Define Distributed System and describe its goals?

Distributed System

- A distributed system is a collection of independent computers that appear to the users of the system as a single computer.
- In such a system, multiple resources work together to deliver the required processing speed and the operating system takes care of maintaining the system and overall maintenance.
- In distributed system computers are not independent but interconnected by a high speed network.
- It means that many computers, be they workstation or desktop systems linked together, can do work of a high performance supercomputer.

Goals (Advantages)

Economics

- With microprocessor technology, Grosch's law no longer holds.
- For a few hundred dollars we can get the CPU chip that can execute more instructions per second than one of the largest 1980s mainframe.

Speed

- A collection of microprocessors cannot only give a better price/performance ratio than a single mainframe, but also give an absolute performance that no mainframe can achieve at any price.

Inherent Distribution

- Many institutions have applications which are more suitable for distributed computation.
- Consider the global employee database of a multinational company. Branch offices would create local database and then link them for global viewing.

Reliability

- By distributing the workload over many machines, a single chip failure will bring down at most one machine, leaving the rest intact.
- For critical applications, such as control of nuclear reactors or aircraft, using a distributed system to achieve high reliability may be the dominant consideration.

Incremental Growth:

- With the distributed system, it may be possible simply to add more processors to the system, thus allowing it to expand gradually as the need arises.

Challenges for the distributed system include:

- Security is a big challenge in a distributed environment, especially when using public networks.
- Fault tolerance could be tough when the distributed model is built based on unreliable components.
- Coordination and resource sharing can be difficult if proper protocols or policies are not in place.
- Process knowledge should be put in place for the administrators and users of the distributed model.

Explain the Advantages of Distributed System over Independent PCs

Data sharing

- Many users need to share data.
- Shared data are absolutely essential and many other applications, so the machines must be interconnected.

Resource sharing

- Expensive peripherals, such as color laser printers, phototypesetters, and massive archival storage devices (e.g., optical jukeboxes), can also be shared.

Communication

- For many people, electronic mail has numerous attractions over paper mail, telephone, and FAX.
- It is much faster than paper mail, does not require both parties to be available at the same time as does the telephone, and unlike FAX, produces documents that can be edited, rearranged, stored in the computer, and manipulated with text processing programs.

Flexibility

- A distributed system is more flexible than giving each user an isolated personal computer.

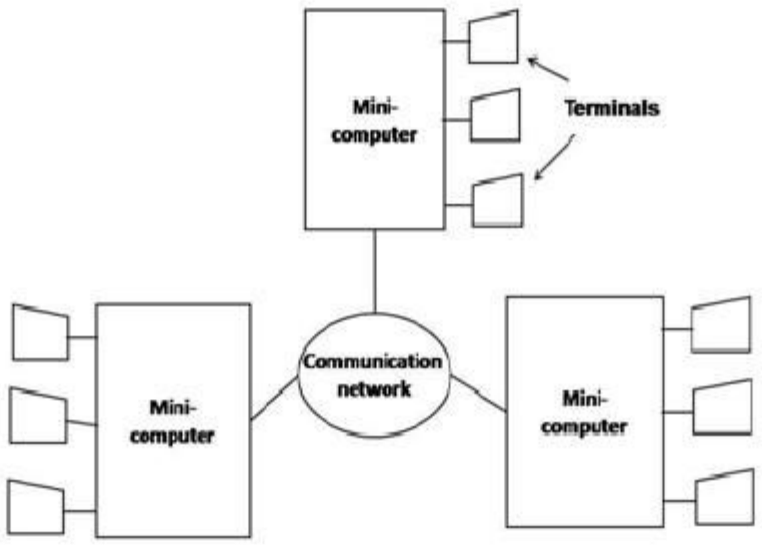
- One way is to give each person a personal computer and connect them all with a LAN, this is not the only possibility.
- Another one is to have a mixture of personal and shared computers, perhaps of different sizes, and let jobs run on the most appropriate one, rather than always on the owner's machine.
- In this way, the workload can be spread over the computers more effectively, and the loss of a few machines may be compensated for by letting people run their jobs elsewhere.

Describe the different distributed system models:

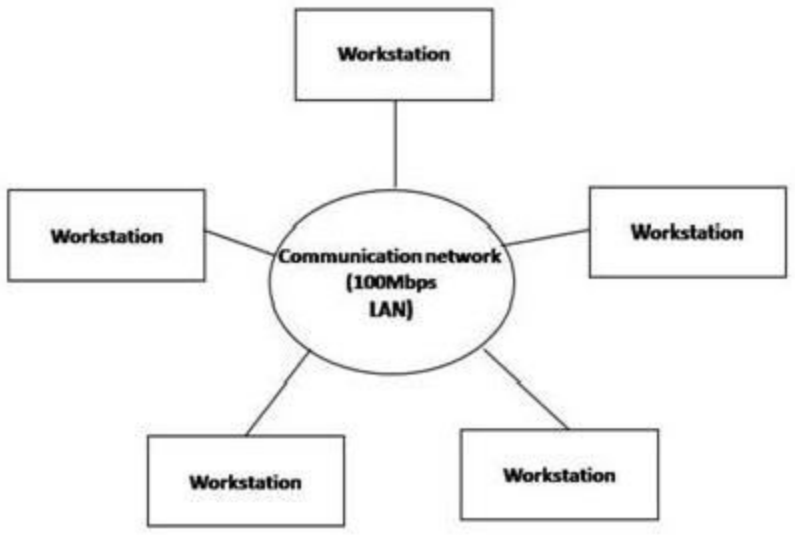
The various models that are used for building distributed computing systems can be classified into 5 categories:

1. Minicomputer Model

- The minicomputer model is a simple extension of the centralized time-sharing system. This model consists of a few minicomputers interconnected by a communication network were each minicomputer usually has multiple users simultaneously logged on to it.
- Several interactive terminals are connected to each minicomputer. The early ARPA net is an example of a distributed computing system based on the minicomputer model.

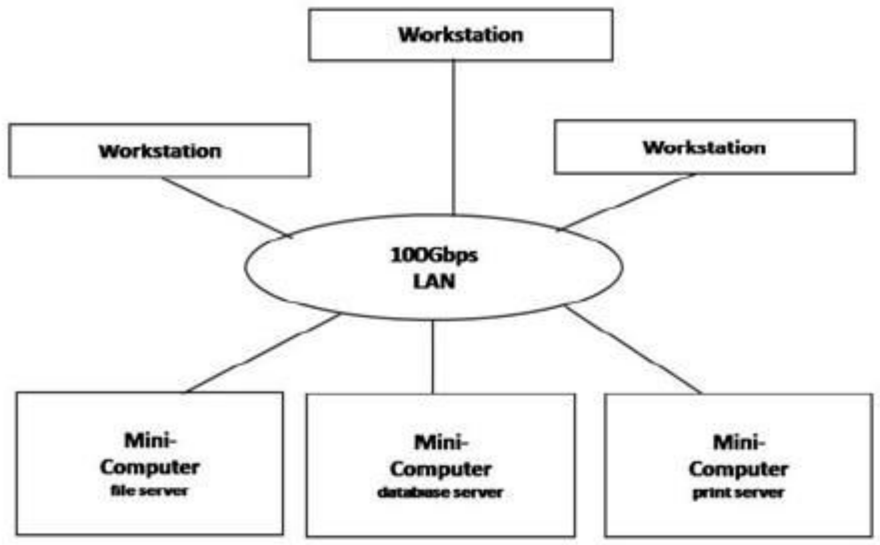


2. **Workstation Model**
    - A distributed computing system based on the workstation model consists of several workstations interconnected by a communication network.



    - 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 & do not have sufficient processing power at their own workstations to get their jobs processed efficiently.
        - Example: Xerox PARC.

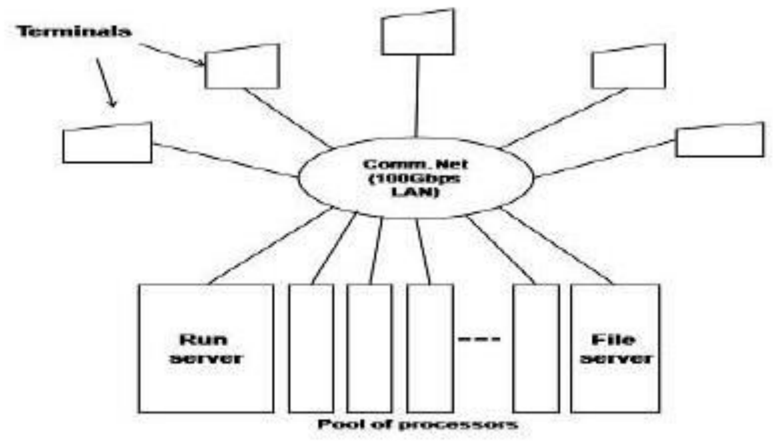
3. **Workstation–Server Model**


Workstation
Workstation
Workstation
100Gbps LAN
Mini-Computer file server
Mini-Computer database server
Mini-Computer print server

- A distributed computing system based on the workstation-server model consists of a few minicomputers & several workstations interconnected by a communication network.
- 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 are sent to a server providing that type of service that performs the user's requested activity & returns the result of request processing to the user's workstation.

Example: The V-System.

4. **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.
- The pool of processors consists of a large number of microcomputers & minicomputers attached to the network.
- Each processor in the pool has its own memory to load & run a system program or an application program of the distributed computing system.

Example: Amoeba & the Cambridge Distributed Computing System.

5. **Hybrid Model:**

- To combine Advantages of workstation-server & processor-pool models, a hybrid model can be used to build a distributed system.

- The hybrid model gives guaranteed response to interactive jobs allowing them to be more processed in local workstations of the users

Explain the issues in design a distributed system

1. **Heterogeneity**

The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks. For eg., a computer attached to an Ethernet has an implementation of the Internet protocols over the Ethernet, whereas a computer on a different sort of network will need an implementation of the Internet protocols for that network.

2. **Openness**

The openness of a computer system is the characteristic that determines whether the system can be extended and re-implemented in various ways. The openness of distributed systems is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs.

3. **Security**

Many of the information resources that are made available and maintained in distributed systems have a high essential value to their users. Their security is therefore of considerable importance. Security for information resources has three components: confidentiality, integrity, and availability.

4. **Scalability**

Distributed systems operate effectively and efficiently at many different scales, ranging from a small intranet to the Internet. A system is described as scalable if it will remain effective when there is a significant increase in the number of resources and the number of users.

5. **Failure handling**

Failures in a distributed system are partial – that is, some components fail while others continue to function. Therefore the handling of failures is particularly difficult.

6. **Concurrency**

Both services and applications provide resources that can be shared by clients in a distributed system. Object that represents a shared resource in a distributed system must be responsible for ensuring that it operates correctly in a concurrent environment. This applies not only to servers but also to objects in applications. Therefore any programmer who takes an implementation of an object that was not intended for use in a distributed system must do whatever is necessary to make it safe in a concurrent environment.

7. **Transparency**

Transparency can be achieved at two different levels. Easiest to do is to hide the distribution from the users. The concept of transparency can be applied to several aspects of a distributed system.

a) Location transparency: The users cannot tell where resources are located

b) Migration transparency: Resources can move at will without changing their names

c) Replication transparency: The users cannot tell how many copies exist.

d) Concurrency transparency: Multiple users can share resources automatically.

e) Parallelism transparency: Activities can happen in parallel without users knowing.

8. **Quality of service**

Once users are provided with the functionality that they require of a service, such as the file service in a distributed system, we can go on to ask about the quality of the service provided. Adaptability to meet changing system configurations and resource availability has been recognized as a further important aspect of service quality.

9. **Reliability**

One of the original goals of building distributed systems was to make them more reliable than single-processor systems. Data entrusted to the system must not be lost or garbled in any way, and if files are stored redundantly on multiple servers, all the copies must be kept consistent. In general, the more copies that are kept, the better the availability, but the greater the chance that they will be inconsistent, especially if updates are frequent.

10. **Performance**

Always the hidden data in the background is the issue of performance. In particular, when running a particular application on a distributed system, it should not be worse than running the same application on a single processor. Unfortunately, achieving this is easier said than done.

Examples of distributed systems

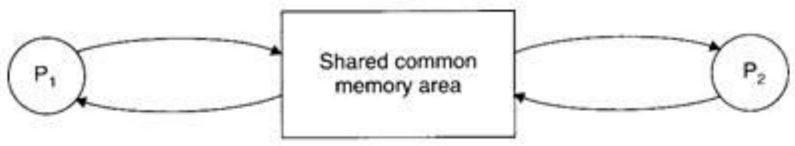
- Intranets, Internet, WWW, email, ...
- DNS (Domain Name Server) - Hierarchical distributed database
- Distributed supercomputers, Grid/Cloud computing
- Electronic banking
- Airline reservation systems
- Peer-to-peer networks
- Sensor networks
- Mobile and Pervasive Computing

UNIT II

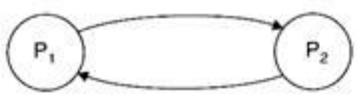
Features of Message Passing System, Synchronization and Buffering, Introduction to RPC and its models, Transparency of RPC, Implementation Mechanism, Stub Generation and RPC Messages, Server Management, Call Semantics, Communication Protocols and Client Server Binding.

Explain Interprocess Communication (IPC)

- Interprocess communication (IPC) basically requires information sharing among two or more processes.
- Two basic methods for information sharing are as follows:
    - Original sharing, or shared-data approach.
    - Copy sharing , or message-passing approach.
- In the **shared-data approach**, the information to be shared is placed in a common memory area that is accessible to all processes involved in an IPC.



- In the **message-passing** approach, the information to be shared is physically copied from the sender process's space to the address space of all the receiver processes.



- A message-passing system is a subsystem of distributed operating system that provides a set of message-based IPC protocols, and does so by shielding the details of complex network protocols and multiple heterogeneous platforms from programmers.
- It enables processes to communicate by exchanging messages and allows programs to be written by using simple communication primitives, such as send and receive.

Explain the Desirable Features of a Good Message Passing system

- **Simplicity**

  A message passing system should be simple and easy to use. It must be straight forward to construct new applications and to communicate with existing one by using the primitives provided by message passing system.

- **Uniform Semantics**

  In a distributed system, a message-passing system may be used for the following two types of interprocess communication:

  Local communication, in which the communicating processes are on the same node.

  Remote communication, in which the communicating processes are on different nodes.

  Semantics of remote communication should be as close as possible to those of local communications.

- **Efficiency**

  An IPC protocol of a message-passing system can be made efficient by reducing the number of message exchanges, as far as practicable, during the communication process.

- **Reliability**

  A good IPC protocol can cope with failure problems and guarantees the delivery of a message. A reliable IPC protocol should also be capable of detecting and handling duplicate messages.

- **Correctness**

  Correctness is a feature related to IPC protocols for group communication. Issues related to correctness are as follows:

- **Atomicity**

  Atomicity ensures that every message sent to a group of receivers will be delivered to either all of them or none of them.

- **Ordered delivery**

  Ordered delivery ensures that messages arrive to all receivers in an order acceptable to the application.

- **Survivability**

  Survivability guarantees that messages will be correctly delivered despite partial failures of processes, machines, or communication links.

- **Flexibility**

  Not all applications require the same degree of reliability and correctness of the IPC protocol. The IPC protocols of a message passing system must be flexible enough to cater to the various needs of different applications.

- **Security**

  A good message passing system must also be capable of providing a secure end to end communication. Steps necessary for secure communication include the following:

  Authentication of the receiver of a message by the sender.

