclusion algorithms, a site communicates with a set of other sites to arbitrate who should execute the CS next. For a site Si. request set Ri contains ids of all those sites from which site Si must acquire permission before entering the CS.

Next, we discuss some non-token-based mutual exclusion algorithms which are good representatives of this class. Non-token-based mutual exclusion algorithms use timestamps to order requests for the CS and to resolve conflicts between simultaneous requests for the CS. In all these algorithms, logical clocks are maintained and updated according to Lamport's scheme . Each request for the CS gets a timestamp, and smaller timestamp requests have priority over larger timestamp requests.

Lamport's Algorithm

Lamport was the first to give a distributed mutual exclusion algorithm as an illustration of his clock synchronization scheme [9]. In Lamport's algorithm, Vi: 1 .,:; i .,:; N :: Ri = {Sb S2, ..., SN}. Every site Si keeps a queue, requesLqueuei' which contains mutual exclusion requests ordered by their timestamps. This algorithm requires messages to be delivered in the FIFO order between every pair of sites.

The Algorithm

Requesting the critical section:

1. When a site Si wants to enter the CS, it sends a REQUEST(tsi, i) message to all the sites in its request set Ri and places the request on requesLqueuei. «tsi, i) is the timestamp of the request.)

2. When a site Hj receives the REQUEST(t8i, i) message from site Si, it retums a timestamped

REPLY message to and places site Si 's request on requesLqueue}.

Executing the critical section. Site Hi enters the CS when the two following conditions hold:

[L1:] Hi has received a message with timestamp larger than (t8i, i) from all other sites.

[L2:] Hi'S request is at the top of requesLqueuei.

Releasing the critical section.

3. Site Si, upon exiting the CS, removes its request from the top of its request queue and sends a timestamped RELEASE message to all the sites in its request set.

4. When a site *S}* receives a RELEASE message from site *Hi,* it removes *Si'S* request from its request queue.

When a site removes a request from its request queue, its own request may come at the top of the queue, enabling it to enter the CS. The algorithm executes CS requests in the increasing order of timestamps.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain any non-token based distributed algorithm.	June 2013	7

UNIT -01/LECTURE- 09

Token-Based Algorithms ([RGPV/ June 2013 (7)], ([RGPV/ June 2014 (7)]

In token-based algorithms, a unique token is shared among all sites. A site is allowed to enter its CS if it possesses the token. Depending upon the way a site carries out its search for the token, there are numerous token-based algorithms. Next, we discuss some representative token-based mutual exclusion algorithms.

Before we start with the discussion of token-based algorithms, two comments are in order: First, token-based algorithms use sequence numbers instead of timestamps. Every request for the token contains a sequence number and the sequence numbers of sites advance independently. A site increments its sequence number counter every time it makes a request for the token. A primary function of the sequence numbers is to distinguish between old and current requests. Second, a correctness proof of token-based algorithms to ensure that mutual exclusion is enforced is trivial because an algorithm guarantees mutual exclusion so long as a site holds the token during the execution of the CS. Rather, the issues of freedom from starvation and freedom from deadlock are prominent.

Comparison of mutual exclusion Algorithms

Algorithm	Messages Per Entry/Exit	Delay before Entry (in messages times)	Problems
Centralized	3	2	Coordinator Crash
Distributed	2(n-1)	2(n-1)	Crash of any Process
Token Ring	1 to infinite	0 to n-1	Lost Token, Process Crash

Election algorithms

Election algorithms attempt to locate the process with the highest process number and designate it as coordinator.

1. The Bully Algorithm

2. The Ring Algorithm

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain any token based distributed algorithm.	June 2013	10
Q.2	Classify the token based algorithm and signify the importance of token based algorithm in different scenario's.	June 2014	7

UNIT -01/LECTURE -10

Bully Algorithm ([RGPV/ June 2013 (10)], ([RGPV/ June 2014 (10)]

• The group consists of 8 processes. Previously process 7 was the coordinator, but it has just crashed. Process 4 is the first one to notice this, so it sends ELECTION messages to all the processes higher than it, namely 5, 6, and 7

• Processes 5 and 6 both respond with OK, Upon getting the first of these responses, 4 knows that its job is over. It just sits back and waits to see who the winner will be.

