Tulsiramji Gaikwad-Patil College of Engineering & Technology
Department of Master in Computer Application

Subject Notes
Academic Session: 2018 – 2019

Subject:Dos

Semester: I

UNIT 1

1)What is distributed computing environment (DCE) ? How it can be created ? Explain.

ANS=In network computing, DCE (Distributed Computing Environment) is an industry-standard software technology for setting up and managing computing and data exchange in a system of distributed computers. DCE is typically used in a larger network of computing systems that include different size servers scattered geographically. DCE uses the client/server model. Using DCE, application users can use applications and data at remote servers. Application programmers need not be aware of where their programs will run or where the data will be located.**Much of DCE setup requires the preparation of distributed directories so that DCE applications and related data can be located when they are being used.** DCE includes security support and some implementations provide support for access to popular databases such as IBM's CICS, IMS, and DB2 databases.DCE was developed by the Open Software Foundation (OSF) using software technologies contributed by some of its member companies The largest unit of management in DCE is a **cell**. The highest privileges within a cell are assigned to a role called *cell administrator*, normally assigned to the "user" *cell_admin*. Note that this need not be a real OS-level user. The cell_admin has all privileges over all DCE resources within the cell. Privileges can be awarded to or removed from the following categories : user_obj, group_obj, other_obj, any_other for any given DCE resource. The first three correspond to the owner, group member, and any other DCE principal respectively. The last group contains any non-DCE principal. Multiple cells can be configured to communicate and share resources with each other. All principals from external cells are treated as "foreign" users and privileges can be awarded or removed accordingly. In addition to this, specific users or groups can be assigned privileges on any DCE resource, something which is not possible with the traditional UNIX filesystem, which lacks ACL's.

Major components of DCE within every cell are:

1. The **Security Server** that is responsible for authentication
2. The **Cell Directory Server** (CDS) that is the repository of resources and ACLs and
3. The **Distributed Time Server** that provides an accurate clock for proper functioning of the entire cell

Modern DCE implementations such as IBM's are fully capable of interoperating with Kerberos as the security server, LDAP for the CDS and the Network Time Protocolimplementations for the time server.

While it is possible to implement a distributed file system using the DCE underpinnings by adding filenames to the CDS and defining the appropriate ACLs on them, this is not user-friendly. DCE/DFS is a DCE-based application which provides a distributed filesystem on DCE. DCE/DFS can support replicas of a fileset (the DCE/DFS equivalent of a filesystem) on multiple DFS servers - there is one read-write copy and zero or more read only copies. Replication is supported between the read-write and the read-only copies. In addition, DCE/DFS also supports what are called "backup" filesets, which if defined for a fileset are capable of storing a version of the fileset as it was prior to the last replication.

DCE/DFS is believed to be the world's only distributed filesystem that correctly implements the full POSIX filesystem semantics, including byte range locking. DCE/DFS was sufficiently reliable and stable to be utilised by IBM to run the back-end filesystem for the 1996 Olympics web site, seamlessly and automatically distributed and edited worldwide in different timezones

2) Describe following distributed Operating system models :

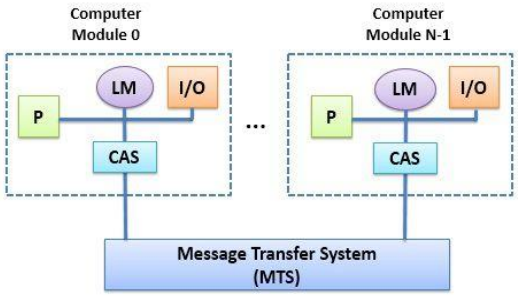
(1) Minicomputer model---The minicomputer model is a simple extension of the centralized time-sharing system. A distributed computing system based on this model consists of a few minicomputers (they may be large supercomputers as well) interconnected by a communication network. Each minicomputer usually has multiple users simultaneously logged on to it. For this, several interactive terminals are connected to each minicomputer. Each user is logged on to one specific minicomputer, with remote access to other minicomputers. The network allows a user to access remote resources that are available on some machine other than the one on to which the user is currently logged. The minicomputer model may be used when resource sharing (such as sharing of information databases of different types, with each type of database located on a different machine) with remote users is desired. The early ARPAnet is an example of a distributed computing system based on the minicomputer model.

(2) Workstation model------A distributed computing system based on the workstation model consists of several workstations interconnected by a communication network. An organization may have several workstations located throughout a building or campus, each workstation equipped with its own disk and serving as a single-user computer. It has been often found that in such an environment, at any one time a significant proportion of the workstations are idle (not being used), resulting in the waste of large amounts of CPU time. Therefore, the idea of the workstation model is to interconnect all these workstations by a high-speed LAN so that idle workstations may be used to process

jobs of users who are logged onto other workstations and do not have sufficient processing power at their own workstations to get their jobs processed efficiently.

3) Explain tightly and loosely coupled system ; also discuss processor pool and hybrid model of DOS

ANS= Multiprocessor is one which has more than two processors in the system. Now when the **degree of coupling** between these processors is very **low**, the system is called **loosely coupled multiprocessor system**. In loosely coupled system each processor has its **own local memory, a set of input-output device**s and a **channel and arbiter switch (CAS)**. We refer to the processor with its local memory and set of input-output devices and CAS as a **computer module**.



Loosely Couple Multiprocessor System

Processes that execute on different computer modules communicate with each other by exchanging the **messages** through a physical segment of **message transfer system (MTS)**. The loosely coupled system is also known as **distributed system**. The loosely coupled system is **efficient** when the processes running on different computer module require **minimal interaction**.If the request fo accessing MTS of two or more computer module collide, the **CAS responsibly** chooses one of the simultaneous requests and delay other requests until selected request is serviced completely. The CAS has a **high-speed communication memory** which can be accessed by all the processors in the system.The communication memory in CAS is used to **buffer the transfers of messages**.

Pool Model

The processor-pool model is based on the observation that most of the time a user does not need any computing power but once in a while the user may need a very large amount of computing power for a short time (e.g., when recompiling a program consisting of a large number of files after changing a basic shared declaration). Therefore, unlike the workstation-server model in which a processor is allocated to each user, in the processor-pool model the processors are pooled together to be shared by the users as needed. The pool of processors consists of a large number of microcomputers and minicomputers attached to the network. Each processor in the pool has its own memory to load and run a system program or an application program of the distributed computing system

Hybrid Model

Out of the four models described above, the workstation-server model, is the most widely used model for building distributed computing systems. This is because a large number of computer users only perform simple interactive tasks such as editing jobs, sending electronic mails, and executing small programs. The workstation-server model is ideal for such simple usage. However, in a

working environment that has groups of users who often perform jobs needing massive computation, the processor-pool model is more attractive and suitable.

4) What are the advantages and disadvantages of distributed operating system ? Explain.

ANS= Operating system is developed to ease people daily life. For user benefits and needs the operating system may be single user or distributed. In distributed systems, many computers connected to each other and share their resources with each other. There are some advantages and disadvantages of distributed operating system that we will discuss.

Distributed operating system

Advantages of distributed operating systems:-

- Give more performance than single system
- If one pc in distributed system malfunction or corrupts then other node or pc will take care of
- More resources can be added easily
- Resources like printers can be shared on multiple pc's

Disadvantages of distributed operating systems:-

- Security problem due to sharing
- Some messages can be lost in the network system
- Bandwidth is another problem if there is large data then all network wires to be replaced which tends to become expensive
- Overloading is another problem in distributed operating systems

Explain workstation—server model and processor pool model used for building distributed computing

- **system**.
- ANS= **Workstation – Server Model** The workstation model is a network of personal workstations, each with its own disk and a local file system. A workstation with its own local disk is usually called a diskful workstation and a workstation without a local disk is called a diskless workstation. With the proliferation of high-speed networks, diskless workstations have become more popular in network environments than diskful workstations, making the workstation-server model more popular than the workstation model for building distributed computing systems.
- A distributed computing system based on the workstation-server model consists of a few minicomputers and several workstations (most of which are diskless, but a few of which may be diskful) interconnected by a communication network. Note that when diskless workstations are used on a network, the file system to be used by these workstations must be implemented either by a diskful workstation or by a minicomputer equipped with a disk for file storage. One or more of the minicomputers are used for implementing the file system. Other minicomputers may be used for providing other types of services, such as database service and print service. Therefore, each minicomputer is used as a server machine to provide one or more types of services. Therefore in the workstation-server model, in addition to the workstations, there are specialized machines (may be specialized workstations) for running server processes (called servers) for managing and providing access to shared resources. For a number of reasons, such as higher reliability and better scalability, multiple servers are often used for managing the resources of a particular type in a distributed computing system. For example, there may be multiple file servers, each running on a separate minicomputer and cooperating via the network, for managing the files of all the users in the system. Due to this reason, a distinction is often made between the services that are provided to clients and the servers that provide them. That is, a service is an abstract entity that is provided by one or more servers. For example, one or more file servers may be used in a distributed computing system to provide file service to the users.
- In this model, a user logs onto a workstation called hi
- s or her home workstation. Normal computation activities required by the user's processes are performed at the user's home workstation, but requests for services provided by special servers (such as a file server or a database server) are sent to a server providing that type of service that performs the user's requested activity and returns the result of request processing to the user's workstation. Therefore, in this model, the user's processes need not migrated to the server machines for getting the work done by those machines.
- **Processor – Pool Model**
- the processor-pool model is based on the observation that most of the time a user does not need any computing power but once in a while the user may need a very large amount of computing power for a short time (e.g., when recompiling a program consisting of a large number of files after changing a basic shared declaration). Therefore, unlike the workstation-server model in which a processor is allocated to each user, in the processor-pool model the processors are pooled together to be shared by the users as needed. The pool of processors consists of a large number of microcomputers and minicomputers attached to the network. Each processor in the pool has its own memory to load and run a system program or an application program of the distributed computing system .