  Authentication of the sender of a message by its receivers.

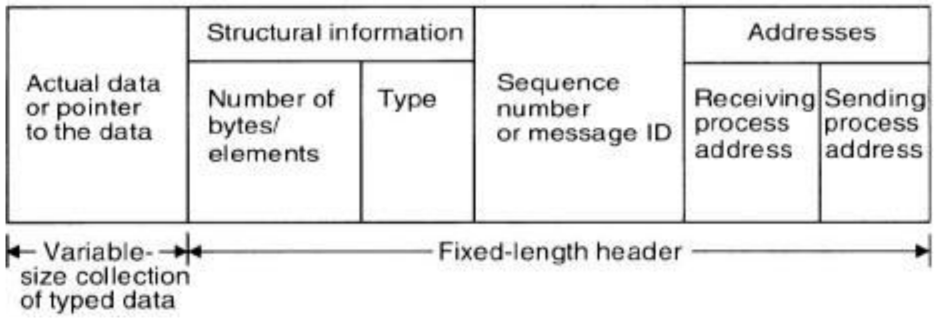
  Encryption of a message before sending it over the network.

- **Portability**

  The message passing system should itself be portable. It should be possible to easily construct a new IPC facility on another system by reusing the basic design of the existing message passing system.

Explain the Issues in IPC by Message Passing

- A message is a block of information formatted by a sending process in such a manner that it is meaningful to the receiving process.
- It consists of a fixed-length header and a variable-size collection of typed data objects.
- The header usually consists of the following elements:


Actual data or pointer to the data
Structural information
Number of bytes/ elements
Type
Sequence number or message ID
Addresses
Receiving process address
Sending process address
Variable-size collection of typed data
Fixed-length header

- **Address**

  It contains characters that uniquely identify the sending and receiving processes in the network.

- **Sequence number**

  This is the message identifier (ID), which is very useful for identifying lost messages and duplicate messages in case of system failures.
- **Structural information**

  This element also has two parts.
  - The **type** part specifies whether the data to be passed on to the receiver is included within the message or the message only contains a pointer to the data, which is stored somewhere outside the contiguous portion of the message.
  - The second part of this element specifies the length of the variable-size message data.

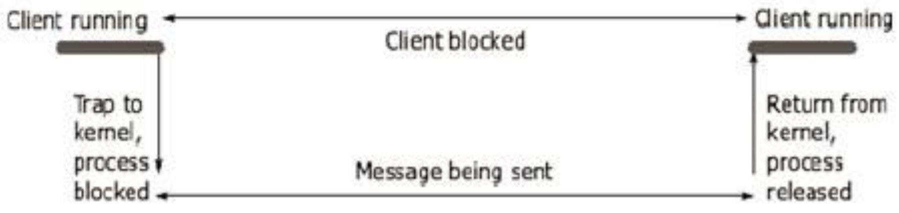
In the design of the IPC protocol for message passing system , the following important issues, need to be considered:

- Who is the sender?
- Who is the receiver?
- Is there one receiver or many receivers?
- Is the message guaranteed to have been accepted by receivers?
- Does the sender need to wait for the reply?
- What should be done if the catastrophic event such as node crash or a communication link failure occurs during the course of communication?
- What should be done if the receiver is not ready to accept the message: will the message be discarded or stored in a buffer? In the case of buffering what would be done if the buffer is full?
- If there are outstanding messages for a receiver, can it choose the order in which to service the outstanding messages?

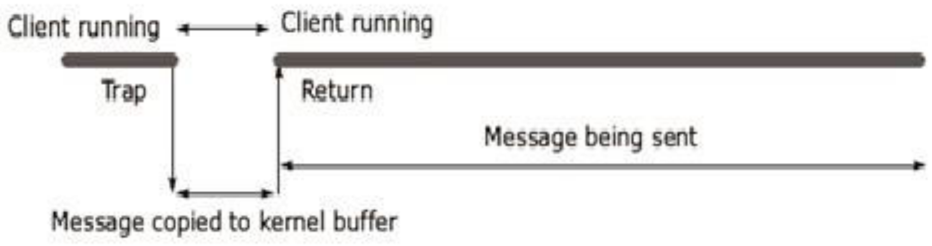
Explain about Synchronization

- A central issue in the communication structure is the synchronization imposed on the communicating processes by the communication primitives. The semantics used for synchronization may by broadly classified as
  - Blocking
  - Nonblocking

- Blocking Semantics:
    - A primitive is said to have blocking semantics if its invocation blocks the execution of its invoker.
    - In case of a blocking send primitive, after execution of the send statement, the sending process is blocked until it receives an acknowledgement from the receiver that the message has been received.


Client running
Client blocked
Client running
Trap to kernel, process blocked
Message being sent
Return from kernel, process released

    - In blocking receive primitive, after execution of the receive statement, the receiving process is blocked until it receives a message.
- Non Blocking Semantics
    - For nonblocking send primitive, after execution of the send statement, the sending process is allowed to proceed with its execution as soon as the message has been copied to a buffer.
    - For a nonblocking receive primitive, the receiving process proceeds with its execution after execution of the receive statement, which returns control almost immediately just after telling the kernel where the message buffer is.



    - An important issue in a nonblocking receive primitive is how the receiving process knows that the message has arrived in the message buffer.
    - One of the following two methods is commonly used for this purpose:

- Polling.

  In this method, a test primitive is provided to allow the receiver to check the buffer status.

- Interrupt.

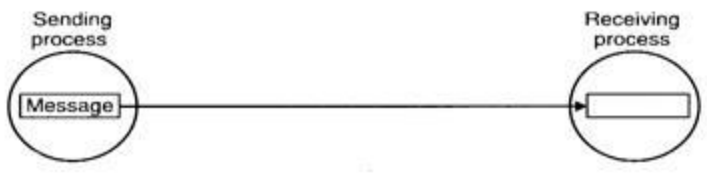
  In this method, when the message has been filled in the buffer and is ready for use by the receiver, a software interrupt is used to notify the receiving process.

Explain Buffering in communication process

- In the standard message passing model, messages can be copied many times: from the user buffer to the kernel buffer (the output buffer of a channel), from the kernel buffer of the sending computer (process) to the kernel buffer in the receiving computer (the input buffer of a channel), and finally from the kernel buffer of the receiving computer (process) to a user buffer. Buffering can be of following types.

Null Buffer (No Buffering)

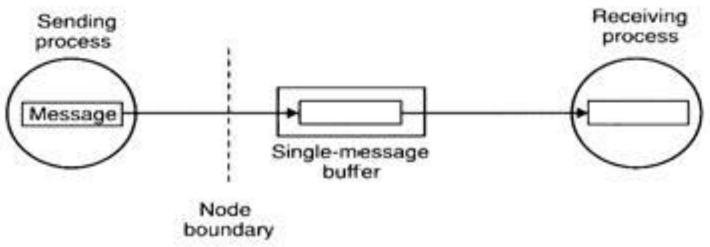
- In this case, there is no place to temporarily store the message. Hence one of the following implementation strategies may be used:
- The message remains in the sender process's address space and the execution of the send is delayed until the receiver executes the corresponding receive.
- The message is simply discarded and the time-out mechanism is used to resend the message after a timeout period.
- The sender may have to try several times before succeeding.



Single Message Buffer

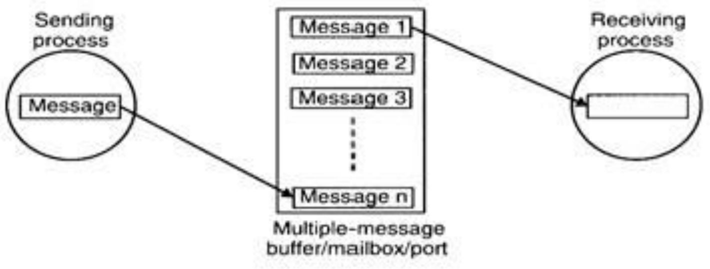
- In single-message buffer strategy, a buffer having a capacity to store a single message is used on the receiver's node.
- This strategy is usually used for synchronous communication, an application module may

have at most one message outstanding at a time.



Finite Bound Buffer or Multi buffer

- Systems using asynchronous mode of communication use finite-bound buffers, also known as multiple-message buffers.
- In this case message is first copied from the sending process's memory into the receiving process's mailbox and then copied from the mailbox to the receiver's memory when the receiver calls for the message.
- When the buffer has finite bounds, a strategy is also needed for handling the problem of a possible buffer overflow.



Unbounded Capacity Buffer

- In the asynchronous mode of communication, since a sender does not wait for the receiver to be ready, there may be several pending messages that have not yet been accepted by the receiver.
- Therefore, an unbounded-capacity message-buffer that can store all unreceived messages is needed to support asynchronous communication with the assurance that all the messages sent to the receiver will be delivered.
- Unbounded capacity of a buffer is practically impossible.

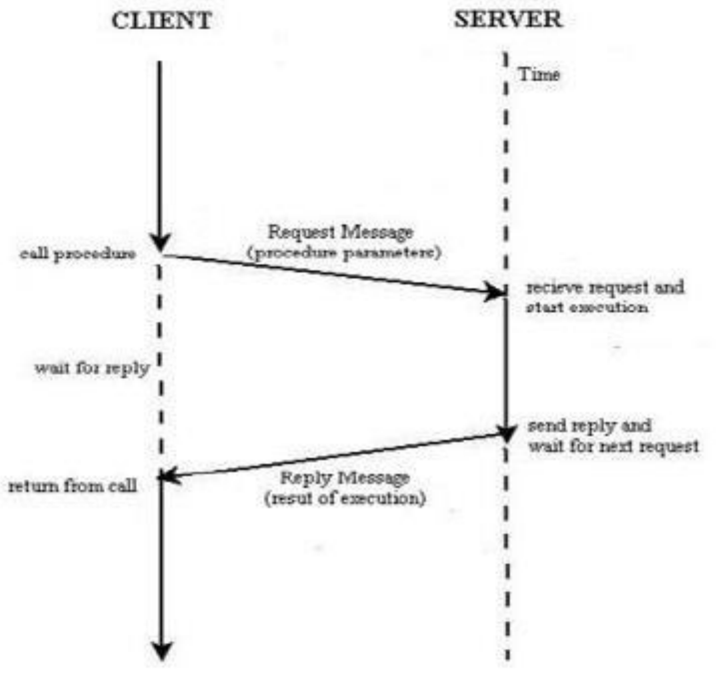
What is RPC? Explain its message procedure?

Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer on a network without having to understand the network's details. A procedure call is also sometimes known as a function call or a subroutine call.

RPC uses the client-server model. The requesting program is a client and the service providing program is the server.

RPC message procedure

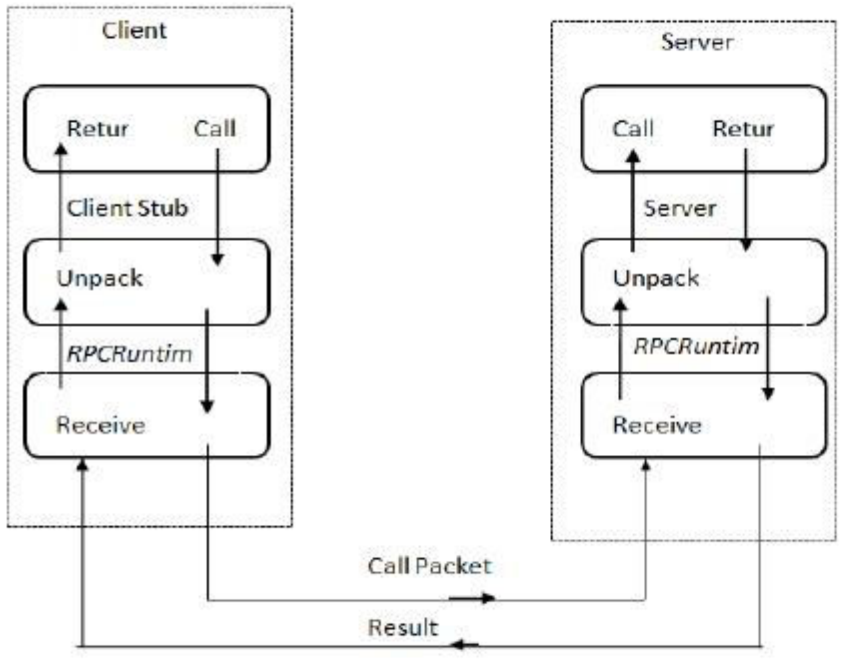
When program statements that use RPC framework are compiled into an executable program, a stub is included in the compiled code that acts as the representative of the remote procedure code. When the program is run and the procedure call is issued, the stub receives the request and forwards it to a client 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.

Explain RPC Implementation Mechanism

- To achieve the goal of semantic transparency, the implementation of an RPC mechanism is based on the concept of stubs, which provide a perfectly normal (local) procedure call abstraction by concealing from programs the interface to the underlying RPC system.
- To conceal the interface of the underlying RPC system from both the client and server processes, a separate stub procedure is associated with each of the two processes.
- To hide the existence and functional details of the underlying network, an RPC communication package (known as RPCRuntime) is used on both the client and server sides.



- Implementation of an RPC mechanism usually involves the following five elements of program.
    1. The client
    2. The client stub
    3. The RPCRuntime
    4. The server stub
    5. The server

- The client, the client stub, and one instance of RPCRuntime execute on the client machine, while the server, the server stub, and another instance of RPCRuntime execute on the server machine.

➢ **Client** :

- o The client is a user process that initiates a remote procedure call.
- o To make a remote procedure call, the client makes a perfectly normal local call that invokes a corresponding procedure in the client stub.

➢ **Client Stub :**

The client stub is responsible for carrying out the following two tasks :

- o On receipt of a call request from the client, it packs a specification of the target procedure and the arguments into a message and then asks the local RPCRuntime to send it to the server stub.
- o On receipt of the result of procedure execution, it unpacks the result and passes it to the client.

➢ **RPCRuntime :**

- o The RPCRuntime handles transmission of messages across the network between client and server machines.
- o It is responsible for retransmissions, acknowledgements, packet routing, and encryption.
- o The RPCRuntime on the client machine receives the call request message from the client stub and sends it to the server machine.
- o It also receives the message containing the result of procedure execution from the server machine and passes it to the client stub.
- o On the other hand, the RPCRuntime on the server machine receives the message containing the result of procedure execution from the server stub and sends it to the client machine.
- o It also receives the call request message from the client machine and passes it to the server stub.

➢ **Server Stub :**

- o The job of the server stub is very similar to that of the client stub. It performs the following two tasks :

- On the receipt of the call request message from the local RPCRuntime, the server stub unpacks it and makes a perfectly normal call to invoke the appropriate procedure in the server.
- On receipt of the result of procedure execution from the server, the server stub packs the result into a message and then asks the local RPCRuntime to send it to the client stub.

➢ **Server :**

- On receiving a call request from the server stub, the server executes the appropriate procedure and returns the result of procedure execution to the server stub.

What is stub? How to generate stubs?

- A stub, in the context of distributed computing, is a piece of code that is used to convert parameters during a remote procedure call (RPC). The parameters used in a function call have to be converted because the client and server computers use different address spaces. Stubs perform this conversion so that the remote server computer perceives the RPC as a local function call.
- Stub libraries are generally installed on the client and server. Client stubs convert parameters used in function calls and reconvert the result obtained from the server after function execution. Server stubs, on the other hand, reconvert parameters passed by clients and convert results back after function execution.
- Stubs are generated either **manually** or **automatically**. In a manual generation, a remote procedure call implementer provides translation functions, from which a user constructs stubs. They handle complex parameter types. Automatic stub generation is commonly used to generate stubs. They use **I**nterface **D**efinition **L**anguage (IDL) to define client and server interfaces. An interface is mainly a list procedure names supported by the interface, together with the types of their arguments and results.