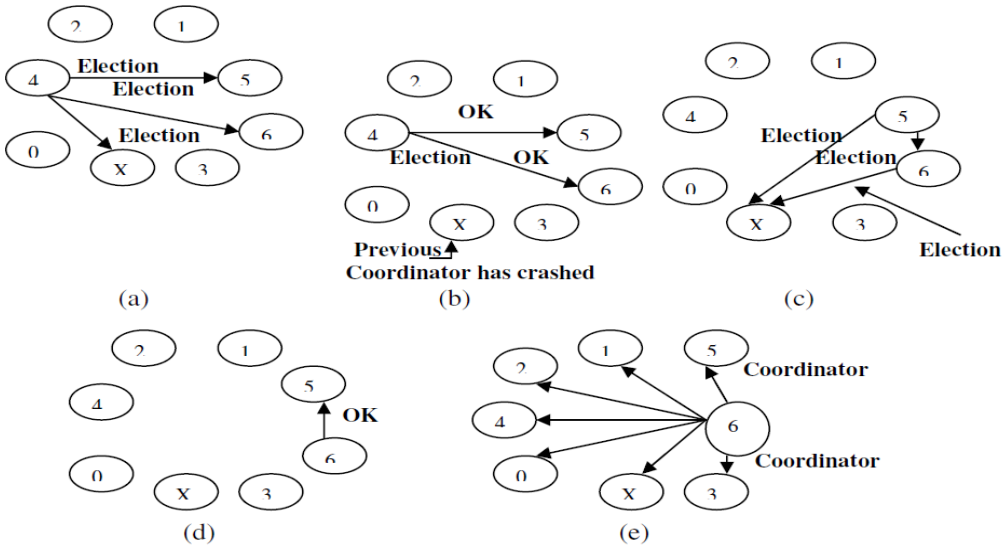
• both 5 and 6 hold elections, each one only sending messages to those processes higher than itself.

• process 6 tells 5 that it will take over. At this point 6 knows that 7 is dead and that it (6) is the winner. When it is ready to take over, 6 announces 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 7 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.

The bully election algorithm. (a) Process 4 holds an election. (b) Processes 5 and 6 respond, telling 4 to stop. (c) Now 5 and 6 each hold an election. (d) Process 6 tells 5 to stop. (e) Process6 wins and tells everyone.

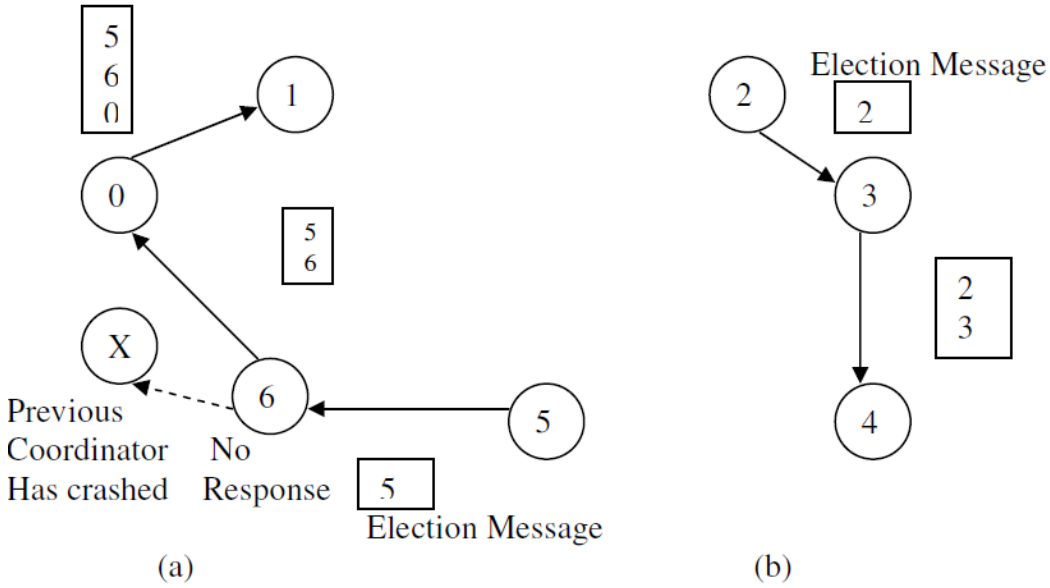


Ring Algorithm

The algorithm is based on the ring-based election algorithm. In this algorithm it is assumed that all the processes in the system are organized in a logical ring. The ring is unidirectional in the sense that all messages related to the election algorithm are always passed only in one direction. Every process in the system knows the structure of the ring, so that each process knows who its successor is. If the successor of the sender process is down, the sender can skip over the successor or the one after that, until an active the algorithm works as follows-

When any process notices that the 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, or the one after that, until a process is located. At each step, the sender adds its own process number to the list in the message. Eventually, the message gets back to the process that started it. That process recognizes this event when it receives an incoming message containing its own process number. At that point the message type is change to COORDINATOR and circulated once again, this time to inform everyone else who the coordinator is and who the members of the new ring are. When this message is circulated once, it is removed and everyone goes back to work. In fig. we see what happen, if two

processes 2 and 5, simultaneously discover that the previous coordinator, process 7 has crashed. Each of these builds an ELECTION message and starts circulating it. Both messages will go all the way around and both 2 and 5 will convert them into COORDINATOR message with exactly the same members and in the order. When both have gone around again, both will be removed. It does no harm extra messages circulating, at most it wastes a little bandwidth.



S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain any token based distributed algorithm.	June 2013	10
Q.2	Classify the token based algorithm and signify the importance of token based algorithm in different scenario's.	June 2014	10

REFERENCES

BOOK	AUTHOR	PRIORITY
Distributed operating systems; Concepts and design	P K Sinha	1
Distributed systems: Principles and paradigms	Tanenbaum and Steen	2