5)Discuss different designing issues of distributed operating system.

ANS= It is not possible to implement centralized control in distributed system due to the problem imposed by the architecture itself of the system, so decentralized control is used that is all control functions are distributed among various sites of the system But in distributed system due to the absence of global shared memory, it is not possible to have upto date knowledge regarding processes and resources and in the absence of updated global state of distributed system, it is also difficult to implement decentralized control.

Another important function of Operating system is scheduling which is mainly done on the basis of arrival time of the processes. But in case of distributed systems due to the absence of global physical clock, scheduling is also difficult to implement.

*** Naming:**

A name refers to an object such as computer, printer, file, a service etc. One of the service is naming service which is implemented using look up procedure. In table look up procedure, tables or directories are used, which contains the logical name. Such directories can be implemented in two ways.

*** Approach1**

A directory is replicated at each site in order to avoid single point of failure & in order to increase the availability of naming service.
Disadvantage: More space is required.
Consistancy problem that is any modification in a directory at one site should also be reflected in same copy at other site.

*** Approach 2**

Directory is partitioned into various blocks and then these blocks are distributed among various sites. The problem is that when a name is referred then how it will be possible to find the location of the corresponding block.
Solution: Another directory is created which contains the information regarding location of these blocks and then this directory is replicated at various sites.

*** Scalability:**

It refers to the concept that when system grows then it should not result in the unavailability of the system or performance degradation. Example: Broadcast based protocols works well for small system but not for large systems. Suppose a distributed system is locating the file by broadcasting the queries then in such system each computer has to handle the message overhead & as the system grows there will be more such queries resulting in more message overhead & degrading the performance of the system.

*** Compatability:**

Compatability refers to inter-operability among resources that is a resource can be used from any computer or it can be used in combination with other resource. There are three levels of compatibility.

*** Process Synchronization:**

Processes running on different machines must access the shared resource in a mutually exclusive manner. In such system, a process can request or release resources at any timeand in any order which may lead to deadlock. Such deadlocks must be detected as early as possible otherwise system performance will degrade.

Resource Management:

In such system, a user must access the remote resources with as much ease as it can access the local resources. Resources can be made available in three ways:
Data Migration: In this approach data is transferred from its source to the location of computation and if any modification is made in the data then it is also reflected at the source side.

6) Define Distributed Computing System. Explain various distributed computing system models in detail

ANS= Workstation – Server Model The workstation model is a network of personal workstations, each with its own disk and a local file system. A workstation with its own local disk is usually called a diskful workstation and a workstation without a local disk is called a diskless workstation. With the proliferation of high-speed networks, diskless workstations have become more popular in network environments than diskful workstations, making the workstation-server model more popular than the workstation model for building distributed computing systems.

A distributed computing system based on the workstation-server model consists of a few minicomputers and several workstations (most of which are diskless, but a few of which may be diskful) interconnected by a communication network. Note that when diskless workstations are used on a network, the file system to be used by these workstations must be implemented either by a diskful workstation or by a minicomputer equipped with a disk for file storage. One or more of the minicomputers are used for implementing the file system. Other minicomputers may be used for providing other types of services, such as database service and print service. Therefore, each minicomputer is used as a server machine to provide one or more types of services. Therefore in the workstation-server model, in addition to the workstations, there are specialized machines (may be specialized workstations) for running server processes (called servers) for managing and providing access to shared resources. For a number of reasons, such as higher reliability and better scalability, multiple servers are often used for managing the resources of a particular type in a distributed computing system. For example, there may be multiple file servers, each running on a separate minicomputer and cooperating via the network, for managing the files of all the users in the system. Due to this reason, a distinction is often made between the services that are provided to clients and the servers that provide them. That is, a service is an abstract entity that is provided by one or more servers. For example, one or more file servers may be used in a distributed computing system to provide file service to the users.

In this model, a user logs onto a workstation called his or her home workstation. Normal computation activities required by the user's processes are performed at the user's home workstation, but requests for services provided by special servers (such as a file server or a database server) are sent to a server providing that type of service that performs the user's requested activity and returns the result of request processing to the user's workstation. Therefore, in this model, the user's processes need not migrated to the server machines for getting the work done by those machines.

7) What are DCE components ? Explain with their interdependencies of DCE components.

ANS= Distributed Computing Environment (DCE) Components
DCE is a blend of various technologies developed independently and nicely integrated by OSF. Each of these technologies forms a component of DCE. The main components of DCE are as follows:

1. Threads package:

It provides a simple programming model for building concurrent applications. It includes operations to create and control multiple threads of execution in a

single process and to synchronize access to global data within an application.
2. Remote Procedure Call facility:
It provides programmers with a number of powerful tools necessary to build client-server applications. The DCE RPC facility is the basis for all communication in DCE because the programming model underlying all of DCE is the client-server model. It is easy to use, is network-independent and protocol-independent, provides secure communication between a client and a server and hides differences in data requirements by automatically converting data to the appropriate forms needed by client and servers.
3. Distributed Time Service:
It closely synchronizer the clocks of all the computers in the system. It also permits the use of time values from external time sources, such as those of the U.S National Institute for Standards and Technology, to synchronize the clocks of the computers in the system with external time.
4. Name services:
The name services of DCE include the Cell Directory Service, the Global Directory Service and the Global Directory Agent. These services allow resources such as servers, file, devices and so on to be uniquely named and accessed in a location-transparent manner.
5. Security Service:
It provides the tools needed for authentication and authorization to protect system resources against illegitimate access.
6. Distributed File Service:
It provides a systemswide file system that has such characteristics as location transparency, high performance and high availability. A unique feature of DCE DFS is that it can also provide file services to clients of other file systems.

DCE Cells n Group Of Users, Machines Or Other Resources Having A Common Purpose & Share Common DCE Services n Helps To Break Down A Large System Into Manageable Units n Minimum Configuration : ÿ Cell Directory Server ÿ Security Server ÿ Distributed Time Server ÿ One Or More Client Machines Factors For Cell Boundaries n Purpose n Administration n Security n OverheaD

UNIT 2

1) What are the characteristics of Good message passing system ?

ANS= Message Passing system is subsystem of Distributed System that provides a set of message based IPC protocol and does so by hiding details of complex network protocols and multiples heterogeneous platform from programmers.

2 Primitives: Send 2 Receive.

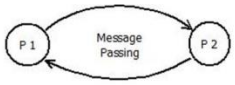


Fig. Message Passing: Basic IPC Mechanism

Features of Message Passing: - **Simplicity**: Simple and Ease to use. Simple and Clear Semantics of IPC Protocols of message passing makes it easier.

- **Uniform Semantics:** There are two types of communication :
    - Local
    - Remote
        1. Semantics of Remote should be as close as possible to local communication for ease of use.
- **Efficiency** :
    - It can be made efficient by reducing number of message exchange.
    - Avoiding cost of setting and terminating connections between the same pair.
    - Minimizing the cost of maintaining connections.
    - Piggybacking.
- **Reliability** :
    - Distributed System is prone to node crashes or communication link failure resulting into loss of data.
    - Handling of lost message
- Duplicate messages, capable of detecting and handling duplicates.
- Generating and assigning appropriate sequences
- **Correctness :**

- Issues
- **Atomicity**: Message to All of None.
- Order Delivery: Message in Order.
- Survivability: Message delivery despite of partial failure.
- **Flexibility** :
  - User may choose types and level of reliability
  - Synchronous/Asynchronous, Send/Receive Choice.
- **Security:**
  - It must provide secure end to end communication.
  - Necessary Steps:
- Authentication of receiver by sender.
- **Authentication of sender by receiver.**
- **Portability:**
  - Two Aspects:
- Message passing system should itself be portable.
- Applications written by using primitives of IPC Protocols of Message Passing System should be Portable.