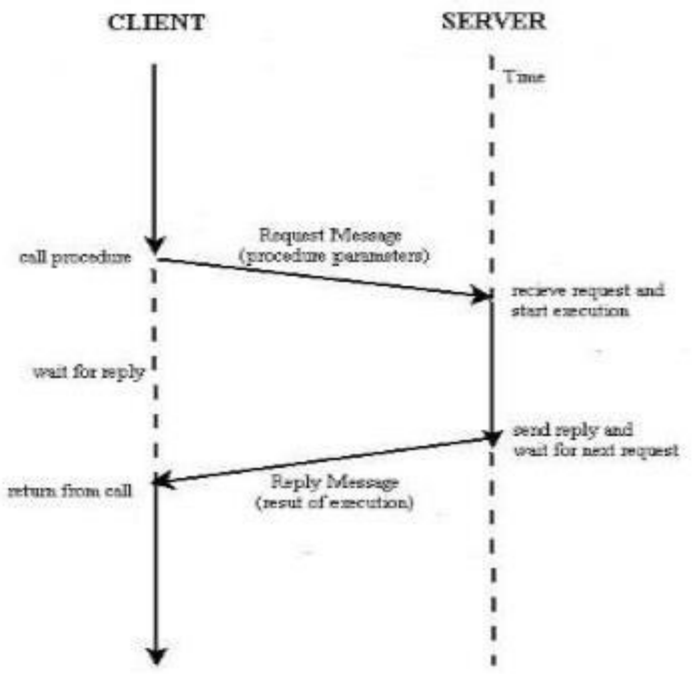
How transparency is achieved in RPC?

- A transparent RPC is one in which the local and remote procedure calls are indistinguishable to programmers. The main issue in designing an RPC facility is its transparent property.

There are two types of transparencies required:

- Syntactic transparency: A remote procedure and a local procedure call should have the same syntax.
- Semantic transparency: The semantics of a remote procedure call and local procedure call are identical.

Achieving Syntactic transparency is not an issue but semantic transparency is difficult due to some differences between remote procedure calls and local procedure calls.



- The calling procedure should not be aware that the called procedure is executing on a different machine.
- When read is a remote procedure a different version of read called client stub is put in the library. It is called using the calling sequence.
- The parameters are packed into a message and sent to the server. When the message arrives at the server, the server's operating system passes it to server stub which calls receive and blocks incoming messages.
- The server unpacks parameters and calls server procedure and performs its work and returns the result to caller in the usual way of packing the result and sending it to client stub.

- When the client receives the message from server, the operating system of client sees that it is addressed to client process. When the caller gets control following the call to read, all it knows is that its data are available.
- It has no idea that the work was done remotely instead of the local operating system. In this way transparency is achieved.

Explain different RPC messages?

- RPC system is independent of transport protocols and is not concerned as to how a message is passed from one process to another.
- Types of messages involved in the implementation of RPC system:

  1. *Call messages* that are sent by the client to the server for requesting execution of a particular remote procedure
  2. *Reply messages* that are sent by the server to the client for returning the result of remote procedure execution

- **Call Messages**

  Components necessary in a call message are

  1. The id. Information of the remote procedure to be executed.

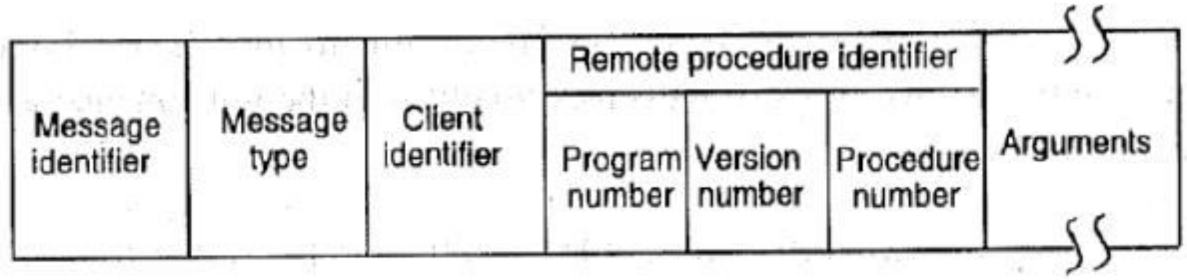
  2. The arguments necessary for the execution.

  3. Message Identification field consists of a sequence number for identifying lost and duplicate messages.

  4. Message Type field: whether call or reply messages.

  5. Client Identification Field allows the server to:

  - Identify the client to whom the reply message has to be returned and
  - To allow server to authenticate the client process.



○ **Reply Messages**

1. Sent by the server to the client for returning the result of remote procedure execution.
2. Conditions for unsuccessful message sent by the server:
   - The server finds that the call message is not intelligible to it.
   - Client is not authorized to use the service.
   - Remote procedure identifier is missing.
   - The remote procedure is not able to decode the supplied arguments.
   - Occurrence of exception condition.

➢ Reply message formats

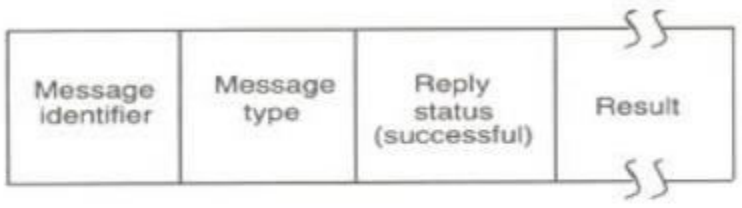


Figure 1 successful

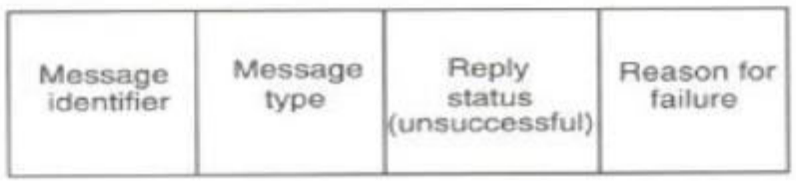


Figure 2 unsuccessful

Explain the Call Semantics in RPC

The different types of call semantics used in RPC systems are described below.

➢ Possibly or May be Call Semantics
➢ Last One Call Semantics.
➢ Lost of Many Call Semantics
➢ At Least Once Call Semantics
➢ Exactly one semantics.

- **Possibly or May be Call Semantics**
  ○ This is the weakest semantics and is not really appropriate to RPC. In this method, to prevent the caller from waiting indefinitely for a response from the callee a timeout

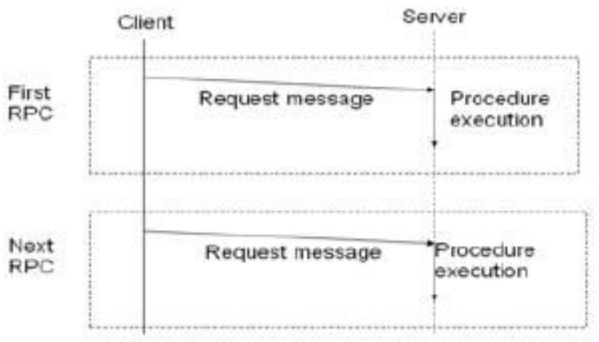
mechanism is used.
  - The caller waits until a predetermined timeout period and then continues with its execution.
  - Therefore the semantics does not guarantee anything about the receipt of the call message or the procedure execution by the caller.
- **Last One Call Semantics.**
  - It uses the idea of retransmitting the call message based on timeouts until a response it received by the caller.
  - The calling of the remote procedure by the caller, the execution of the procedure by callee, and the return of the result to the caller will eventually be repeated until the result of procedure execution is received by the caller.
  - Last one semantics can be easily achieved in the way described above when only two processors are involved in the RPC.
- **Lost of Many Call Semantics**
  - A simple way to neglect orphan calls is to use call identifiers to uniquely identify each call.
  - When a call is repeated, it is assigned a new call identifier.
  - Each response message has the corresponding call identifier associated with it.
  - A caller accepts a response only if the call identifier associated with it matches with the identifier of most recently repeated call, otherwise it ignores the response message.
- **At Least Once Call Semantics**
  - This semantic is weaker than last of many semantics.
  - It just guarantees that the call is executed one or more times but does not specify which results are returned to caller.
  - It can be implemented simply by using timeout based retransmission without caring for the orphan calls.
  - For nested calls, if there are any orphan calls, it takes the result of the first response message and ignores the others, whether or not the accepted response is from an orphan.
- **Exactly one semantics.**
  - This is the strongest and the most desirable call semantics because it eliminates the

possibility of a procedure being executed more than once no matter how many time a call is retransmitted.

Explain Communication Protocols in RPC

Based on the needs of different systems, several communication protocols have been proposed for use in RPC which are mentioned below:

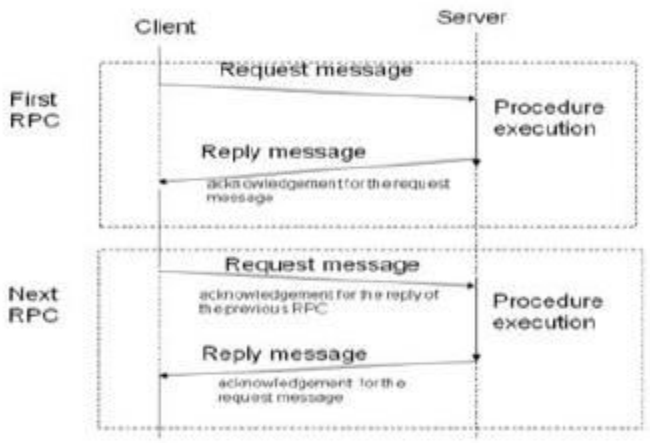
➤ **The Request Protocol**



- This protocol is also called as R (request) protocol.
- It is used in RPC when a called procedure has nothing to return as a result of execution and the requirement of client confirmation about procedure execution is not needed.
- As no acknowledgement or reply message is involved, only single message is transmitted from client to server.
- RPC that uses the R protocol is known as asynchronous RPC which helps to improve the combined performance of the client and server. This is done because the client does not wait for a reply and server does not need to send a reply.
- Asynchronous RPC with unreliable transport protocol are generally used in implementing periodic update services. Distributed system window is one of its applications.

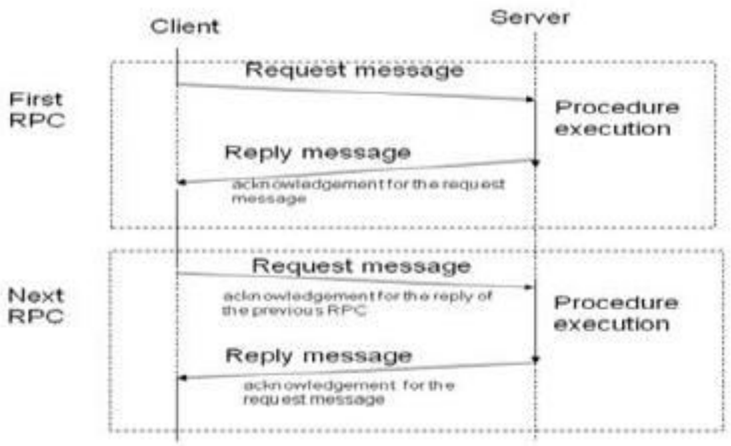
➤ **The Request/Reply protocol**

- This protocol is also known as RR(request/reply) protocol. It is useful for designing systems which involve simple RPCs.
- In a simple RPC all the arguments and result fit in a single packet buffer while the call duration and intervals between calls are short.
- This protocol is based on the idea of using implicit acknowledgement to eliminate explicit acknowledgement messages.


Client
Server
First RPC
Request message
Procedure execution
Reply message
acknowledgement for the request message
Next RPC
Request message
acknowledgement for the reply of the previous RPC
Procedure execution
Reply message
acknowledgement for the request message

- In this protocol a server reply is considered as an ACK for a clients request and a subsequent call from a client is considered as ACK of the client's previous call.
- Timeout-and-retires technique is used with RR protocol for failure handling. Retransmission of request message is done when there is no response.

➢ **<u>The Request/Reply/Acknowledgement-Reply Protocol</u>**


Client
Server
First RPC
Request message
Procedure execution
Reply message
acknowledgement for the request message
Next RPC
Request message
acknowledgement for the reply of the previous RPC
Procedure execution
Reply message
acknowledgement for the request message

- This protocol is also known as RRA (request/reply/acknowledge-reply) protocol.
- RR protocol implements exactly once semantics which requires storage of a lot of information in the server cache and can lead to loss of replies that have not been delivered.
- In this clients acknowledge the receipt of reply messages and the server deletes information from its cache only after it receives an acknowledgement from client.
- Sometimes the reply acknowledgement message may get lost therefore RRA protocol

needs a unique ordered message identifiers. This keeps a track of the acknowledgement series sent.

Explain Client – Server Binding in RPC

- Client Stub must know the location of a server before RPC can take place between them. Process by which client gets associated with server is known as BINDING.
- Servers export their operations to register their willingness to provide services and Clients import operations, asking the RPCRuntime to locate a server and establish any state that may be needed at each end.

Issues for client-server binding process:

- How does a client specify a server to which it wants to get bound?
- How does the binding process locate the specified server?
- When is it proper to bind a client to a server?
- Is it possible for a client to change a binding during execution?
- Can a client be simultaneously bound to multiple servers that provide the same service?

Server Naming

- The naming issue is the specification by a client of a server with which it wants to communicate.
- Interface name of a server is its unique identifier. It is specified by type & instance.
- Type specifies the version number of different interfaces, Instance specifies a server providing the services.
- Interface names are created by the users, not by the RPC package. The RPC package only dictates the means by which an importer uses the interface name to locate an exporter.

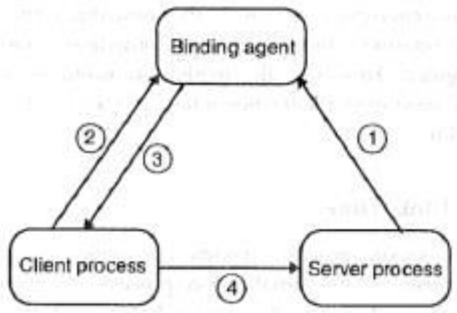
Server locating

- The locating methods are:
  1. Broadcasting
  2. Binding agent
- Broadcasting :-

  A message to locate a node is broadcast to all the nodes from the client node. Node having desired server returns a response message. Suitable for small networks.

- Binding Agent :-

  A name server used to bind a client to a server by providing the client with the location information. Maintains a binding table containing mapping of server interface name to its location. The table contains identification information and a handle to locate it. The location of the binding agent (having a fixed address) is known to all nodes by using a broadcast message.



(1) The server registers itself with the binding agent.

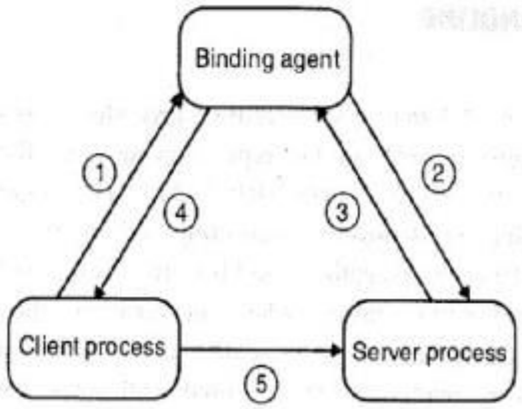
(2) The client requests the binding agent for the server's location.

(3) The binding agent returns the server's location information to the client.

(4) The client calls the server.

Binding Time

- Binding at compile time :- Servers network address can be compiled into client code.
- Binding at link time :- Client makes an import request to the binding agent for the service before making a call. Servers handle cached by the client to avoid contacting binding agent for subsequent calls. This method is suitable for those situations in which a client calls a server several times once it is bound to it.
- Binding at call time:- A client is bound to a server at the time when it calls the server for the first time during its execution. The commonly used approach for binding at call time is the indirect call method.



① The client process passes the server's interface name and the arguments of the RPC call to the binding agent.

② The binding agent sends an RPC call message to the server, including in it the arguments received from the client.

③ The server returns the result of request processing to the binding agent.

④ The binding agent returns this result to the client along with the server's handle.

⑤ Subsequent calls are sent directly from the client process to the server process.

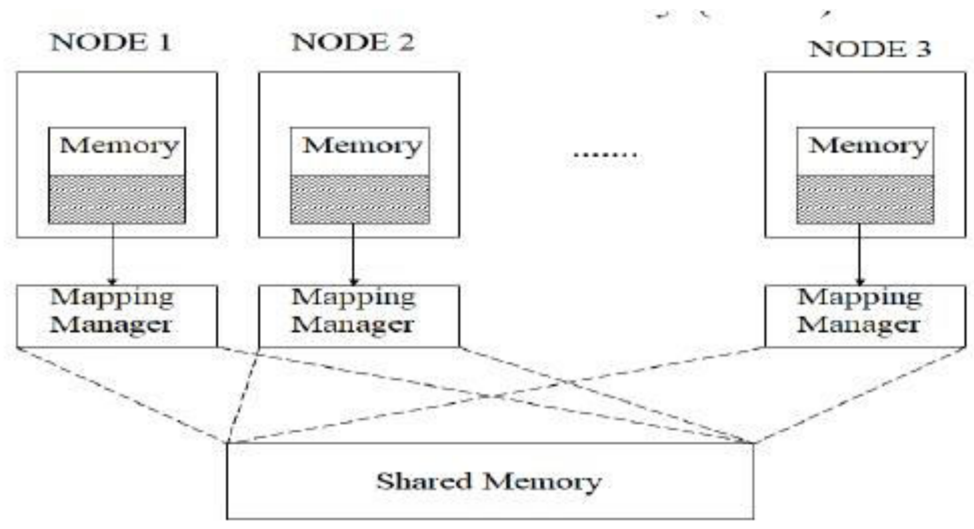
Changing bindings

- The client or server of a connection may wish to change the binding at some instance of time due to some change in the system state.
- When a binding is altered by the concerned server, it is important to ensure that any state data held by the server is no longer needed or can be duplicated in the replacement server.

Introduction, Design and implementation of DSM system, Granularity and Consistency Model, Advantages of DSM, Clock Synchronization, Event Ordering, Mutual exclusion, Deadlock, Election Algorithms.

Explain Design and implementation of DSM

- The distributed shared memory (DSM) implements the shared memory model in distributed systems, which have no physical shared memory
- The shared memory model provides a virtual address space shared between all nodes
- 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.
- A distributed shared memory system implements the shared-memory model on a physically distributed memory system.



Methods of achieving DSM

- There are usually two methods of achieving distributed shared memory:
- Hardware, such as cache coherence circuits and network interfaces
- software

Software DSM implementation

- There are three ways of implementing a software distributed shared memory:
- Page based approach using the system‟s virtual memory;
- Shared variable approach using some routines to access shared variables;
- Object based approach ideally accessing shared data through object-oriented discipline.

Advantages

- Data sharing is implicit, hiding data movement (as opposed to „Send‟/„Receive‟ in message passing model)
- Passing data structures containing pointers is easier (in message passing model data moves between different address spaces)
- Moving entire object to user takes advantage of locality difference
- Less expensive to build than tightly coupled multiprocessor system: off-the-shelf hardware, no expensive interface to shared physical memory
- Very large total physical memory for all nodes: Large programs can run more efficiently
- No serial access to common bus for shared physical memory like in multiprocessor systems
- Programs written for shared memory multiprocessors can be run on DSM systems with minimum changes

Disadvantages

- Generally slower to access than non-distributed shared memory
- Must provide additional protection against simultaneous accesses to shared data

Explain Design and Implementation Issues of DSM

Granularity:

- o Granularity refers to the blocks size of a DSM system, that is, to the unit of sharing and the unit of data transfer across the network when a network block fault occurs.
- o Possible units are a few words, a page, or a few pages.
- o Selecting proper block size is an important part of the design of a DSM system because block size is usually a measure of the granularity of parallelism explored and the amount

of network traffic generated by network block faults.

Structure of shared-memory space.

- Structure refers to the layout of the share data in memory.
- The structure of the shared-memory space of a DSM system is normally dependent on the type of applications that the DSM system is intended to support.

Memory coherence and access synchronization

- In a DSM system that allows replication of shared data items, copies of shared data items may simultaneously be available in the main memories of a number of nodes.
- The main problem is to solve the memory coherence problem that deals with the consistency of a piece of shared data lying in the main memories of two or more nodes.
- Since different memory coherence protocols make different assumptions and trade-offs, the choice is usually dependent on the pattern of memory access.
- In a DSM system, concurrent accesses to shared data may be generated.
- A memory coherence protocol alone is not sufficient to maintain the consistency of shared data.
- Synchronization primitives, such as semaphores, event count, and lock, are needed to synchronize concurrent accesses to shared data.

Data location and access.

- To share data in a DSM system, it should be possible to locate and retrieve the data accessed by a user process.
- Therefore, a DSM system must implement some form of data block locating mechanism in order to service network data block faults to meet the requirement of the memory coherence semantics being used.

Replacement strategy.

- If the local memory of a node is full, a cache miss at that node implies not only a fetch of the accessed data block from a remote node but also a replacement.
- A data block of the local memory must be replaced by the new data block.
- Thus, a cache replacement strategy is also necessary in the design of a DSM system.

Thrashing

- In a DSM system, data blocks migrate between nodes on demand.
- Therefore, if two nodes complete for write access to a single data item, the corresponding

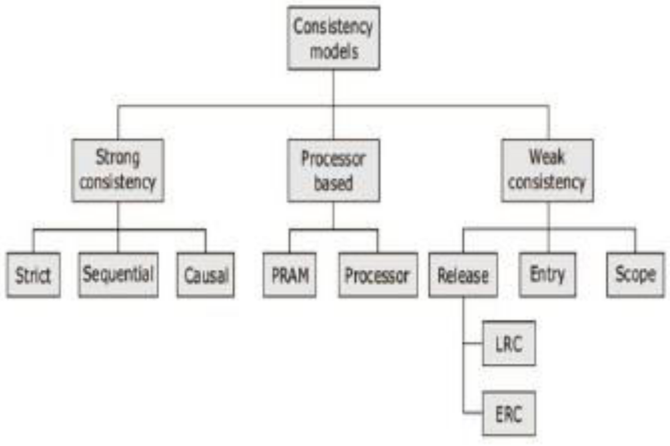
data block may be transferred back and forth at such a high rate that no real work can get gone.

- A DSM system must use a policy to avoid this situation usually known as thrashing.

Explain Consistency Models in Distributed Shared Memory (DSM)?

- Refers to how recent the shared memory updates are visible to all the other processes running on different machines
- It is defined as a set of rules that applications must obey if they want to DSM system to provide the degree of consistency guaranteed by the consistency model.

1. Strict Consistency Model
2. Sequential Consistency Model
3. Causal Consistency Model
4. Pipelined Random Access Memory Consistency Model
5. Processor Consistency Model
6. Weak Consistency Model
7. Release Consistency Model



Strict Consistency Model

- The strict consistency model is the strongest form of memory coherence, having the most stringent consistency requirement.
- A shared-memory system is said to support the strict consistency model if the value returned by a read operation on a memory address is always the same as the value written by the most recent write operation to that address, irrespective of the locations of the processes performing the read and write operations.
- All writes instantaneously become visible to all processes.
- As the existence of an absolute global time in a distributed system is not possible, and hence implementation of strict consistency model for DSM system is practically not possible.

Sequential Consistency Model

- The sequential consistency model was proposed by " Lamport".
- A shared-memory system is said to support the sequential consistency model if all processes see the same order of all memory access operations on the shared memory.
- The exact order in which the memory access operations are inserted does not matter.
- If the three operations read (r1), write (w1) , read(r2) are performed on a memory address in that order , any of the orderings (r1,w1,r2) , (r1,r2,w1), (w1,r1,r2), (w1,r2,r1), (r2,r1,w1), (r2,w1,r1) of the three operations is acceptable provided all processes see the same ordering.
- If one process sees one of the orderings of the three operations and another process sees a different one, the memory is not a sequentially consistent memory.
- A DSM system supporting the sequential consistency model can be implemented by ensuring that no memory operation is started until all the previous ones have been completed.

Causal Consistency Model

- Proposed by Hutto and Ahamad (1990)
- Unlike the sequential consistency model, in the causal consistency model, all processes see only those memory reference operations in the same (correct) order that are potentially causally related.
- Memory reference operations that are not potentially causally related may be seen by different processes in different orders.
- A memory reference operation (Read / write) is said to be potentially causally related to another memory reference operation if the first one might have been influenced in any way by the second one.
- Note that "correct order" means that if a write operation (w2) is causally related to another write operation (w1), the acceptable order is (w1, w2) because the value written by w2 might have been influenced in some way by the value written by w1.
- Therefore, (w2, w1) is not an acceptable order.

Pipelined Random Access Memory Consistency Model

- The pipelined random-access memory (PRAM) consistency model, proposed by Lipton and Sandberg provides a weaker consistency semantics than other consistency models.
- It only ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed as if all the write operations performed by a single process are in a pipeline.
- Write operations performed by different processes may be seen by different processes in different orders.
- For example, if w11 and w12 are two write operations performed by a process P1 in that order and w21 and w22 are two write operations performed by a process P2 in that order, a process P3 may see them in the order [w11, w12), (w21, w22)] and another process P4 may see them in the order [w21, w22), (w11, w22)].

Processor Consistency Model

- The processor consistency model, proposed by Goodman is very similar to the PRAM consistency model with an additional restriction of memory coherence.
- A processor consistent memory is both coherent and adheres to the PRAM consistency model.
- Memory coherence means that for any memory location all processes agree on the same order of all write operations to that location.
- A processor consistency ensures that all write operations performed on the same memory location (no matter by which process they are performed) are seen by all processes in the same order.
- Therefore, in the example given for PRAM consistency, if w12 and w22 are write operations for writing to the same memory location x, all processes must see them in the same order-w12 before w22 or w22 before w22.
- For processor consistency both processes P3 and P4 must see the write operations in the same order, which may be either [w11, w22), (w21, w22), (w11, w12)].

Weak Consistency Model

- The weak consistency model, proposed by Dubois, is designed to take advantage of the following two characteristics common to many applications:

1. It is not necessary to show the change in memory done by every write operation to other processes.
2. Isolated accesses to shared variables are rare.

- A DSM system that supports the weak consistency model uses a special variable called a synchronization variable.

Release Consistency Model

- Since a single synchronization variable is used in the weak consistency model, the system cannot know whether a process accessing a synchronization variable is entering a critical section or existing from a critical section.
- For better performance, the release consistency model provides a mechanism to clearly tell the system whether a process is entering a critical section or exiting from a critical section so that the system can decide and perform only either the first or the second operation when a synchronization variable is accessed by a process.
- This is achieved by using two synchronization variables instead of a single synchronization variable.
- **Acquire** is used by a process to tell the system that it is about to enter a critical section, so that the system performs only the second operation when this variable is accessed.
- On the other hand, **Release** is used by a process to tell the system that it has just exited a critical section.

Explain Granularity in DSM

- One of the most visible parameters to the chosen in the design of a DSM system is the block size. Several criteria for choosing this granularity parameter are described below.
    1. Factors Influencing Block Size Selection
    2. Using Page Size as Block Size
    3. Structure of shared memory

Factors Influencing Block Size Selection

In a typical loosely coupled multiprocessor system, sending large packets of data is not much more expensive than sending small ones. factors that influence the choice of block size are described below:

Paging overhead:

- Because shared-memory programs provide locality of reference, a process is likely to access a large region of its shared address space in a small amount of time.

Directory size:

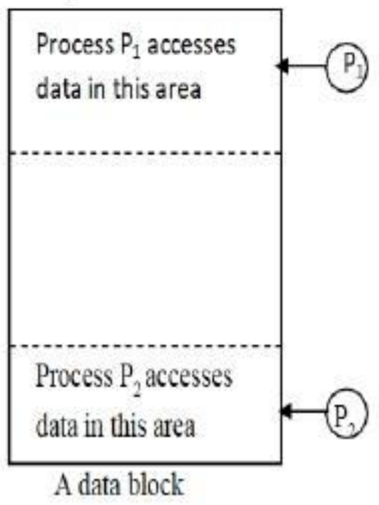
- Another factor affecting the choice of block size is the need to keep directory information about the blocks in the system.
- The larger the block size, the smaller the directory.

Thrashing:

- The problem of thrashing may occur when data items in the same data block are being updated by multiple nodes at the same time, causing large numbers of data block transfers among the nodes without much progress in the execution of the application.

False sharing

- False sharing occurs when two different processes access two unrelated variable that reside in the same data block


Process P_1 accesses data in this area
P_1
Process P_2 accesses data in this area
P_2
A data block

- In such a situation, even though the original variables are not shared, the data block appears to be shared by the two processes.

Using Page Size as Block Size

- The relative advantages and disadvantages of small and large block sizes make it difficult for a DSM designer to decide on a proper size.
- Using page size as the block size of a DSM system has the following advantages:
  - It allows the use of existing page-fault schemes to trigger a DSM page fault.
  - Thus memory coherence problems can be resolved in page-fault handlers.
  - It allows the access right control to be readily integrated into the functionality of the memory management unit of the system.
  - As long as a page can fit into a packet, page sizes do not impose undue communication overhead at the time of network page fault.
  - Experience has shown that a page size is a suitable data entity until with respect to memory contention.

Structure of shared memory

- Structure defines the abstract view of the shared-memory space to be presented to the application programmers of a DSM system.
- The three commonly used approaches for structuring the shared-memory space of DSM system are as follows :

1. No structuring.

- In this system, the shared-memory space is simply a linear array of words.
- An advantage of the use of unstructured shared-memory space is that it is convenient to choose any suitable page size as the unit of sharing and a fixed grain size may be used for all applications.
- It also allows applications to impose whatever data structures they want on the shared memory.

2. Structuring by data type.

- In this method, the shared-memory space is structured either as a collection of objects or as a collection of variables in the source language.
- The granularity in such DSM systems is an object or a variable.

3. Structuring as a database

- Another method is to structure the shared memory like a database.

- Its shared-memory space is ordered as an associative memory i.e a memory addressed by content rather than by name or address called a table space, which is a collection of immutable rows with typed data items in their fields.

Physical Clocks

- Every Uniprocessor system needs a timer mechanism to keep track of time for process execution and accounting for the time spent by the process for using various resources like CPU, I/O or even memory.
- In Distributed system, applications will have several processes running on different machines.
- Ideally global clock is required to coordinate all such processes.
- But in real systems, this is not possible.
- Each CPU has its own clock and it is required that all clocks in the system displays the same time.