2) Write short notes on :—

(i) Failure Handling----- en to be a difficult task: in addition to the complications induced by normal distributed execution, the programmer must also deal with component failures that can occur asynchronously during program execution. Although abstractions such as atomic transactions [9] are often used at the application level to handle this extra complexity, such high-level techniques are inappropriate for building system software. In this paper, we describe how flexible mechanisms oriented towards handling processor failures in system software can be integrated into concurrent or distributed programming languages such as Ada [1], CSP [8], Mesa [11], or SR [3]. The design of the mechanisms is based on the observation that the occurrence of failures can be considered as events (Le., state transitions) caused spontaneously by the adverse environment of the system [5]. With this view, the failure of a processor is treated logically within the software as a concurrent event that is generated and signaled in real-time by an underlying processor membership service that detects the failure [7]. The interprocess synchronization and communication mechanism provided by the language (e.g., semaphores, condition variables, messages) is then used to wait for the occurrence of a failure signal and synchronize its activity with normal processing.

(ii) Group Communication--- An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently but in practical, there are many situations when co-operative nature can be utilised for increasing computational speed, convenience and modularity. Inter process communication (IPC) is a mechanism which allows processes to communicate each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other using these two ways:

1. Shared Memory
2. Message passing

The Figure 1 below shows a basic structure of communication between processes via shared memory method and via message passing. Message passing is a technique for invoking behavior (i.e., running a program) on a computer. In contrast to the traditional technique of calling a program by name, message passing uses an object model to distinguish the general function from the specific implementations. The invoking program sends a message and relies on the object to select and execute the appropriate code. The justifications for using an intermediate layer essentially falls into two categories: encapsulation and distribution.

3) Explain process addressing in detail.

ANS= A message passing system generally supports two types of addressing:

- Explicit Addressing: The process with which communication is desired is explicitly specified as a parameter in thE communication primitive. e.g. send (pid, msg), receive (pid, msg).
- Implicit Addressing: A process does not explicitly name a process for communication. For example, a process can specify a service instead of a process. e.g. send any (service id, msg), receive any (pid, msgMethods for process addressing:
- machine id@local id: UNIX uses this form of addressing (IP address, port number).

Advantages: No global coordination needed for process addressing. Disadvantages: Does not allow process migration.machine id1@local id@machine id2: machine id1 identifies the node on which the process is created. local id is generated by the node on which the process is created.machine id2 identifies the last known location of the process. When a process migrates to another

node, the link information (the machine id to which the process migrates) is left with the current machine. This information is used for forwarding messages to migrated processes.

Disadvantages:

– Overhead involved in locating a process may be large.
– If the node on which the process was executing is down, it may not be possible to locate the process.

4) Explain following buffering strategies :—

(1) No buffering--- a buffer is a region of memory used to temporarily hold data while it is being moved from one place to another.

That would be the most simple yet sensible definition for a buffer irrespective of where it may appear. Computers often have different devices in it that work at different speeds. For example the RAM is much faster when compared to the Hard Disk. Further the CPU of a computer is only capable of handling a specific amount of data in a given time.

These and many other reasons make it a need for operating systems to have Buffers or Temporary memory locations it can use. For example imagine that there are two different processes. It can be tricky to transfer data between these processes as the processes may be at two different states at a given time.

Let us say process A : Is sending a bitmap to the printer driver so that it can send it to the printer. Unfortunately the driver is busy printing another page at that time. So until the driver is ready the OS stores the data in a buffer.

The same concept is applied for other things like copying files to a USB drive, playing a video, taking input from a IO device etc.

(2) **Finite-Bound buffer---** The bounded-buffer problems (aka the producer-consumer problem) is a classic example of concurrent access to a shared resource. A bounded buffer lets multiple producers and multiple consumers share a single buffer. Producers write data to the buffer and consumers read data from the buffer.

- Producers must block if the buffer is full.
- Consumers must block if the buffer is empty.

A bounded buffer with capacity N has can store N data items. The places used to store the data items inside the bounded buffer are called slots. Without proper synchronization the following errors may occur.

- The producers doesn't block when the buffer is full.
- A Consumer consumes an empty slot in the buffer.
- A consumer attempts to consume a slot that is only half-filled by a producer.
- Two producers writes into the same slot.
- Two consumers reads the same slot.
- And possibly more

5) Explain buffering in message passing system.

ANS= The transmission of messages from one process to another can be done by copying the body of the message from the sender's address space to the receiver's address space. In some cases, the receiving process may not be ready to receive the message but it wants the operating system to save that message for later reception. In such cases, the operating system would rely on the receiver's buffer space in which the transmitted messages can be stored prior to receiving process executing specific code to receive the message.The synchronous and asynchronous modes of communication correspond to the two extremes of buffering: a null buffer, or no buffering, and a buffer with unbounded capacity. Two other commonly used buffering strategies aresingle-message and finite-bound, or multiple message buffers. These four types of buffering strategies are given below:

- No buffering: In this case, message remains in the sender's address space until the receiver executes the corresponding receive.

- Single message buffer: A buffer to hold a single message at the receiver side is used. It is used for implementing synchronous communication because in this case an application can have only one outstanding message at any given time.

- Unbounded - Capacity buffer: Convenient to support asynchronous communication. However, it is impossible to support unbounded buffer.

- Finite-Bound Buffer: Used for supporting asynchronous communication.

- Buffer overflow can be handled in one of the following ways:

- Unsuccessful communication: send returns an error message to the sending process, indicating that the message could not be delivered to the receiver because the buffer is full.

- Flow-controlled communication: The sender is blocked until the receiver accepts some messages. This violates the semantics of asynchronous send. This will also result in communication deadlock.

1.An absolute pointer value has no meaning (more on this when we talk about RPC). For example, a pointer to a tree or linked list. So, proper encoding mechanisms should be adopted to pass such objects.

2. Different program objects, such as integers, long integers, short integers, and character strings occupy different storage space. So, from the encoding of these objects, the receiver should be able to identify the type and size of the objects.A message data should be meaningful to the receiving process. This implies ideally that the structure of the program should be preserved while they are being transmitted from the address space of the sending process to the address space of the receiving process. It is not possible in heterogeneous systems in which the sending and receiving processes are on computers of different architectures. Even in homogeneous systems, it is very difficult to achieve this goal mainly because of two reasons:One of the following two representations may be used for the encoding and decoding of a message data: 1. Tagged representation: The type of each program object as well as its value is encoded in the message. In this method, it is a simple matter for the receiving process to check the type of each program object in the message because of the self-describing nature of the coded data format. 2. Untagged representation: The message contains only program objects, no information is included in the message about the type of each program object. In this method, the receiving object should have a prior knowledge of how to decode the received data because the coded data format is not self-describing.

6) Discuss synchronization in message passing system.

ANS= Processes in a distributed program may have to communicate and synchronize to accomplish their tasks. The problem of synchronization in different processes, where a region is a block of code whose execution may require synchronization. In general, the region occupancy rules may be complex and ad hot in nature and deriving an algorithm to enforce such rules can be a tedious and an error prone task. In this thesis, we propose a technique to derive algorithms for synchronization in message passing systems. We adopt an aspect oriented technique for systematic development of synchronization code for message passing systems. Our approach is to factor out synchronization as a separate aspect, synthesize synchronization code and then compose it with the functional code. Specifically, we allow the designer to first design the functional code. The designer can then annotate the functional code with regions and specify a high-level "global invariant" specifying the synchronization policy. Given this invariant, a region synchronization protocol is derived in a systematic manner that grants permissions for entering and exiting regions in a sequence that does not invalidate the invariant. Although algorithms for centralized systems have been presented earlier, we show that straightforward translation of these algorithms to distributed systems may result in an incorrect solution. We first give a correctness condition for a region synchronization algorithm in a distributed system and present translations for point-to-point message passing systems and broadcast based message systems such as a controller area network (CAN). We also show how application semantics can be used to improve the performance of our algorithms. With these optimizations, the performance of our algorithms matches that of existing algorithms designed for specific synchronization problems.

The problem of synchronization can be formulated in terms of rules constraining the occupancy of regions in different processes, where a region is a block of code whose execution may require synchronization. In this region synchronization problem, the region occupancy rules (or the synchronization policy) can be specified using a global invariant. The task of a region synchronization algorithm is to constrain the region entry and exit of processes in a manner that satisfies the global invariant. This paper proposes efficient algorithms for region synchronization in message passing systems. In particular, we propose extension of two existing mutual exclusion algorithms to solve the region synchronization problem. We show that our algorithms are message efficient and satisfy the property of absence of unnecessary synchronization. We show that many existing synchronization problems such as group mutual exclusion, readers/writers, committee coordination, and barrier can be specified as instances of the region synchronization problem, and hence our algorithms can be used to solve a large class of problems.