Implementing Computer Clocks

- Nearly all computers have a circuit for keeping track of time.
- A computer timer is usually a precisely machined quartz crystal.
- When kept under tension, quartz crystals oscillate at a well-defined frequency that depends on the kind of crystal, how it is cut, and the amount of tension.
- Associated with each crystal are two registers: a **counter** and a **holding** register.
- Holding register holds some constant value.
- Counter register is initialized with holding registers value.
- Each oscillation of the crystal decrements the counter by one.
- When the counter gets to zero, an interrupt is generated and the counter is reloaded from the holding register.
- It is possible to program a timer to generate an interrupt 60 times a second, or at any other desired frequency.
- Each interrupt is called one clock tick.
- When the system is booted initially, it usually asks the operator to enter the date and time, which is then converted to the number of ticks after some known starting date and stored

in memory.

- At every clock tick, the interrupt service procedure adds one to the time stored in memory.
- With a single computer and a single clock, it does not matter much if this clock is off by a small amount.
- As soon as multiple CPUs are introduced, each with its own clock, the situation changes.
- Although the frequency at which a crystal oscillator runs is usually fairly stable, it is impossible to guarantee that the crystals in different computers all run at exactly the same frequency.
- In practice, when a system has n computers, all n crystals will run at slightly different rates, causing the clocks gradually to get out of sync and give different values when read out.
- This difference in time values is called clock skew.

What do you understand by clock synchronization in distributed systems? Explain one technique of logical clock synchronization.

- To synchronize logical clocks, Lamport defined a relation called happens-before.
- The expression a ->b is read as „a happens before b‟ and means that all processes agree that event a occurs then event b occurs.
- From the definition of the happened-before relation, the clock condition mentioned above is satisfied if the following conditions hold:
  I. C1- If a and b are two events within the same process Pi and a occurs before then Ci(a) <ci(b).< p="">
  II. C2- If a is message sent by process Pi and b is the receipt of that message by process Pj then Ci(a) <ci(b).< p="">
  III. C3- A clock Ci associated with a process Pi must always go forward, never backward. Correction to time of a logical clock must always be made by positive value not by subtracting.
- First two conditions are needed to satisfy clock condition while the third is needed for correct functioning of the system.

- Any algorithm used for implementing a set of logical clocks must satisfy all three conditions proposed by Lamport.
- The algorithm for assigning time to events using physical and logical clocks depending on the above conditions is as follows.

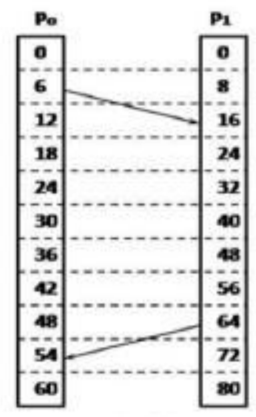


Fig (a)

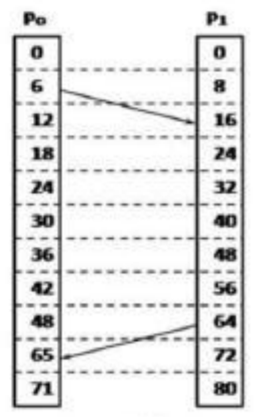


Fig (b)

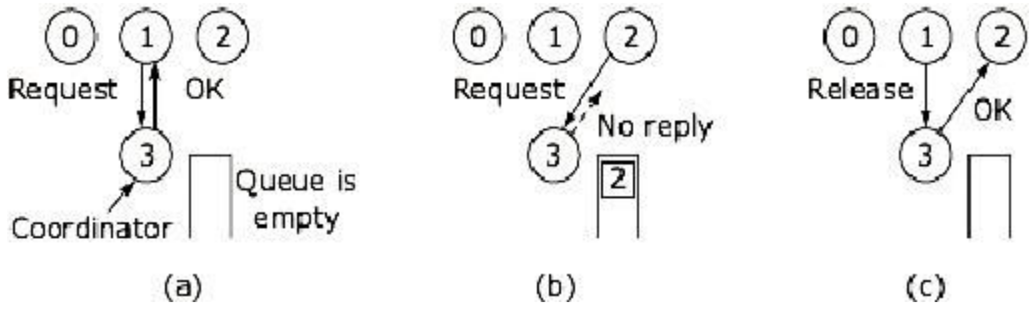
- Consider two processes in fig (a) the processes run on different machines each having a physical clock associated with it running at its own speed. For process Po the time ticked is 6 times and process P1 has the time ticked 8 times. The clocks run at constant rate but are different due to difference in the crystals.
- At time 6, Po sends message A to P1 and is received at time 16. Consider the time taken for message transfer as 10 is considerable.
- Now when Message B is sent from P2 at time 64 it takes 10 ticks to reach P1. So the message is received at time 54 by P1 because the time for P1 is slower. This value is certainly impossible and such a situation must be prevented.
- The solution for this is given by Lamports happens-before relation. Since B left at 64 it must arrive at 65 or later. There for the message also carries the sending time according to sender‟s clock.
- When a message arrives and the receiver‟s clock shows a prior value to the time the message was sent, the receiver fast forwards its clock to be one more than the sending time.
- In fig (b) we can see that message C now arrives at 65. With one small addition this algorithm meets the requirements for global time.
- This algorithm provides total ordering of all events in the system.

Explain Distributed Approach for providing Mutual Exclusion

- Systems involving multiple processes are often most easily programmed using critical regions.
- When a process has to read or update certain shared data structures, it first enters a critical region to achieve mutual exclusion and ensure that no other process will use the shared data structures at the same time.
- Mutual Exclusion ensures that no other process will use shared resources at same time.
- Following are the algorithm used for mutual exclusion
  1. Centralized Algorithm
  2. Distributed Algorithm
  3. Token Ring Algorithm.

Centralized Algorithm

- One process is elected as coordinator.
- Whenever process wants to enter a critical region , it sends request message 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 another process is already in critical regions, so it cannot be granted permission.



In the above example

a. Process 1 asks the coordinator for permission to enter a critical region. Permission is granted.

b. Process 2 then asks permission to enter the same critical region. The coordinator does not reply.

c. When process 1 exits the critical region, it tells the coordinator, which then replies to 2.

Distributed Algorithm:

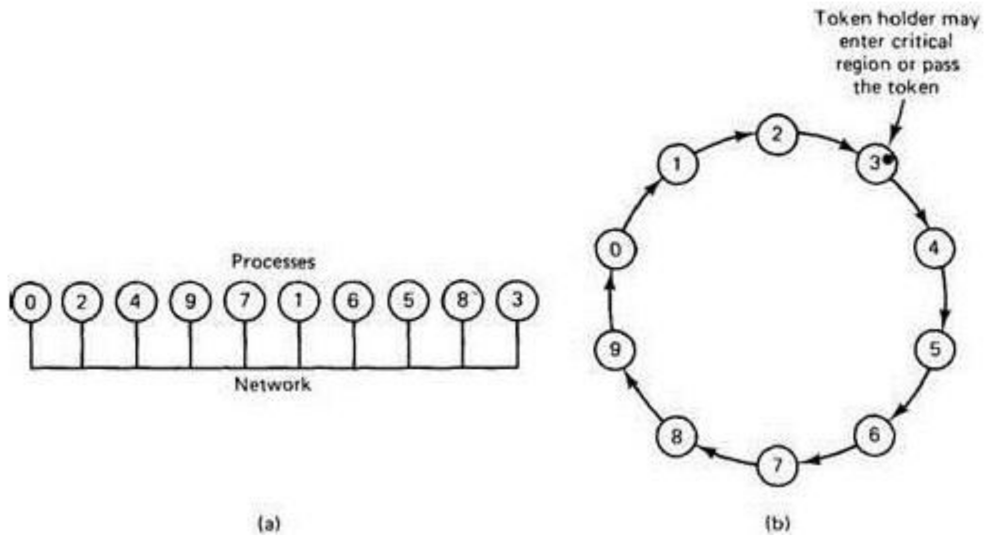
- Timestamps are used for distributed mutual exclusion.

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.
- If its own msg has lower timestamp, the receiver queues the incoming request and sends nothing.

Token Ring Algorithm

- We have a bus network, as shown in Fig.(a), (e.g., Ethernet), with no inherent ordering of the processes.
- In software, a logical ring is constructed in which each process is assigned a position in the ring, as shown in Fig. (b).
- Each process should know who is next in line after itself.



- When the ring is initialized, process 0 is given a token.
- The token circulates around the ring; it is passed from process k to process k+1 (modulo the ring size) in point-to-point messages.
- When a process acquires the token from its neighbor, it checks to see if it is attempting to enter a critical region.
- If so, the process enters the region, does all the work it needs to, and leaves the region.
- After it has exited, it passes the token along the ring.
- It is not permitted to enter a second critical region using the same token.
- If a process is handed the token by its neighbour and is not interested in entering a critical region, it just passes it along.
- As a consequence, when no processes want to enter any critical regions, the token just circulates at high speed around the ring.
- Only one process has the token at any instant, so only one process can be in a critical region.
- Since the token circulates among the processes in a well-defined order, starvation cannot occur.

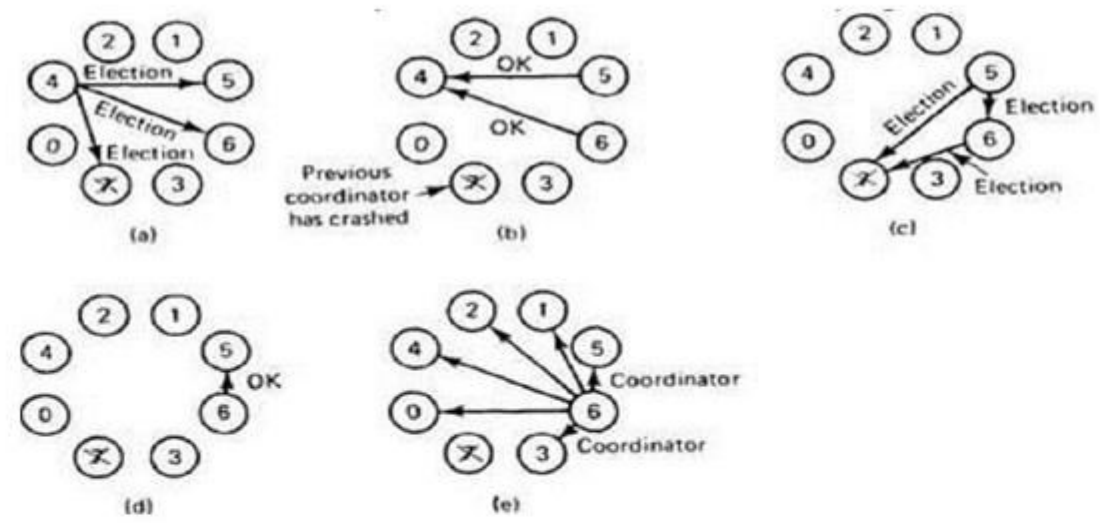
Explain Election algorithms in detail

- Election algorithms are meant for electing a coordinator process from among the currently running processes in such a manner that at any instance of time there is a single coordinator for all processes in the system.
- Therefore, whenever initiated, an election algorithm basically finds out which of the currently active processes has the highest priority number and then informs this to all other active processes.

Bully Algorithm

- When a process notices that the coordinator is no longer responding to requests, it initiates an election. A process, P, holds an election as follows:
  - P sends an ELECTION message to all processes with higher numbers.
  - If no one responds, P wins the election and becomes coordinator.
  - If one of the higher-ups answers, it takes over. P's job is done.

- At any moment, a process can get an ELECTION message from one of its lower-numbered colleagues.
- When such a message arrives, the receiver sends an OK message back to the sender to indicate that he is alive and will take over.
- The receiver then holds an election, unless it is already holding one.
- Eventually, all processes give up but one, and that one is the new coordinator.
- It announces its victory by sending all processes a message telling them that starting immediately it is the new coordinator.
- Thus the biggest guy in town always wins, hence the name "bully algorithm."



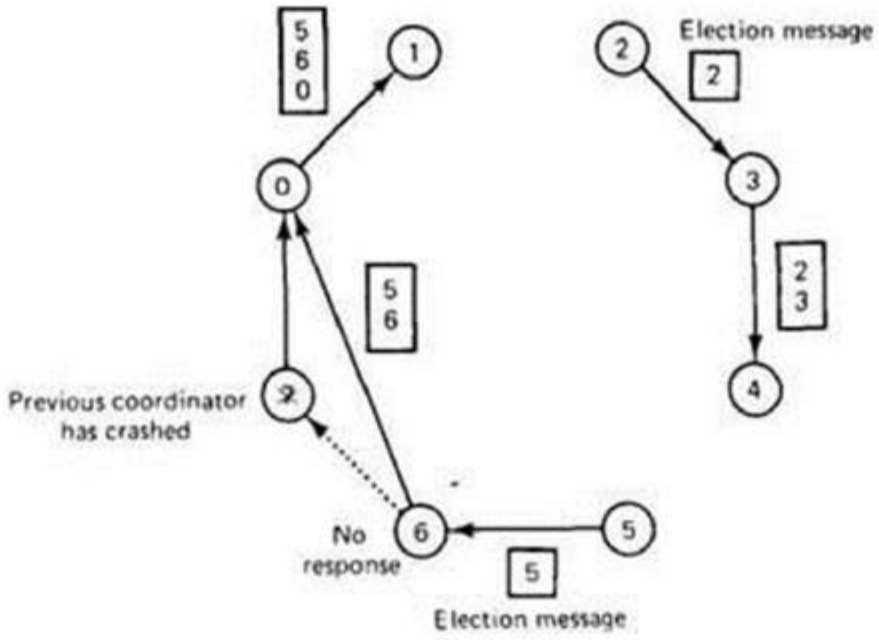
a. Process 4 holds an election.
b. Processes 5 and 6 respond, telling 4 to stop.
c. Now 5 and 6 each hold an election.
d. Process 6 tells 5 to stop.
e. Process 6 wins and tells everyone.

Ring Algorithm

- This algorithm uses a ring for its election but does not use any token. In this algorithm it is assumed that the processes are physically or logically ordered so each processor knows its successor.
- When any process notices that a coordinator is not functioning, it builds an ELECTION message containing its own process number and sends the message to its successor. If the successor is down the sender skips over the successor and goes to the next member along

the ring until a process is located.

- At each step the sender adds its own process number to the list in the message making itself a candidate to elected as coordinator
- The message gets back to the process that started it and recognizes this event as the message consists its own process number.
- At that point the message type is changed to COORDINATOR and circulated once again to inform everyone who the coordinator is and who are the new members. The coordinator is selected with the process having highest number.
- When this message is circulated once it is removed and normal work is preceded.



What is a Dead Lock? What are the common strategies used for handling deadlock in Distributed Systems.?

A deadlock is a condition in a system where a set of processes (or threads) have requests for resources that can never be satisfied. Essentially, a process cannot proceed because it needs to obtain a resource held by another process but it itself is holding a resource that the other process needs.

Handling deadlocks in distributed systems is complex because the resources, the processes and other information are scattered on different nodes of a system. The commonly used strategies to handle deadlock are:

1. Avoidance
2. Prevention
3. Deadlock Detection
4. Recovery from Deadlock

Avoidance

Deadlock avoidance methods use some advance knowledge of the resource usage of process to predict the future state of the system for avoiding allocations that can finally lead to a deadlock. Deadlock avoidance algorithms are usually in the following steps:

1. When a process requests for a resource, if the resource is available for allocation it is not immediately allocated to the process rather the system assumes that the request is granted.
2. Using advance knowledge of resource usage of processes and assumptions of step 1 the system analysis whether granting the process request is safe or unsafe.
3. The resource is allocated to the process only if the analysis of step 2 shows that it is safe to do so otherwise the request is postponed.

Deadlock avoidance algorithm basically perform resource allocation is such a manner that ensures the system will always remain in safe state.

Prevention

This approach is based on the idea of designing the system in such a way that deadlocks become impossible. It differs from the avoidance and detection where no runtime testing of potential allocations need to be performed.

Mutual exclusion, hold-and-wait, no anticipation and circular-wait are the four necessary conditions for a deadlock to occur. If one of these conditions can never be satisfied then deadlock can be prevented. For this there are three prevention methods namely:

I. Collective requests

These methods deny the hold and wait condition by ensuring that whenever a process requests a resource it does not hold any other resource. Various resource allocation policies can be used.

II. Ordered Requests

In this method circular-wait is denied such that each resource type is assigned a unique global number to impose total ordering of all resource types.

III. Preemption

A preemptable resource is one whose state can be easily saved and restored later. Deadlocks can be prevented using resource allocation policies to deny no-preemption condition.

Deadlock Detection

In this approach for deadlock detection, the system does not make any attempt to prevent deadlock but allows processes to request resources and wait for each other in uncontrolled manner.

Deadlock detection algorithms are same in centralized and distributed systems. Deadlock detection algorithms get simplified by maintaining Wait-for-graph (WFG) and searching for cycles.

I. Centralized Approach for Deadlock Detection

In this approach a local coordinator at each site maintains a WFG for its local resources and a central coordinator for constructing the union of all the individual WFGs.

The central coordinator constructs the global WFG from the information received from the local coordinators of all sites.

II. Hierarchical Approach for Deadlock Detection

The hierarchical approach overcomes drawbacks of the centralized approach. This approach uses a logical hierarchy of deadlock detectors called as controllers.

Each controller detects only those deadlocks that have the sites falling within the range of the hierarchy. Global WFG is distributed over a number of different controllers in this approach.

III. Fully Distributed Approaches for Deadlock Detection

In this approach each site shares equal responsibility for deadlock detection. The first algorithm is based on construction of WFG and second one is a probe-based algorithm.

Recovery from Deadlock

When a system chooses to use the detection and recovery strategy for handling deadlocks it is not enough to only detect deadlocks. The system must also be able to recover from a detected deadlock. One of the following can be used:

- Ask for operator intervention

  This is the simplest way which is to inform the operator of a deadlock occurrence and let the operator handle it manually. .

- Termination of Process

  Automatic recovery from deadlock can be done by terminating one or more processes to reclaim the resources held by them.

- Rollback of processes

  In this method processes are check pointed periodically so when a deadlock occurs the process is rolled back to a point where the resource was not allocated to the process causing deadlock.

UNIT IV

Task Assignment Approach, Load Balancing Approach, Load Sharing Approach, Process Migration and Threads.

Explain the Desirable Features of a Good Global Scheduling Algorithm?

No A Priori knowledge about the Processes

- A good process scheduling algorithm should operate with absolutely no a priori knowledge about the processes to be executed.

Dynamic in Nature

- Process assignment decisions should be based on the current load of the system and not on some fixed static policy.

Quick Decision making capability

- A good process scheduling algorithm must make quick decisions about the assignment of processes to processors.

Balance System Performance and Scheduling Overhead

- Several global scheduling algorithms collect global state information and use this information in making process assignment decisions.
- In a distributed environment, however, information regarding the state of the system is typically gathered at a higher cost than in a centralized system.

Stability

- A scheduling algorithm is said to be unstable if it can enter a state in which all the nodes of the system are spending all of their time migrating processes without accomplishing any useful work in an attempt to properly schedule the processes for better performance.
- This form of fruitless migration of processes is known as processor thrashing.

Scalable

- Algorithm should be scalable and able to handle workload inquiry from any number of machines in the network.

Fault Tolerance.

- A good scheduling algorithm should not be disabled by the crash of one or more nodes of the system.
- At any instance of time, it should continue functioning for nodes that are up at that time.

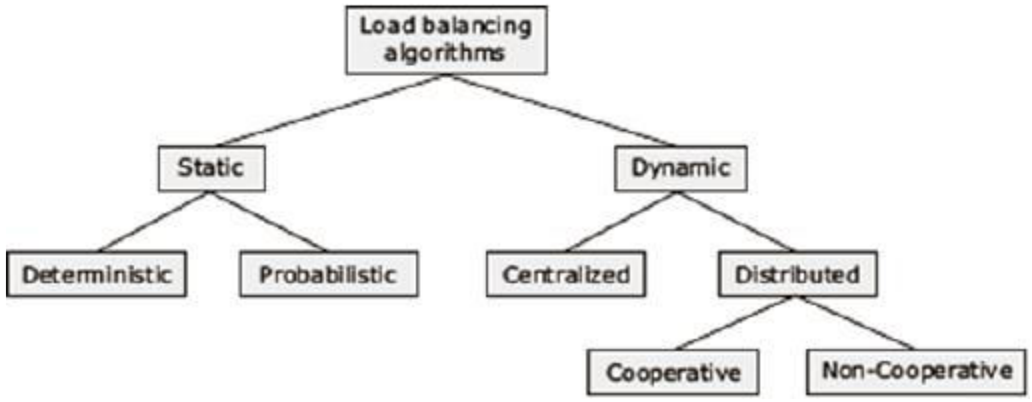
Fairness of Service

- In any load balancing scheme, heavily loaded nodes will obtain all the benefits while tightly loaded nodes will suffer poor response time A fair strategy that improves response time of heavily loaded nodes without unduly affecting response time of poorly loaded node.

Explain all about Load balancing?

Load balancing is a technique in which workload is distributed across multiple computers or other resources to get optimal resource utilization, minimum time delay, maximize throughput and avoid overload.

Load balancing approaches



Static vs. Dynamic:

Static:

It use only information about the average behavior of the system, ignore the current state of the system.

Dynamic:

It responds to the current system state that changes dynamically.

Deterministic vs. heuristic:

- Deterministic algorithms are suitable when the process behavior is known in advance.
- If all the details like list of processes, computing requirements, file requirements, and communication requirements are known prior to execution, then it is possible to make a perfect assignment.

- In the case load is unpredictable or variable from minute to minute or hour to hour, a Heuristic processor allocation is preferred.

Centralized vs. Distributed:

- Scheduling decision is made at one single node called centralized server node.
- This approach can efficiently make process assignment decision, because the centralized server knows both the load at each node and the number of processes needing service.
- The other nodes periodically send status update message to the central server node.
- In dynamic distributed scheduling algorithms, task of processor assignment is physically distributed among various nodes.
- Dynamic distributed scheduling are more effective than centralized server.
- Centralized algorithms are liable to a single point of failure,

Cooperative Vs Non cooperative:

- No cooperative:

  In this algorithm, individual entities act as autonomous entities and make scheduling decisions independently.
- Cooperative:

  In this algorithm, the distributed entities cooperate with each other to make scheduling decisions. It is more complex and its stability is better than no cooperative algorithm.

Explain Issues in Designing Load Sharing Approach Algorithms

Load Estimation Policies

- Simple load estimation policy of counting the total number of processes on a node is not suitable for use is modern distributed systems.
- Therefore, measuring CPU utilization is used as a method of load estimation in these systems.

Process Transfer Policies

- Load sharing approach uses all or nothing strategy.
- This strategy uses the single-threshold policy with the threshold value of all the nodes fixed at 1.

- That is, a node becomes a candidate for accepting a remote process only when it has no process, and a node becomes a candidate for transferring a process as soon as it has more than one process.
- To improve this strategy pre-emptive transfer is made to nodes that are not idle but are expected to soon become idle.
- Some load sharing algorithms use a threshold value 2 instead of 1.

Location Policies

- In load-sharing algorithms, the location policy decides the sender node or the receiver node of a process that is to be moved within the system for load sharing. Following are the location policies:

1. Sender-initiated policy:
   - The sender node of the process decides where to send the process.
   - In the sender-initiated location policy, heavily loaded nodes search for lightly, loaded nodes to which work may be transferred.
   - A node is a viable candidate for receiving a process from the sender node only if the transfer of the process to that node would not increase the receiver node's load above its threshold value.
2. Receiver-initiated policy,
   - The receiver node of the process decides from where to get the process.
   - In the receiver-initiated location policy, lightly loaded nodes search for heavily loaded nodes from which work may be transferred.
   - A node is viable candidate for sending one of its processes only if the transfer of the process from that node would not reduce its load below the threshold value.

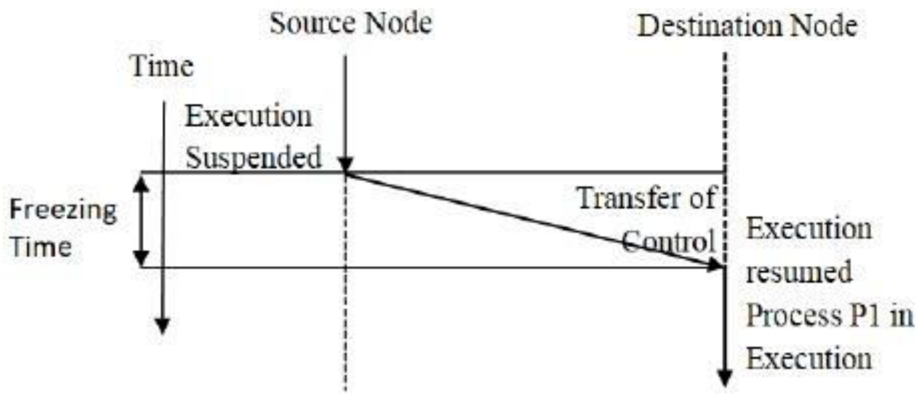
State Information Exchange policies

- A node needs to know the state of other nodes only when it is either under loaded or overloaded
- Therefore, in load-sharing algorithms, a node normally exchange state information with other nodes only when its state changes.
- The two commonly used policies for this purpose are described below.

1. Broadcast When State Changes.
   - In this method, a node broadcasts a State Information Request message when it becomes either underloaded or overloaded.
   - In the sender-initiated policy, a node broadcasts this message only when it becomes overloaded and in the receiver-initiated policy, this message is broadcast by a node when it becomes underloaded.
2. Poll When State Changes.
   - In this method, when a node's state changes, it does not exchange state information with all other nodes but randomly polls the other nodes one by one and exchanges state information with the polled nodes.
   - The state exchange process stops either when a suitable node for sharing load with the probing node has been found or the number of probes has reached the probe limit.
   - In sender-initiated policy, poling is done by a node when it becomes overloaded, and in receiver-initiated policy, polling is done by a node when it becomes underloaded.

What is Process Migration and explain its advantages

- Process migration is the relocation of a process from its current location (the source node) to another node (the destination node).



- A process may be migrated either before it starts executing on its source node (non pre-emptive migration) or during the course of its execution (pre-emptive migration).
- Process migration involves the following major steps :
   1. Selection of a process that should be migrated.

2. Selection of the destination node to which the selected process should be migrated.
3. Actual transfer of the selected process to the destination node.

Advantages of Process Migration

Reducing average response time of processors

- Process migration facility is be used to reduce the average response time of the processes of a heavily loaded node by some processes on idle or underutilized nodes.

Speeding up individual jobs

- Process migration facility may be used to speed up individual jobs in two ways.
- The first method is to migrate the tasks of a job to the different nodes of the system and to execute them concurrently.
- The second approach is to migrate a job to a node having a faster CPU or to a node at which it has minimum turnaround time

Gaining higher throughout

- In a system with process migration facility, the capabilities of the CPUs of all the nodes can be better utilized by using a suitable load-balancing policy.
- This helps in improving the throughout of the system.

Utilizing resources effectively

- Process migration policy also helps in utilizing resources effectively , since any distributed system consisting of different resources such as CPU, printers, storage , etc. and software with different capabilities are optimally used.
- Depending nature of process, it can be appropriately migrated to utilize the system resource efficiently.

Improving system reliability.

- Simply migrate a critical process to a node whose reliability is higher than other nodes in the system.

Improving system security

- A sensitive process may be migrated and run on a secure node that is not directly accessible to general users, thus improving the security of that process.

Desirable Features of Good Process Migration Mechanism

Transparency

- Transparency is an important requirement for a system that supports process migration.

Object access level.

- Object access level transparency is the minimum requirement for a system to support non pre-emptive process migration
- Access to objects such as files and devices can be done in a location independent manner.
- The object access level transparency allows, free initiation of programs at an arbitrary node.

System call and interprocess communication

- System call and inter-process communication level transparency is must for a system to support pre-emptive process migration facility.
- Transparency of inter-process communication is also desired for the transparent redirection of messages during the transient state of process that recently migrated.

Minimal Interface

- Migration of a process should cause minimal interference to the progress of the process involved and to the system as a whole.
- This can be achieved by minimizing the freezing time of the process being migrated.

Minimal Residual Dependencies

- A migrated process should not in any way continue to depend on its previous node once it has started executing on its new node

Efficiency

- It is the major issue in implementing process migration.
- Main sources of inefficiency are: time required to migrate a process, the cost of locating an object and the code of supporting remote execution once the process is migrated.

Robustness

- The failure of a node other than the one on which a process is currently running should not in any way affect the accessibility or execution of that process.
- Communication between coprocessors of a job

- Benefit of process migration is the parallel processing among the processes of a single job distributed over several nodes.
- Coprocessors are able to directly communicate with each other irrespective of their locations.

Explain Process Migration Mechanism

1. Mechanisms for Freezing and Restarting a Process

- In preemptive process migration, at some point during migration, the process is frozen on its source node its state information is transferred to its destination node.
- The process is restarted on its destination node using this state information.

Immediate and Delayed Blocking of the Process :

- Depending upon the process's current state, it may be blocked immediately or the blocking may have to be delayed until the process reaches a state when it can be blocked.

Fast and Slow I/O operations

- The process is frozen after the completion of all fast I/O operations.
- It is feasible to wait for fast I/O operations to complete before freezing the process.
- It is not feasible to wait for slow I/o operations such as those on a pipe or terminal because the process must be frozen in a timely manner for the effectiveness of process migration

Information about Open Files

- A process's state information also consists of the information pertaining to files currently open by the process. This includes information such as the names or identifiers of the files, their access modes, and the current positions of their file pointes.

Reinitiating the Process on its Destination Node

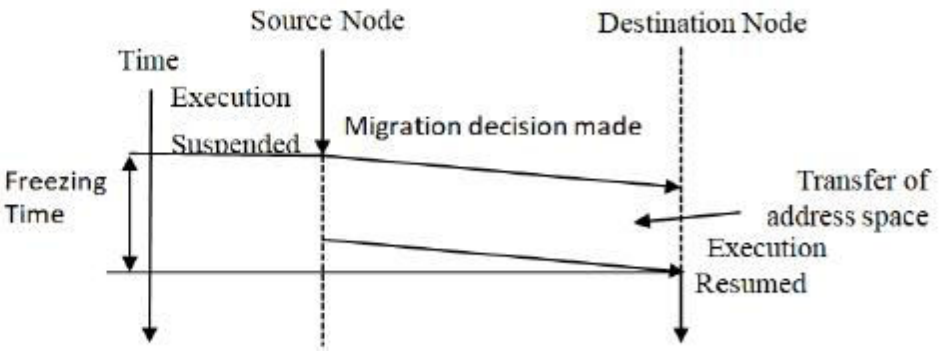
- On the destination node, an empty process state is created that is similar to that allocated during process creation.
- Depending upon the implementation, the newly allocated process may or may not have the same process identifier as the migrating process.
- Once all the state of the migrating process has been transferred from the source node to the destination node and copied into the empty process state, the new copy of the process is unfrozen and the old copy is deleted.