7) Discuss various issues in Inter-Process Communication (IPC) by message passing.

ANS= • How a process can pass information to another • Make sure processes don't get into each others' way • Sequencing and dependencies

Starvation

A starvation condition can occur when multiple processes or threads compete for access to a shared resource. One process may monopolise the resource while others are denied access.

Deadlock

A deadlock condition can occur when two processes need multiple shared resources at the same time in order to continue.

Data Inconsistency

When shared resources are modified at the same time by multiple resources, data errors or inconsistencies may occur. Sections of a program that might cause these problems are called critical sections. Failure to coordinate access to a critical section is called a race condition because success or failure depends on the ability of one process to exit the critical section before another process enters the critical section. It is often the case that two processes are seldom in the critical section at the same time; but when there is overlap in accessing the critical section, the result is a disaster.

Shared Buffer Problem

An example of data inconsistency that can occur because of a race condition is what can happen with a shared bank account. Dear Old Dad adds money to an account and Poor Student withdraws from the account. When either accesses the account, they execute a critical section consisting of three steps.

1. Read account balance
2. Update the balance
3. Write the new balance to the account

One day, Dear Old Dad checks the balance and seeing that it is $100 decides to add $50 to the account. Unfortunately, access to the account is not locked, so just then Poor Students withdraws $20 from the account and the new balance is recorded as $80. After adding the $50, Dear Old Dad records the balance as $150, rather than $130, as it should be.

The nature of the problem is more clear when we examine the assembly language code for such an operation:

UNIT 3

1) What is RPC ? Explain principle of RPC between a client and server program.

ANS= 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.

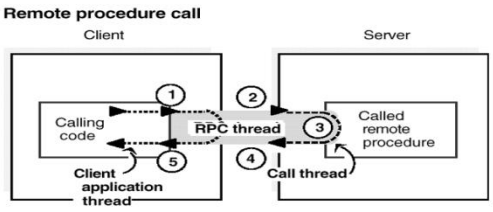


Figure 6-1 Execution Phases of an RPC Thread

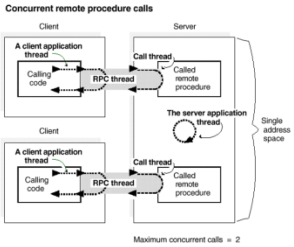


Figure 6-2 Concurrent Call Threads Executing in Shared Execution Context

- The execution phases of an RPC thread, as shown in Execution Phases of an RPC Thread include the following:
    1. 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.
    2. The RPC thread extends across the network to the server.
    3. 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.
    4. The RPC thread then retracts across the network to the client.
    5. 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.

2) What is stateful server ? Explain with example.

ANS= A stateful server keeps state between connections. A stateless server does not.So, when you send a request to a stateful server, it may create some kind of connection object that tracks what information you request. When you send another request, that request operates on the state from the previous request. So you can send a request to "open" something. And then you can send a request to "close" it later. In-between the two requests, that thing is "open" on the server.

When you send a request to a stateless server, it does not create any objects that track information regarding your requests. If you "open" something on the server, the server retains no information at all that you have something open. A "close" operation would make no sense, since there would be nothing to close.

HTTP and NFS are stateless protocols. Each request stands on its own.Sometimes cookies are used to add some state to a stateless protocol. In HTTP (web pages), the server sends you a cookie and then the browser holds the state, only to send it back to the server on a subsequent request.SMB is a stateful protocol. A client can open a file on the server, and the server may deny other clients access to that file until client closes it.

A **stateful** system instead can be seen as a box where at any point in time the value of the output(s) depends on the value of the input(s) and of an internal state, so basicaly a stateful system is like a state machine with "*memory*" as the same set of input(s) value can generate different output(s) depending on the previous input(s) received by the system.

From the **parallel programming** point of view, a **stateless** system, if properly implemented, can be executed by multiple threads/tasks at the same time without any concurrency issue [as an example think of a reentrant function] A **stateful** system will requires that multiple threads of execution access and update the internal state of the system in an exclusive way, hence there will be a need for a serialization [synchronization] point.

3) Explain in detail RPC messages and Marshaling Arguments.

ANS= When program statements that use RPC framework are compiled into an executable program, a stub is included in the compiled codethat acts as

the representative of the remote procedure code. When the program is run and the procedure call is issued, the stub receives the request and forwards it

to a client runtime program in the local computer.

The client runtime program has the knowledge of how to address the remote computer and server application and sends the message across the network that requests the remote procedure. Similarly, the server includes a runtime program and stub that interface with the remote procedure itself. Response-request protocols are returned the same way. **marshalling** or **marshaling** is the process of transforming the memory representation of an object to a data format suitable for storage or transmission, and it is typically used when data must be moved between different parts of a computer program or from one program to another. Marshalling is similar to serialization and is used to communicate to remote objects with an object, in this case a serialized object. It simplifies complex communication, using composite objects in order to communicate instead of *primitives*. The inverse of marshalling is called *unmarshalling* (or *demarshalling*, similar to *deserialization*). To "serialize" an object means to convert its state into a byte stream in such a way that the byte stream can be converted back into a copy of the object.

The term "marshal" is considered to be synonymous with "serialize" in the Python standard library,[5] but the terms are not synonymous in the Java-related RFC 2713:

To "marshal" an object means to record its state and codebase(s)[note 1] in such a way that when the marshalled object is "unmarshalled," a copy of the original object is obtained, possibly by automatically loading the class definitions of the object. You can marshal any object that is serializable or remote (that is, implements the java.rmi.Remote interface). Marshalling is like serialization, except marshalling also records codebases. Marshalling is different from serialization in that marshalling treats remote objects specially.

4) Write note on :—

(i) Types of RPC's----- Synchronous

This is the normal method of operation. The client makes a call and does not continue until the server returns the reply.

Nonblocking

The client makes a call and continues with its own processing. The server does not reply.

Batching

This is a facility for sending several client nonblocking calls in one batch.

Broadcast RPC

RPC clients have a broadcast facility, that is, they can send messages to many servers and then receive all the consequent replies.

Callback RPC

The client makes a nonblocking client/server call, and the server signals completion by calling a procedure associated with the client.

(ii) Exception Handling----- Actors send error messages to others by returning an `error` (see § *:ref:`error <#error>`__*) from a message handler. Similar to exit messages, error messages usually cause the receiving actor to terminate, unless a custom handler was installed via `set_error_handler(f)`, where `f` is a function object with signature `void (error&)` or `void (scheduled_actor*, error&)`. Additionally, `request` accepts an error handler as second argument to handle errors for a particular request (see § 1.5.2). The default handler is used as fallback if `request` is used without error handler. By default, all messages have the default priority, i.e., `message_priority::normal`. Actors can send urgent messages by setting the priority explicitly: `send<message_priority::high>(dst, ...)`. Urgent messages are put into a different queue of the receiver's mailbox. Hence, long wait delays can be avoided for urgent communication.+

(A) Explain RPC mechanism with the help of RPC model using suitable diagram

- ANS= 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.

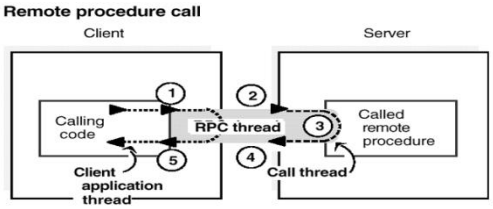


Figure 6-1 Execution Phases of an RPC Thread

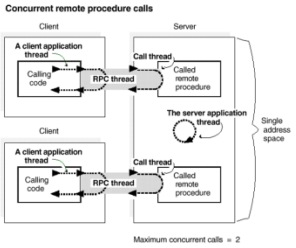


Figure 6-2 Concurrent Call Threads Executing in Shared Execution Context

- The execution phases of an RPC thread, as shown in Execution Phases of an RPC Thread include the following:
    1. 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.
    2. The RPC thread extends across the network to the server.
    3. 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.
    4. The RPC thread then retracts across the network to the client.
    5. 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.

5) Discuss the implementation of RPC Mechanism in detail.

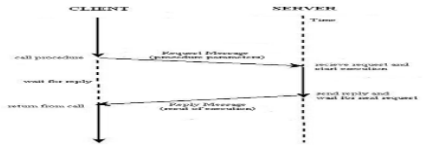
ANS= RPC Implementation Mechanism

- RPC mechanism uses the concepts of stubs to achieve the goal of semantic transparency.
- Stubs provide a local procedure call abstraction by concealing the underlying RPC mechanism.
- A separate stub procedure is associated with both the client and server processes.
- RPC communication package known as RPC Runtime is used on both the sides to hide existence and functionalities of a network.
- Thus implementation of RPC involves the five elements of program:
    1. Client
    2. Client Stub
    3. RPC Runtime
    4. Server stub
    5. Server

The client, the client stub, and one instance of RPC Run time execute on the client machine.

The server, the server stub, and one instance of RPC Run time execute on the server machine.

Remote services are accessed by the user by making ordinary LPC.



1. Client

- A Client is a user process which initiates a RPC
- The client makes a normal call that will invoke a corresponding procedure in the client stub.

2. Client Stub

Client stub is responsible for the following two tasks:

i. On receipt of a call request from the client, it packs specifications of the target procedure and arguments into a message and asks the local RPCRuntime to send it to the server stub.

ii. On receipt of the result of procedure execution, it unpacks the result and passes it to the client.

3. RPCRuntime

- Transmission of messages between Client and the server machine across the network is handled by RPCRuntime.
- It performs Retransmission, Acknowledgement, Routing and Encryption.
- RPCRuntime on Client machine receives messages containing result of procedure execution from server and sends it client stub as well as the RPCRuntime on server machine receives the same message from server stub and passes it to client machine.
- It also receives call request messages from client machine and sends it to server stub.

4. Server Stub

Server stub is similar to client stub and is responsible for the following two tasks:

i. On receipt of a call request message from the local RPCRuntime, it unpacks and makes a normal call to invoke the required procedure in the server.

ii. On receipt of the result of procedure execution from the server, it unpacks the result into a message and then asks the local RPCRuntime to send it to the client stub.

5. Server

When a call request is received from the server stub, the server executes the required procedure and returns the result to the server stub.

6) Discuss server implementation in detail.

ANS= Our goal in constructing server operating systems is to enable modular construction of server applications that deliver full hardware performance, without artificial limitations caused by unnecessary software overheads. This section describes our ongoing implementation of a prototype server OS, which includes efficient and parameterizable implementations of TCP/IP [21] and a disk-based file system, in terms of the server OS support outlined in Section 2. We are building server OSs as libraries on top of an exokernel [8]. The exokernel OS architecture is designed to provide application-level software with direct, protected access to hardware resources by limiting kernel functionality to multiplexing hardware resources among applications. This kernel support is sufficient to allow us to construct server OSs such as the one described in Section 2. In addition, because the exokernel OS architecture allows multiple differently-specialized applications to safely co-exist and timeshare hardware resources, we can focus our server OS implementation on providing useful abstractions and reducing software engineering complexity. As a beneficial sideeffect of the exokernel architecture, the fact that the server OS is in application space generally makes it easier to build, test and debug. Specialization To support modular specialization, we have made our TCP/IP and file system libraries highly parameterizable and easy to integrate with other components of a server application. For example, the TCP/IP library does not manage protocol control block (PCB) allocation, allowing server applications to incorporate PCBs into their own central data structures. It also allows applications to specify that a particular transfer is the last for the connection (so that the FIN flag can be set in the last data packet, instead of sending a separate packet) and to provide precomputed checksums for the data being transmitted. The file system library implements a file system similar to the Fast File System [14]. However, in addition to support for non-blocking operations, as described above, this library is highly configurable. In particular, the cache replacement, miss handling, write-back and flush routines are all specified by the application during the initialization phase. Although default implementations exist, it is trivial to replace them. Also, the disk allocation code can be replaced easily, allowing for application-specific data layouts. Finally, extra inode fields are provided to allow server applications to add their own information (e.g., prefetching hints and extra type information). Direct device-to-device access The TCP/IP library and the file system library both support direct device-to-device data movement by allowing applications to use scatter/gather I/O and specify source (destination)locations for outgoing (incoming) data. In addition, the TCP/IP library does not keep retransmission buffers, but instead invokes a call-back if the data must be retransmitted. Together with file system support for pinning and write-locking disk cache blocks, this allows applications to construct a combined, copy-free disk cache/retransmission pool.

7) What are the main differences between the RPC model and ordinary procedure ?

ANS= RPC: Remote procedure call (RPC) is an Inter-process communication technology that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction.

Web Service: Web services are typically application programming interfaces (API) or web APIs that are accessed via Hypertext Transfer Protocol and executed on a remote system hosting the requested services. Web services tend to fall into one of two camps: Big Web Services[1] and RESTful Web Services.Remote Procedure Call (RPC) and WebService, in the sake of Functionality both go parallely. But there is a subtle difference in their way of invoking. An Web Service can be invoked by any application, using XML format over HTTP protocol for proceedings and its interoperable in nature, whereas in case of RPC the function can be Invoked by multi applications so it follow the path of Serialization to store the object data. It supports Binary Format over TCP protocol. In a better approach we can brief RPC workflow, like we are executing a function through proper Socket and proper format of message, but don't know the actual existence of the particular function in client server.Even the provided socket might not be in the same server in which the function resides. But every time it give a feel like the function is located in the local. In Remote Service, the Function resides in remote machine and it can be invoked by proper format and Protocol and it allows Scalability. A definition of the format of the messages that are passed between two endpoints using its and elements and appropriate schema definitions. • The semantics of the service: how it might be called to make a synchronous request/reply, synchronous reply-only or asynchronously communicate. • The end point and transport of the service via the element: that is, who provides the service. • An encoding via the element, that is how the service is accessed.

8) What is the role of stubs in RPC mechanism ? Explain with its method of generation.

- ANS= 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.

UNIT 4

1) Why is distributed shared memory (DSM) used ? Explain the general architecture of DSM.

ANS= distributed shared memory (**DSM**) is a form of memory architecture where physically separated memories can be addressed as one logically shared address space. Here, the term "shared" does not mean that there is a single centralized memory, but that the address space is "shared" (same physical address on two processors refers to the same location in memory).[1]:201 **Distributed global address space** (**DGAS**), is a similar term for a wide class of software and hardware implementations, in which each node of a cluster has access to shared memory in addition to each node's non-shared private memory.

A distributed-memory system, often called a multicomputer, consists of multiple independent processing nodes with local memory modules which is connected by a general interconnection network. Software DSM systems can be implemented in an operating system, or as a programming library and can be thought of as extensions of the underlying virtual memory architecture. When implemented in the operating system, such systems are transparent to the developer; which means that the underlying distributed memory is completely hidden from the users. In contrast, software DSM systems implemented at the library or language level are not transparent and developers usually have to program them differently. However, these systems offer a more portable approach to DSM system implementations. A distributed shared memory system implements the shared-memory model on a physically distributed memory system. A distributed-memory system, often called a multicomputer, consists of multiple independent processing nodes with local memory modules which is connected by a general interconnection network. Software DSM systems can be implemented in an operating system, or as a programming library and can be thought of as extensions of the underlying virtual memory architecture. When implemented in the operating system, such systems are transparent to the developer; which means that the underlying distributed memory is completely hidden from the users. In contrast, software DSM systems implemented at the library or language level are not transparent and developers usually have to program them differently. However, these systems offer a more portable approach to DSM system implementations. A distributed shared memory system implements the shared-memory model on a physically distributd memory system

The trade-off between false sharing elimination and aggregation in distributed shared memory (DSM) systems has a major effect on their performance. Some studies in this area show that fine grain access is advantageous, while others advocate the use of large coherency units. One way to resolve the trade-off is to dynamically adapt the granularity to the application memory access pattern. In this paper, we propose a novel technique for implementing

multiple sharing granularities over page-based DSMs. We present protocols for efficient switching between small and large sharing units during runtime. We show that applications may benefit from adapting the memory sharing to the memory access pattern, using both coarse grain sharing and fine grain sharing interchangeably in different stages of the computation. Our experiments show a substantial improvement in the performance using adapted granularity levelover using a fixed granularity level.

2) Explain following consistency models :

(1) Sequential consistency model---- Sequential consistency is one of the consistency models used in the domain of concurrent computing (e.g. in distributed shared memory, distributed transactions, etc.).

It was first defined as the property that requires that

> "... the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program."[1]

To understand this statement, it is essential to understand one key property of sequential consistency: execution order of program in the same processor (or thread) is the same as the program order, while execution order of program between processors (or threads) is undefined. The sequential consistency is weaker than strict consistency, which requires a read from a location to return the value of the last write to that location; strict consistency demands that operations be seen in the order in which they were actually issued.

(2) Causal consistency model---Causal consistency is a weakening model of sequential consistency by categorizing events into those causally related and those that are not. It defines that only write operations that are causally related need to be seen in the same order by all processes.

This model relaxes Sequential consistency on concurrent writes by a processor and on writes that are not causally related. Two writes can become causally related if one write to a variable is dependent on a previous write to any variable if the processor doing the second write has just read the first write. The two writes could have been done by the same processor or by different processors.

As in sequential consistency, reads do not need to reflect changes instantaneously, however, they need to reflect all changes to a variable sequentially.