- The process is restarted on its destination node in whatever state it was in before being migrated.

2. Address Space Transfer Mechanism

Total Freezing

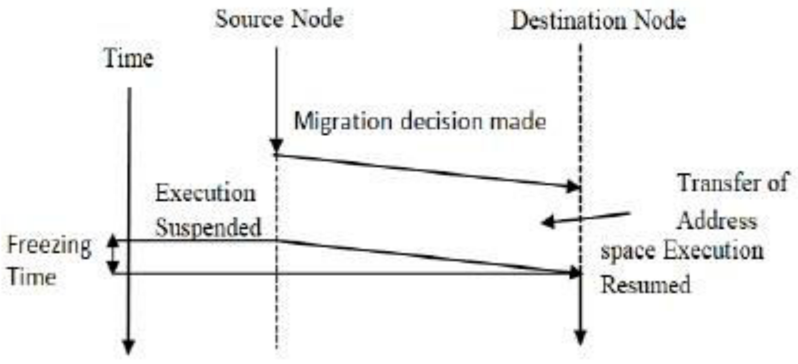
- In this method, a process's execution is stopped while its address space is being transferred as shown in figure



- Its disadvantage is that if a process is suspended for a long time during migration, timeouts may occur, and if the process is interactive, the delay will be noticed by the user.

Pre transferring

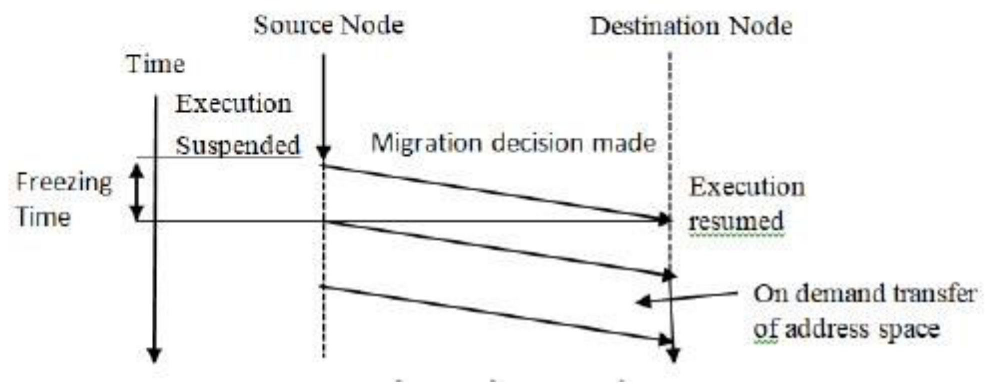
- In this method, the address space is transferred while the process is still running on the source node as shown in figure



- Address space transferred while process is running on the source node
- After decision for migration is made process continues to execute on source node until address space is has been transferred.

Transfer on Reference

- In this method, the process's address space is left behind on its source node.



- The relocated process executes on its destination node, attempts to reference memory pages results in the generation of requests to copy in the desired blocks from their remote locations.
- Failure of source node results in failure of process.

3. <u>Message Forwarding Mechanism</u>

- The message to be forwarded to the migrant process's new location can be classified into the following :

Type-1:

Messages received at the source node after the process's execution has been stopped on its source node and the process's execution has not yet been started on its destination node.

Type-2:

Messages received at the source node after the process's execution has started on its destination node.

Type-3:

Messages that are to be sent to the migrant process from any other node after it has started executing on the destination node.

<u>Mechanism of Resending the Message</u>

- Messages of types 1 and 2 are returned to the sender as not deliverable or are simply dropped
- The sender of the message is stores a copy of the data and is prepared to retransmit it.
- Sender retries after locating the new node (using locate operation)
- Type 3 message directly sent to new node.

Origin site Mechanism

- The process identifier of these systems has the process's origin site embedded in if, and each site is responsible for keeping information about the current locations of all the processes created on it.
- In these systems, messages, for a particular process are always first sent to its origin site.
- The origin site then forwards the message to the process's current location.
- There is continuous load on the migrant process's origin site even after the process has migrated from that node.

4. Mechanism for Handling Coprocesses

Disallowing Separation of Coprocesses

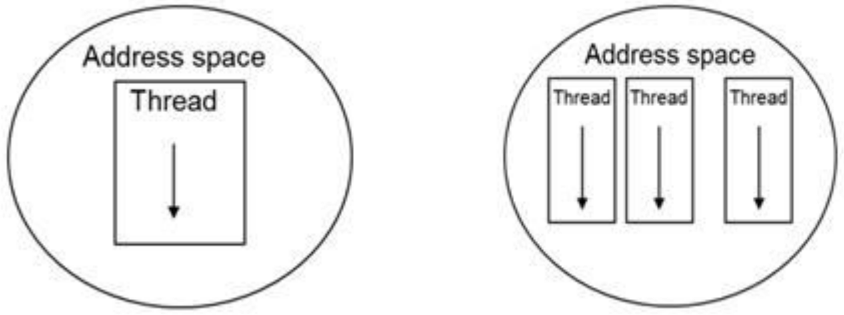
- Process will not be migrated unless their children complete execution.
- Parent and child process are migrated together.

Home Node or Origin Site Concept

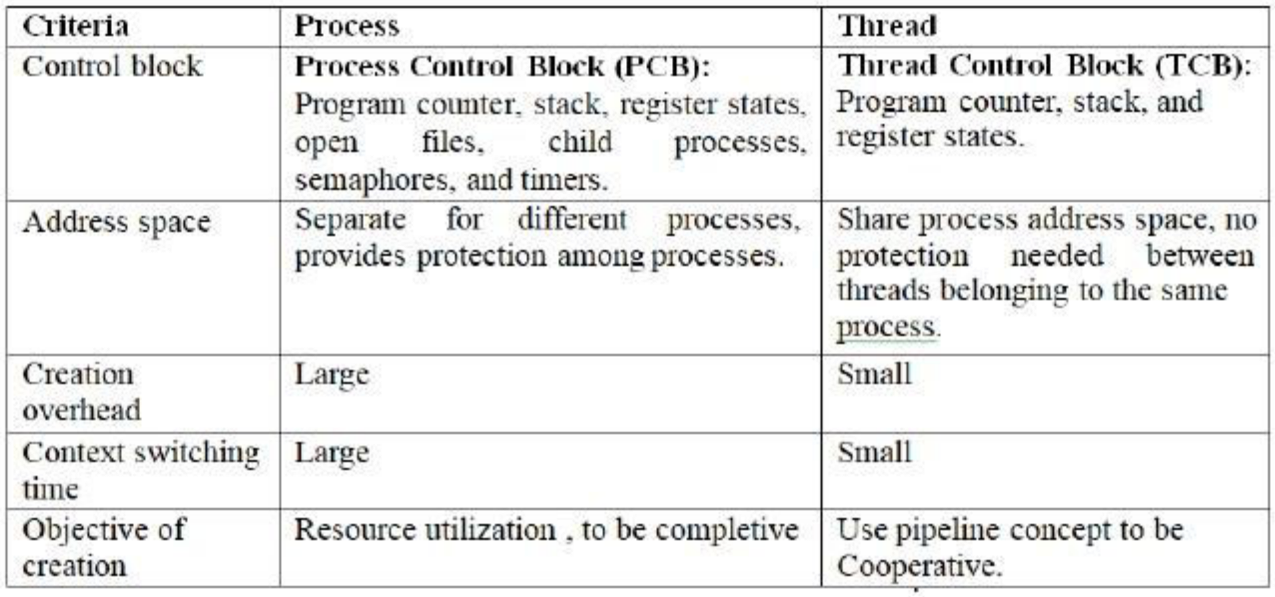
- It implies that each process has a home node but it can execute independently on different nodes of the system.
- It increases message traffic and communication cost.

Comparison of Process and Threads?

- In traditional operating systems the basic unit of CPU utilization is a process.
- Each process has its own program counter, its own register states, its own stack, and its own address space.
- In operating systems with threads facility, the basic unit of CPU utilization is a thread.
- Each thread of a process has its own program counter, its own register states, and its own stack.



- All the threads of a process share of same address space and global variables.
- All threads of a process also share the same set of operating system resources, such as open files, child processes, semaphores, signals, accounting information.
- On a uniprocessor, threads run in quasi-parallel (time sharing), whereas on a shared-memory multiprocessor, as many threads can run simultaneously as there are processors.
- At a particular instance of time, a threads can be in any one of several states: running, blocked, ready, or terminated.

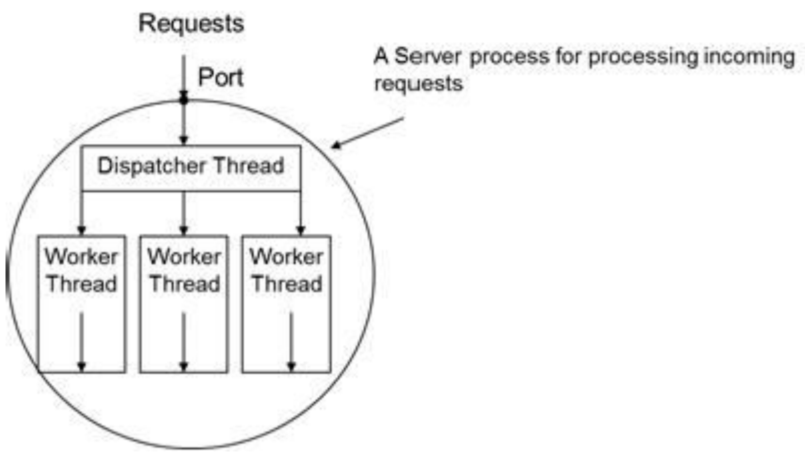


Criteria	Process	Thread
Control block	**Process Control Block (PCB):** Program counter, stack, register states, open files, child processes, semaphores, and timers.	**Thread Control Block (TCB):** Program counter, stack, and register states.
Address space	Separate for different processes, provides protection among processes.	Share process address space, no protection needed between threads belonging to the same process.
Creation overhead	Large	Small
Context switching time	Large	Small
Objective of creation	Resource utilization , to be completive	Use pipeline concept to be Cooperative.

Explain Thread Models in detail?

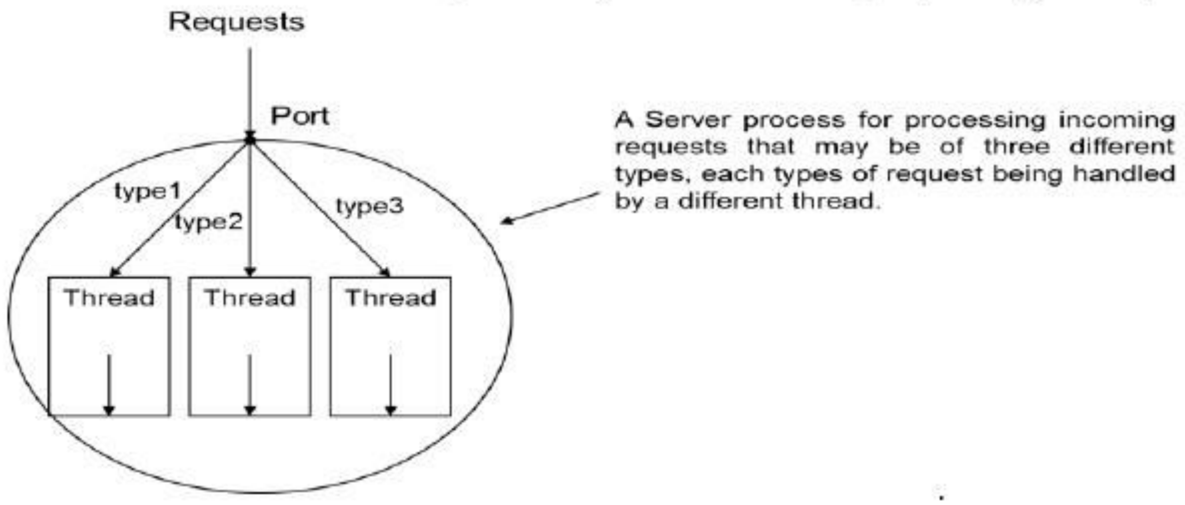
1. Dispatcher Worker Model

- In this model, the process consists of a single dispatcher thread and multiple worker threads.
- The dispatcher thread accepts requests from clients and, after examining the request, dispatches the request to one of the free worker threads for further processing of the request.
- Each worker thread works on a different client request.
- Therefore multiple client requests can be processed in parallel as shown in figure



2. Team Model

- In this model, all threads behave as equal.
- Each threads gets and processes clients' requests on its own.

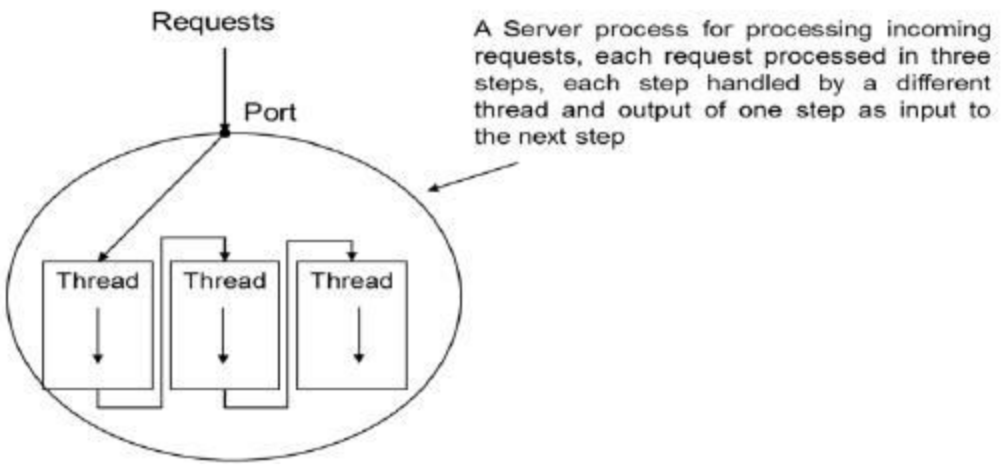

Requests
Port
A Server process for processing incoming requests that may be of three different types, each types of request being handled by a different thread.
type1
type2
type3
Thread
Thread
Thread

- This model is often used for implementing specialized threads within a process.
- Each thread of the process is specialized in servicing a specific type of request.

3. Pipeline model.

- This model is useful for applications based on the producer-consumer model.
- The output data generated by one part of the application is used as input for another part of the application.
- In this model, the threads of a process are organized as a pipeline so that the output data generated by the first thread is used for processing by the second thread, the output of the second thread is used for processing by the third thread, and so on.

- The output of the last thread in the pipeline is the final output of the process to which the threads belong.