*P*1: $W_1(x)3$
*P*2: $W_2(x)5$ $R_1(x)3$

W_1 is not causally related to W_2. R1 would be Sequentially Inconsistent but is Causally consistent.(Possibly wrong citation and wrong answer for sequentially inconsistent, someone check)[5]

*P*1:W(x)1		W(x)3	
*P*2:R(x)1	W(x)2		
*P*3:R(x)1		R(x)3	R(x)2
*P*4:R(x)1		R(x)2	R(x)3

W(x)1 and W(x) 2 are causally related due to the read made by P2 to x before W(x)2

3) Explain the terms with respect to DOS :—

(i) Granularity---- Granularity Granularity refers to the size of a data unit in which it exists in the distributed shared memory. This is an important decision which essentially governs the design of a DSM. The immediate successor of shared memory from multiprocessor world would have a page as the unit for data transfer. But it has its own disadvantages

Naming scheme When a process wants to access remote data it has to know on which machine does the data reside and fetch it from there. Since all data (or at least the shared one) is visible to all the machines, there has to be a unique naming mechanism to avoid any conflicts. One possible solution is to have a logical global address space. The VM manager at each node performs the translation of the logical address to get the location of the data segment on a remote machine. But such an approach would not be useful if the granularity of shared data is less than a page. In such a case the calling process will have to possess explicit knowledge of the remote location of the data which it wants to access

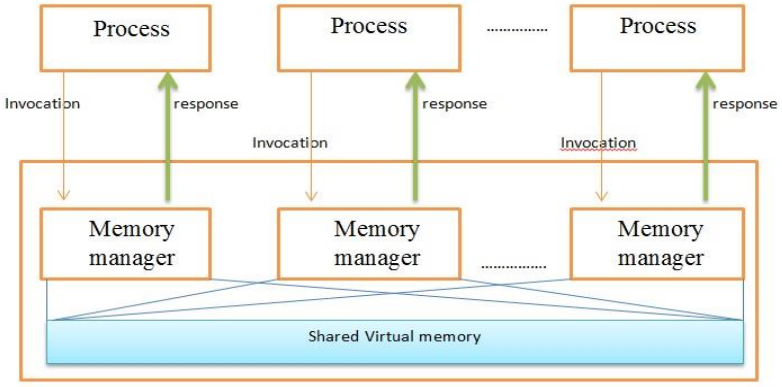
(ii) Thrashing----- Thrashing is computer activity that makes little or no progress, usually because memory or other resources have become exhausted or too limited to perform needed operations. When this happens, a pattern typically develops in which a request is made of the operating system by a process or program, the operating system tries to find resources by taking them from some other process, which in turn makes new requests that can't be satisfied. In a virtual storage system (an operating system that manages its logical storage or memory in units called pages), thrashing is a condition in which excessive paging operations are taking place.A system that is thrashing can be perceived as either a very slow system or one that has come to a halt. Thrashing happens when too many computer processes compete for inadequate memory resources. Thrashing can occur due to several factors, with the most prominent reason being insufficient RAM or memory leakage. In a computer, some applications have higher priorities than others and this can also attribute to thrashing when there is a lack of memory resources. Thrashing can cause slowdown of the system performance since data transfer has to be between the hard drive and physical memory. One of the early signs of thrashing is when an application stops responding while the disk drive light blinks on and off. The operating system often warns users of low virtual memory when thrashing is occurring.

A temporary solution for thrashing is to eliminate one or more running applications. One of the recommended ways to eliminate thrashing is to add more memory to main memory. Another way of resolving the issue of thrashing is by adjusting the size of the swap file.

4) How is concurrency control achieved using DSM ? Explain the structure of shared memory space in distributed shared memory.

ANS= distributed shared memory (**DSM**) is a form of memory architecture where physically separated memories can be addressed as one logically shared address space. Here, the term "shared" does not mean that there is a single centralized memory, but that the address space is "shared" (same physical address on two processors refers to the same location in memory).[1]:201 **Distributed global address space** (**DGAS**), is a similar term for a wide class of software and hardware implementations, in which each node of a cluster has access to shared memory in addition to each node's non-shared private memory.

A distributed-memory system, often called a multicomputer, consists of multiple independent processing nodes with local memory modules which is connected by a general interconnection network. Software DSM systems can be implemented in an operating system, or as a programming library and can be thought of as extensions of the underlying virtual memory architecture. When implemented in the operating system, such systems are transparent to the developer; which means that the underlying distributed memory is completely hidden from the users. In contrast, software DSM systems implemented at the library or language level are not transparent and developers usually have to program them differently. However, these systems offer a more portable approach to DSM system implementations. A distributed shared memory system implements the shared-memory model on a physically distributed memory system



Distributed shared memory

The trade-off between false sharing elimination and aggregation in distributed shared memory (DSM) systems has a major effect on their performance. Some studies in this area show that fine grain access is advantageous, while others advocate the use of large coherency units. One way to resolve the trade-off is to dynamically adapt the granularity to the application memory access pattern. In this paper, we propose a novel technique for implementing multiple sharing granularities over page-based DSMs.

Distributed concurrency control is the concurrency control of a system distributed over a computer network (Bernstein et al. 1987, Weikum and Vossen 2001).

In *database systems* and *transaction processing* (*transaction management*) distributed concurrency control refers primarily to the concurrency control of a distributed database. It also refers to the concurrency control in a multidatabase (and other multi-transactional object) environment (e.g., federated database, grid computing, and cloud computing environments. A major goal for distributed concurrency control is distributed serializability (or global serializability for multidatabase systems). Distributed concurrency control poses special challenges beyond centralized one, primarily due to communication and computer latency. It often requires special techniques, like distributed lock manager over fast computer networks with low latency, like switched fabric (e.g., InfiniBand). Commitment ordering (or commit ordering) is a general serializability technique that achieves distributed serializability (and global serializability in particular) effectively on a large scale, without concurrency control information distribution (e.g., local precedence relations, locks, timestamps, or tickets), and thus without performance penalties that are typical to other serializability techniques (Raz 1992).

5) Discuss design and implementation issues in DSM system.

Ans= Any DSM system has to rely on the underlying message passing technique across the network for data exchange between two computers. The DSM has to present a uniform global view of the entire address space (consisting of physical memories of all the machines in the network) to a program executing on any machine. A DSM manager on a particular machine would capture all the remote data accesses made by any process running on that machine. A design of a DSM would involve making following choices · Where, with respect to the Virtual Memory Manager does the DSM operate? · What kind of consistency model should the system provide? · What should be the granularity of the shared data? · What kind of naming scheme has to be used to access remote data?

Relation with Virtual Memory Manager The issue that arises here is from the fact that the virtual memory on a particular machine and the distributed shared memory (which is nothing but an abstraction) are inherently different. But the aim is to provide a uniform view of both to the user applications. A DSM can be implemented parallel to the VM so that in case of a page fault the VM can decide if the missing page is available locally or requires a remote access.

Choice of consistency model Consistency is a very important parameter in a design of DSM. As discussed earlier, the possible replication of shared data across multiple nodes mean that different machines have different copies of the data. Thus the DSM has to maintain consistency in the state of these copies of the same data. This

problem is similar to cache coherency in multi processors. Several solutions which are also prevalent in other scenarios are proposed for this which can be broadly classified as · Write-all scheme: In this scheme, if a data is modified at some node, the updated copy of the data is shipped to all the nodes which are currently holding a copy of that data. This scheme is simple to implement and most reliable, but it is expensive in terms of network traffic because the whole data has to be shipped across multiple clients. · Write-invalidate scheme: In this scheme, if a data is modified at some node, instead of sending the entire data across, the host sends an invalidate message to all other machine holding a copy of that data. Thus this scheme puts lesser load on the network and hence performs better. Whenever a machine tries to access an invalid data, the DSM manager fetches the valid data from the host.

Granularity Granularity refers to the size of a data unit in which it exists in the distributed shared memory. This is an important decision which essentially governs the design of a DSM. The immediate successor of shared memory from multiprocessor world would have a page as the unit for data transfer. But it has its own disadvantages

Naming scheme When a process wants to access remote data it has to know on which machine does the data reside and fetch it from there. Since all data (or at least the shared one) is visible to all the machines, there has to be a unique naming mechanism to avoid any conflicts. One possible solution is to have a logical global address space. The VM manager at each node performs the translation of the logical address to get the location of the data segment on a remote machine. But such an approach would not be useful if the granularity of shared data is less than a page. In such a case the calling process will have to possess explicit knowledge of the remote location of the data which it wants to access

6) Write notes on :

(i) Causal Consistency Model------- Causal consistency is a weakening model of sequential consistency by categorizing events into those causally related and those that are not. It defines that only write operations that are causally related need to be seen in the same order by all processes.

This model relaxes Sequential consistency on concurrent writes by a processor and on writes that are not causally related. Two writes can become causally related if one write to a variable is dependent on a previous write to any variable if the processor doing the second write has just read the first write. The two writes could have been done by the same processor or by different processors.

As in sequential consistency, reads do not need to reflect changes instantaneously, however, they need to reflect all changes to a variable sequentially.

*P*1: $W_1(x)3$

*P*2: $W_2(x)5$ $R_1(x)3$

W_1 is not causally related to W_2. R1 would be Sequentially Inconsistent but is Causally consistent.(Possibly wrong citation and wrong answer for sequentially inconsistent, someone check)[5]

*P*1:W(x)1		W(x)3	
*P*2:R(x)1	W(x)2		
*P*3:R(x)1		R(x)3	R(x)2
*P*4:R(x)1		R(x)2	R(x)3

W(x)1 and W(x) 2 are causally related due to the read made by P2 to x before W(x)2

(ii) Processor Consistency Model----- In order for consistency in data to be maintained and to attain scalable processor systems where every processor has its own memory, the Processor consistencymodel was derived.[5] All processors need to be consistent in the order in which they see writes done by one processor and in the way they see writes by different processors to the same location (coherence is maintained). However, they do not need to be consistent when the writes are by different processors to different locations.

Every write operation can be divided into several sub-writes to all memories. A read from one such memory can happen before the write to this memory completes. Therefore, the data read can be stale. Thus, a processor under PC can execute a younger load when an older store needs to be stalled. Read before Write, Read after Read and Write before Write ordering is still preserved in this model.

The processor consistency model[6] is similar to PRAM consistency model with a stronger condition that defines all writes to the same memory location must be seen in the same sequential order by all other processes. Processor consistency is weaker than sequential consistency but stronger than PRAM consistency model.

The Stanford DASH multiprocessor system implements a variation of processor consistency which is incomparable (neither weaker nor stronger) to Goodman's definitions.[7] All processors need to be consistent in the order in which they see writes by one processor and in the way they see writes by different processors to the same location. However, they do not need to be consistent when the writes are by different processors to different locations.

UN IT 5

1)How do clock synchronization issues differ in centralized and distributed computing systems ?

ANS= As a result of the difficulties managing time at smaller scale, there are problems associated with clock skew that take on more complexity in distributed computing in which several computers will need to realize the same global time.

For instance, in Unix systems the *make* command is used to compile new or modified code and seeks to avoid recompiling unchanged code.

The *make* command uses the clock of the machine it runs on to determine which source files need to be recompiled. If the sources reside on a separate file server and the two machines have unsynchronized clocks, the *make* program might not produce the correct results.

In a system with central server, the synchronization solution is trivial; the server will dictate the system time. Cristian's algorithm and the Berkeley algorithm are potential solutions to the clock synchronization problem in this environment.

In a distributed system the problem takes on more complexity because a global time is not easily known. The most used clock synchronization solution on the Internet is the Network Time Protocol (NTP) which is a layered client-server architecture based on UDP message passing. Lamport timestamps and vector clocks are concepts of the logical clock in distributed systems.

In a wireless network, the problem becomes even more challenging due to the possibility of collision of the synchronization packets on the wireless medium and the higher drift rate of clocks on the low-cost wireless devices.

Synchronization is achieved in ad-hoc wireless networks through sending synchronization messages in a multi-hop manner and each node progressively synchronizing with the node that is the immediate sender of a synchronization message. A prominent example is the Flooding Time Synchronization Protocol (FTSP),[4] which achieves highly accurate synchronization in the order of a few microseconds. Another protocol, Harmonia,[5] is able to achieve synchronization even when the device firmware cannot be modified and is the fastest known synchronization protocol in ad-hoc wireless networks.

2) Discuss advantages of process migration.

ANS-- Load balancing (load sharing) policy determines: u if the process needs to be moved (migrated) from one node of the distributed system to another. u which process needs to be migrated u what is the node to which the process it to be moved n process migration mechanism deals with the actual transfer of the process source node destination node execution resumed process in execution time freezing time execution suspended

Advantages of process migration n balancing the load: u reduces average response time of processes u speeds up individual jobs u gains higher throughput n moving the process closer to the resources it is using: u utilizes resources effectively u reduces network traffic n being able to move a copy of a process (replicate) on another node improves system reliability n a process dealing with sensitive data may be moved to a secure machine (or just to a machine holding the data) to improve security.Desirable features of good process migration mechanism n Transparency u object access level - access to objects (such as files and devices) by process can be done in location -independent manner. u system call and interprocess communication level - the communicating processes should not notice if one of the parties is moved to another node, system calls should be equivalent n Minimal interference (with process execution) - minimize freezing time n Minimal residual dependencies - the migrated process should not depend on the node it migrated from or: u previous node is still loaded u what if the previous node fails? n Efficiency: u minimize time required to migrate a process u minimize cost relocating the process u minimize cost of supporting the migrated process after migration

3) Write characteristics of good global scheduling algorithm.

ANS= **1) No apriority knowledge about the processes:**

The working of scheduling algorithm is based on the information about the characteristics and resource requirements of the processes. These pose extra burden on the users who must provide this information while submitting their processes for execution. No such information is required for global scheduling algorithm.

2) Dynamic in Nature:

The decision regarding the assignment of process should be dynamic, which means that it should based on the current load of the system and not on the some static policy. The flexibility to migrate the process more than once should be the property of algorithm. The process should be placed on a particular node which can be changed afterwords, based on the initial decision in order to adapt to the change in system load.

3) Decision - making capabilities:

If we think about the heuristic methods, they require less computational efforts that result in less time requirement for the output. This will provide a near optimal result and has decision - making capability.

4) Balancing System performance and Scheduling overhead:

Here we require algorithm that will provide us near optimal system performance. It is desirable to collect minimum of global state information, such as CPU load. Such information is crucial because as the amount of global state information collected increases, the overhead also increases. This will have an impact on the result of the cost of gathering and processing the extra information so there is a need to improve the system performance by minimizing scheduling overhead.

5) Stability:

Processors thrashing (due to the fruitless migration of processes) must be prevented. For example, if nodes n_1 and n_2 are loaded with processes and it is observed that node n_3 is idle, then we can offload a portion of their work to n_3 without being aware of the offloading decision made by some of the other nodes. In case $n_3$become overloaded due to this, it may again start transferring its processes to other nodes. The main reason for this is that scheduling decisions are being made at each node independently of decisions made by other nodes.

6) Scalability:

The scheduling algorithm should be able to scale as the number of nodes increases. An algorithm can be termed as having poor scalability if it makes a scheduling decisions by first inquiring the workload from all the nodes and then selecting the most lightly loaded node. This concept will work fine only when we have few nodes in the system. This will happen because the inquirer receives a flood of replies simultaneously, and the time required

to process the reply messages for making a node selection is too long. As the number of nodes (N) increases, the network traffic consumes network bandwidth quickly.

7) Fault Tolerance:

In case one or more nodes of the system crash, good scheduling algorithm should not be disabled by this and mechanism to handle this should be available. The algorithm should be capable of functioning properly within the nodes if it is observed that the nodes are partitioned into two or more groups due to link failures. In order to have better fault tolerance capability, algorithms should decentralize the decision - making capability and must consider only available nodes in their decision - making and have better fault tolerance capability.

8) Fairness of Service:

If the global scheduling policy blindly attempts to balance the load on all the nodes of the system, then they are not good if viewed in terms of fairness of service. This is because in any load - balancing scheme, heavily nodes will obtain all benefits while lightly loaded nodes will suffer poor response time than in a stand alone configuration. Thus, we say that load balancing needs should be replaced by load sharing. That is, a node will share some of its resources as long as it users are not significantly affected.

4) What is deadlock ? Discuss in detail deadlock prevention.

- **ANS=** Ans= For the purposes of deadlock discussion, a system can be modeled as a collection of limited resources, which can be partitioned into different categories, to be allocated to a number of processes, each having different needs.
- Resource categories may include memory, printers, CPUs, open files, tape drives, CD-ROMS, etc.
- By definition, all the resources within a category are equivalent, and a request of this category can be equally satisfied by any one of the resources in that category. If this is not the case (i.e. if there is some difference between the resources within a category), then that category needs to be further divided into separate categories. For example, "printers" may need to be separated into "laser printers" and "color inkjet printers".
- Some categories may have a single resource.
- In normal operation a process must request a resource before using it, and release it when it is done, in the following sequence:
    1. Request - If the request cannot be immediately granted, then the process must wait until the resource(s) it needs become available. For example the system calls open( ), malloc( ), new( ), and request( ).
    2. Use - The process uses the resource, e.g. prints to the printer or reads from the file.
    3. Release - The process relinquishes the resource. so that it becomes available for other processes. For example, close( ), free( ), delete( ), and release( ).
- For all kernel-managed resources, the kernel keeps track of what resources are free and which are allocated, to which process they are allocated, and a queue of processes waiting for this resource to become available. Application-managed resources can be controlled using mutexes or wait( ) and signal( ) calls, (i.e. binary or counting semaphores.)
- A set of processes is deadlocked when every process in the set is waiting for a resource that is currently allocated to another process in the set (and which can only be released when that other waiting process makes progress.)

Eliminate Mutual Exclusion
It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tap drive and printer, are inherently non-shareable.

Eliminate Hold and wait
1. Allocate all required resources to the process before start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization. for example, if a process requires printer at a later time and we have allocated printer before the start of its execution printer will remained blocked till it has completed its execution.
2. Process will make new request for resources after releasing the current set of resources. This solution may lead to starvation.

Eliminate No Preemption
Preempt resources from process when resources required by other high priority process.

Eliminate Circular Wait
Each resource will be assigned with a numerical number. A process can request for the resources only in increasing order of numbering.
For Example, if P1 process is allocated R5 resources, now next time if P1 ask for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.

. 5) What is Event Ordering ? Explain Happened-Before Relation.

Ans= A distributed system consists of a collection of distinct processes which are spatially separated, and which communicate with one another by exchanging messages. A network of interconnected computers, such as the ARPA net, is a distributed system. A single computer can also be viewed as a distributed system in which the central control unit, the memory units, and the input-output channels are separate processes. A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process. We will concern ourselves primarily with systems of spatially separated computers. However, many of our remarks will apply more generally. In particular, a multiprocessing system on a single computer involves problems similar to those of a distributed system because of the unpredictable order in which certain events can occur. In a distributed system, it is sometimes impossible to say that one of two events occurred first. The relation "happened before" is therefore only a partial ordering of the events in the system. We have found that problems often arise because people are not fully aware of this fact and its implications.

Ordering the Events Totally In computer science, the **happened-before** relation (denoted:) is a relation between the result of two events, such that if one event should happen before another event, the result must reflect that, even if those events are in reality executed out of order (usually to optimize program flow). This involves ordering events based on the potential causal relationship of pairs of events in a concurrent system, especially asynchronous distributed systems. It was formulated by Leslie Lamport.[1] In Java specifically, a **happens-before** relationship is a guarantee that memory written to by statement A is visible to statement B, that is, that statement A completes its write before statement B starts its read.[1]

The happened-before relation is formally defined as the least strict partial order on events such that:

he happened-before relation is formally defined as the least strict partial order on events such that:

6) What is Load-Balancing Approach ? Explain Centralized Versus Distributed.

ANS=

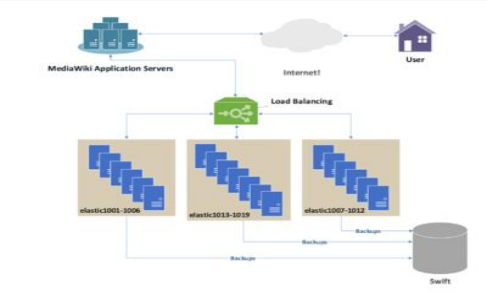


Diagram illustrating user requests to an Elasticsearch cluster being distributed by a load balancer. (Example for Wikipedia.)

In computing, **load balancing**[1] improves the distribution of workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units, or disk drives.[1] Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource. Using multiple components with load balancing instead of a single component may increase reliability and availability through redundancy. Load balancing usually involves dedicated software or hardware, such as a multilayer switch or a Domain Name System server process.

Hardware and software load balancers may have a variety of special features. The fundamental feature of a load balancer is to be able to distribute incoming requests over a number of backend servers in the cluster according to a scheduling algorithm. Most of the following features are vendor specific:

Asymmetric load

A ratio can be manually assigned to cause some backend servers to get a greater share of the workload than others. This is sometimes used as a crude way to account for some servers having more capacity than others and may not always work as desired.

Priority activation

When the number of available servers drops below a certain number, or load gets too high, standby servers can be brought online

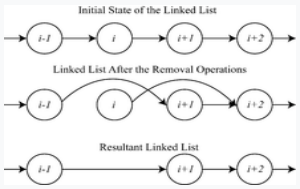
centralized versus Distributed

Centralized approach collects information to server node and makes assignment decision Distributed approach contains entities to make decisions on a predefined set of nodes Centralized algorithms can make efficient decisions, have lower fault-tolerance Distributed algorithms avoid the bottleneck of collecting state information and react faster

Load-balancing approach Type of distributed load-balancing algorithms

7) Explain Mutual Exclusion in detail. Explain distributed approach with respect to Mutual Exclusion.

ANS=



Two nodes, *i* and *i* + 1, being removed simultaneously results in node *i* + 1 not being removed.

In computer science, **mutual exclusion** is a property of concurrency control, which is instituted for the purpose of preventing race conditions; it is the requirement that one thread of execution never enters its critical section at the same time that another concurrent thread of execution enters its own critical section.[a]

The requirement of mutual exclusion was first identified and solved by Edsger W. Dijkstra in his seminal 1965 paper titled *Solution of a problem in concurrent programming control*,[1][2] which is credited as the first topic in the study of concurrent algorithms.[3]

A simple example of why mutual exclusion is important in practice can be visualized using a singly linked list of four items, where the second and third are to be removed. The removal of a node that sits between 2 other nodes is performed by changing the *next* pointer of the previous node to point to the next node (in other words, if node *i* is being removed, then the *next* pointer of node *i* − 1 is changed to point to node *i* + 1, thereby removing from the linked list any reference to node *i*). When such a linked list is being shared between multiple threads of execution, two threads of execution may attempt to remove two different nodes simultaneously, one thread of execution changing the *next* pointer of node *i* − 1 to point to node *i* + 1, while another thread of execution changes the *next* pointer of node *i* to point to node *i* + 2. Although both removal operations complete successfully, the desired state of the linked list is not achieved: node *i* + 1 remains in the list, because the *next* pointer of node *i* − 1 points to node *i* + 1.

This problem (called a *race condition*) can be avoided by using the requirement of mutual exclusion to ensure that simultaneous updates to the same part of the list cannot occur.

- Mutual Exclusion ensures that no other process will use shared resources at same time.

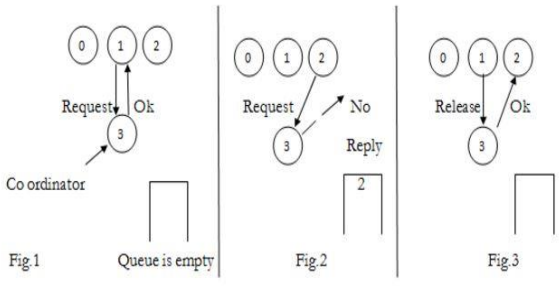
  1) Centralized Algorithm

  2) Distributed Algorithm

  3) Token Ring Algorithm.

- One process is elected as coordinator.
- Whenever process wants to enter a critical region , it sends request msg to coordinator asking for permission.
- If no other process is currently in that critical region, the coordinator sends back a reply granting permission.
- When reply arrives, the requesting process enters the critical region.
- If the coordinator knows that a different process is already in critical regions, so it cannot be granted permission.

Centralized Algorithm:



Advantages:

- Guarantees mutual exclusion.
- Fair Approach (Request Granted In FCFS).

- No Starvation.
- Easy to Implement.
- Only 3 Msgs per use of Critical Section (request, grant, release).

Drawbacks:

- Single point of failure.
- Dead co-ordinate & permission denied cannot distinguish.
- In large systems, single coordinators can create performance bottleneck.

Distributed Algorithm:

- Timestamps are used for distributed mutual exclusion.
- Kieart & Agarwala's Algorithm:
- When process wants to enter critical region, it builds message containing name of critical

region its process number and current time

- It sends msg to all including itself.
- If receiver if not in critical region and doesn't want to enter it sends back Ok msg to sender.
- If the receiver is already in critical region, it doesn't reply, instead it queues request.
- If the receiver wants to enter critical region but has not yet done, so it compares the timestamp in the
- incoming msg the lowest one wins.
-

8) Explain Centralized versus Distributed system with suitable example.

ANS.= Centralized

A system with centralized multiprocessor parallel architecture.In the late 1980 s Centralized systems have been progressively replaced by distributed systems.

characteristics of centralized system

- Non autonomous components
- usually homogeneous technology
- Multiple users share the same resources at all time
- single point of control
- single point of failure

Distributed
set of tightly coupled programs executing on one or more computers which are interconnected through a network and coordinating their actions. These programs know about one another and carry out tasks that none could carry out in isolation

characteristics of distributed system

- autonomous components
- Mostly build using heterogeneous technology
- System components may be used exclusively
- Concurrent processes can execute
- Multiple point of failure

Requirement of distributed system

1. Scalability- possibility of adding new hosts
2. openness- easily extended and modified
3. Heterogeneity-supports various H/W S/w platforms
4. Resource sharing- H/w, S/W and data
5. fault tolerance- ability to function correctly even if faults occur