Requests
Port
Thread
Thread
Thread
A Server process for processing incoming requests, each request processed in three steps, each step handled by a different thread and output of one step as input to the next step

Explain the Designing Issues in Thread Package

1. Thread Creation
   - Threads can be created either statically or dynamically.
   - In the static approach, the number of threads of a process remains fixed for its entire lifetime, while in the dynamic approach, the number of threads of a process keeps changing dynamically.
   - In the dynamic approach, a process is started with a single thread, new threads are created as and when needed during the execution of the process, and a thread may destroy itself when it finishes its job by making an exit call.
   - On the other hand, in the static approach, the number of threads of a process is decided either at the time of writing the corresponding program or when the program is compiled.
2. Thread Termination
   - A thread may either destroy itself when it finishes its job by making an exit call or be killed from outside by using the kill command and specifying the thread identifier as its parameter.
   - In statically created threads, the number of threads remain constant and are never killed.
3. Thread Synchronization
   - Two threads of a process need to increment the same global variable within the process.

- For this to occur safely, each thread must ensure that it has exclusive access for this variable for some period of time.
- A segment of code in which a thread may be accessing some shared variable is called a critical region.
- Two commonly used mutual exclusion techniques in a threads package are mutex variables and condition variables.

4. Thread Scheduling
   - Priority assignment facility.
     - A priority-based threads scheduling scheme may be either non-preemptive or preemptive.
     - In non-preemptive a CPU is not taken away from the assigned thread until some thread with higher priority becomes ready to run.
     - In preemptive scheme, a higher priority thread always preempts a lower priority one.

5. Signal Handling
   - Signals provide software-generated interrupts and exceptions.
   - Interrupts are externally generated disruptions of a thread or process, whereas exceptions are caused by the occurrence of unusual condition during a thread's execution.
   - The two main issues associated with handling signals in a multithreaded environment are as follows:
     - A signal must be handled properly no matter which thread of the process receives it.
     - Signals must be prevented from getting lost.

UNIT V:

File Models, File Accessing Models, File Sharing Semantics, File Caching Schemes, File Replication, Atomic Transactions, Cryptography, Authentication, Access control and Digital Signatures.

A file is a subsystem of an operating system that performs file management activities such as organization, storing, retrieval, naming, sharing and protection of files. A distributed file system supports:

- Remote Information sharing

  A distributed file system allows a file to be transparently accessed by processes of any node of the system irrespective of file's location.
- User mobility

  User should not be forced to work on one specific node but should have flexibility to work on different nodes at different times.
- Availability

  For better fault tolerance file should be available for use even in the event of temporary failure of one or more nodes of the system.
- Diskless workstation
  - A diskless workstation is more economical, less noisy and generates less heat.
  - A distributed system with its transparent remote file sharing capability allows use of diskless workstation in system.

A distributed system provides following three types of services:

- Storage service
  - Deals with allocation and management of secondary device spaces.
  - It provides a logical view of the storage system by providing operations for storing and retrieving data.
  - Also known as disk service and block service.
- True file service.
  - It is concerned with the operations on individual file such as operation for accessing and modifying the data in files and for creating and deleting files.

- Name service
    - It provides mapping between text names for files to their IDs.
    - It is also known as directory service.
    - It performs directory related activities such as creation and deletion of directories, adding new file to directory, deleting file from directory, changing name of file, moving a file from one directory to another.

Explain Feature and goal of distributed file system

1. Transparency
    - Structure Transparency
        - In multiple file servers, the variety of file servers should be transparent to the clients of a distributed file system.
        - Clients should not know the number or locations of the file servers and the storage devices
    - Naming Transparency
        - The name of the file should not give any hint regarding file location.
    - Access Transparency
        - Client process on a host has uniform mechanism to access all files in system regardless of files are on local/remote host.
    - Replication Transparency.
        - Files may be replicated to provide redundancy for availability and also to permit concurrent access for efficiency.
2. User Mobility
    - User should not be forced to work on one specific node but should have flexibility to work on different nodes at different times
3. Performance
    - Performance is average amount of time needed to satisfy client requests.
    - The extra overhead due to network communication for remote file should be hidden to users.

4. Simplicity and ease of use.
    - The user interface should be simple and number of commands should be as small as possible.
5. Scalability
    - A good distributed system should be able to cope with growth of nodes and users in the system.
    - Growth should not affect performance of overall the system
6. High availability
    - Failures of one or more components of system should not degrade performance of system.
    - Stable storage is technique used by several file system for high reliability.
7. High reliability
    - The file system should automatically generate backup copies of critical files in order to recover from sudden failure.
8. Data integrity
    - For a shared file, the file system must guarantee the integrity of data stored in it.
    - Concurrent access requests from multiple users who are competing to access the file must be properly synchronized by the use of some form of concurrency control mechanism.
9. Security
    - A distributed file system should be secure so that its user can be confident of the privacy of their data.
10. Heterogeneity
    - Heterogeneous distributed system provides the flexibility to their users to use different computer platforms for different applications.

Explain Different types of File models

Unstructured and Structured file

- Unstructured Files
    - In this model there is no substructure known to the file server and the contents of each file of the file system appears to the file server as an uninterrupted sequence of bytes.
    - UNIX and MS DOS use this file model.
    - In structured file model a file appears of the file server as an ordered sequence of records.
    - In this model a record is the smallest unit of file data that can be accessed and the file system read or writes operations are carried out on a set of records.
- Structured Files
- Structured files are two types (1) non-indexed records and (2) indexed records.

1. Non-indexed records:
    - In this model a file record is accessed by specifying its position within the file for example fifth record from the beginning of the file or the second record from the end of the file.
2. Indexed records:
    - In indexed records a file is maintained as a B-Tree or other suitable data structure or a hash table is used to store quickly.

Mutable and Immutable files.

- Mutable Files
    - In this model update performed on a file overwrites on its old content to produce the new contents.
    - Most existing operating systems use the mutable file model.
- Immutable Files
    - In this model file cannot be modified once it has been created except to be deleted.
    - Versioning approach is normally used to implement file updates and each file is represented by a history of immutable versions.

- Rather than updating the same file a new file is created each time a change is made to the file contents and the old version is retained unchanged.
- Immutable file makes easy to support consistent sharing.
- Disadvantage: increased use of disk space and increased disk allocation activity.

Explain File accessing models

1. Accessing Remote Files

- A distributed file system may use one of the following models to service client's file access request.

❖ **Remote service model**

- The client's request for file access is delivered to the server, the server machine performs the access request, and finally the result is forwarded back to the client.
- The access requests from the client and the server replies for the client are transferred across the network as messages.
- The file server interface and the communication protocols must be designed carefully to minimize the overhead of generating messages as well as the number of messages that must be exchanged in order to satisfy a particular request.

❖ **Data caching model**

- If the data needed to satisfy the client's access request is not present locally, it is copied from the server's node to the client's node and is cached there.
- The client's request is processed on the client's node itself by using the cached data.
- Thus repeated accesses to the same data can be handled locally.
- A replacement strategy LRU is used is used for cache management.

2. Unit of Data Transfer

Unit of data refers to fraction of file data that is transferred to and from clients as a result of a single read or write operations.

1. File level transfer model

- Whole file is treated as unit of data transfer between client and server in both the direction.

Advantages

- o Transmitting an entire file in response to a single request is more efficient than transmitting it page by page as the network protocol overhead is required only once.
- o It has better scalability because it requires fewer accesses to file servers, resulting in reduced server load and network traffic.
- o Disk access routines on the servers can be better optimized if it is known that requests are always for entire files rather than for random disk blocks.
- o Once an entire file is cached at a client's site, it becomes immune to server and network failures.

Disadvantage

- o This model requires sufficient storage space on the client's node for storing all the requires files in their entirely.

2. Block level transfer model

- o In this model, file data transfer across the network between a client and a server take place in units of file blocks.
- o A file block is a contiguous portion of a file and is usually fixed in length.
- o In page level transfer model block size is equal to virtual memory page size.

Advantages

- o This model does not require client nodes to have large storage.
- o It eliminates the need to copy an entire file when only a small portion of the file data is needed.
- o It provides large virtual memory for client nodes that do not have their own secondary storage devices.

Disadvantage

- o When an entire file is to be accessed, multiple server requests are needed in this model, results in more network traffic and more network protocol overhead

3. Byte level transfer model

In this model, file data transfers across the network between a client and a server take pace in units of bytes.

Advantages

- This model provides maximum provides maximum flexibility because it allows storage and retrieval of an arbitrary sequential subrange of a file, specified by an offset within a file, and a length.
- Cache management is difficult due to variable length data for different requests.

4. Record level transfer model

- In this model, file data transfers across the network between a client and a server take place in units of records.

Explain File-sharing semantics in DDS

Multiple users may access a shared file simultaneously. An important design issue for any file system is to define when modifications of file data done by a user are visible to other users. This is defined by the file-sharing semantics used by the file system.

- UNIX semantics
- Session semantic
- Immutable shared file semantics
- Transaction like semantics

1. UNIX semantics

- Absolute time ordering is enforced on operations which ensure that read operation on a file sees the effects of all previous write operations performed on that file. Write to an open file immediately become visible to users accessing the file at the same time.

2. Session semantics

- A session is a series of file accesses made between the open and close operations.
- In session semantics, all changes made to a file during a session are initially made visible only to the client process that opened the session and are invisible to other remote processes who have the same file open simultaneously.
- Once the session is closed, the changes made to the file are made visible to remote processes only in later starting sessions.

3. **Immutable shared file semantics**
   - According to this semantics, once the creator of a file declares it to be sharable, the file is treated as immutable, so that it cannot be modified any more.
   - Change to the file is handled by creating a new updated version of the file.
   - Therefore, the semantics allows files to be shared only in the read-only mode.
4. **Transaction like semantics**
   - This semantics is based on the transaction mechanism, which is a high-level mechanism for controlling concurrent access to shared mutable data.
   - A transaction is a set of operations enclosed in-between a pair of begin_transaction and end_transaction like operations.
   - The transaction mechanism ensures that the partial modifications made to the shared data by a transaction will not be visible to other concurrently executing transactions until the transaction ends

Explain File caching schemes in DDS

Cache Location

- Cache location refers to the place where the cached data is stored. Assuming that the original location of a file is on its server's disk, there are these possible cache location in a distributed file system.

1. Server's Main Memory
   - When no caching scheme is used, before a remote client can access a file, the file must first be transferred from the server's disk to the server's main memory.
   - Then across the network from the server's main memory to the client's main memory.
   - Thus the total cost involved is one disk access and one network access.

Advantages

- It is easy to implement and is totally transparent to the clients.
- It is easy to keep the original file and cached data consistent.
- Since a single server manages both the cached data and the file, multiple accesses from different clients can be easily synchronized to support UNIX-like file-sharing semantics.

Disadvantage

- It does not eliminate the network access cost and does not contribute to the scalability and reliability of the distributed file system.

2. Client's Disk

A cache located in a client's disk eliminates network access cost but requires disk access cost on a cache hit.

Advantages

- If the cached data is kept on the client's disk, the data is still there during recovery and there is no need to fetch it again from the server's node.
- As compared to a main-memory cache, a disk cache has plenty of storage space.
- Disk cache is useful to those applications that use file level transfer.

Disadvantage

- This policy does not work if the system is to support diskless workstations.
- Access time increases as every time disk access is done.

3. Client main memory

- A cache located in a client's main memory eliminates both network access cost and disk access cost.
- It also permits workstations to be diskless.

Advantages

- It provides high scalability and reliability as cache hit access request can be serviced locally without the need to contact server.

Disadvantage

- Client disk memory is small compared to client's disk size.

Modification Propagation

1. Write through Scheme

- In this scheme, when a cache entry is modified, the new value is immediately sent to the server for updating the master copy of the file.

Advantage

- High degree of reliability and suitability for UNIX-like semantics

- Since every modification is immediately propagated to the server having the master copy of the file, the risk of updated data getting lost is very low.

Disadvantage

- It has poor write performance as write access has to wait until the information is written to the master copy of the server.
- This scheme is suitable for use only in those cases in which the ratio of read-to-write accesses is fairly large.

2. Delayed Write

- In this scheme, when a cache entry is modified, the new value is written only to the cache and the client just makes a note that the cache entry has been updated.
- After some time, all updated cache entries corresponding to a file are gathered together and sent to the server at a time.
- Depending on when the modifications are sent to the file server, delayed-write policies are of different types.

a. Write on ejection from cache.

- In this method, modified data in a cache entry is sent to the server when the cache replacement policy has decided to eject it from the client's cache.
- Some data can reside in the client's cache for a long time before they are sent to the server
- Such data are subject to reliability problem.

b. Periodic write.

- In this method, the cache is scanned periodically, at regular intervals, and any cached data that have been modified since the last scan are sent to the server.

c. Write on close

- In this method, the modifications made to a cached data by a client are sent to the server when the corresponding file is closed by the client.

Advantages of Delayed write

- Write accesses complete more quickly because the new value is written only in the cache of the client performing the write.
- Modified data may be deleted before it is time to send them to the server, in such cases, modifications need not be propagated at all to the serve, resulting in a major performance gain.

- Gathering of all file updated and sending them together to the server is more efficient than sending each update separately.

Cache Validation Schemes

- A client's cache entry becomes stale as soon as some other client modifies the data corresponding to the cache entry in the master copy of the file.
- Therefore, it becomes necessary to verify if the data cached at a client node is consistent with the master copy.
- If not, the cached data must be invalidated and the updated version of the data must be fetched again from the server. There are two approaches:

1. Client Initiated Approach
   - In this approach, a client contacts the server and checks whether its locally cached data is consistent with the master copy.

a. Checking before every access
   - This approach defeats the main purpose of caching because the server has to be contacted on every access.
   - But it is suitable for supporting UNIX-like semantics.

b. Periodic checking
   - In this method, a check is initiated every fixed interval of time.

c. Check on file open.
   - In this method, a client's cache entry is validated only when the client opens the corresponding file for use.
   - This method is suitable for supporting session semantics.

2. Server Initiated Approach
   - If the frequency of the validity check is high, the client-initiated cache validation approach generates a large amount of network traffic and consumes precious server CPU time.
   - In this method, a client informs the file server when opening a file, indicating whether the file is being opened for reading, writing, or both.
   - The file server keeps a record of which client has which file open and in what mode.
   - The server keeps monitoring the file usage modes being used by different clients and reacts whenever it detects a potential for inconsistency.

- When a new client makes a request to open an already open file and if the server finds that the new open mode conflicts with the already open mode.
- The server can deny the request or queue the request or disable caching and switch to the remote service mode of operation for that particular file by asking all the clients having the file open to remove that file from their caches.

Disadvantages

- It violates the traditional client-server model in which servers simply respond to service request activities by clients.
- It requires that file servers be stateful.
- Client – initiated cache validation, approach must still be used along with the server-initiated approach.
- For example, a client may open a file, cache it, and then close it after use.
- Upon opening it again for use, the cache content must be validated because there is a possibility that some other client might have subsequently opened, modified and closed the file.

Explain File replication in DDS

As compared to a cached copy, a replica is more persistent, widely known, secure, available, complete and accurate.

Advantages of Replication

Increased availability

- The system remains operational and available to the users despite failures.
- By replicating critical data on servers with independent failure modes, the probability that one copy of the data will be accessible increases.

Increased reliability

- Replication allows the existence of multiple copies of their files.
- Due to the presence of redundant information in the system, recovery from failures becomes possible.

Improved response time

- It enables data to be accessed either locally or from a node to which access time is lower than the primary copy access time.

Reduced network traffic

- If a file's replica is available with a file server that resides on a client's node, the client's access requests can be serviced locally, resulting in reduced network traffic.

Improved system throughput

- Replication also enables several clients' requests for access to the same file to be serviced in parallel by different servers, resulting in improved system throughput.

Better scalability

- By replicating the file on multiple servers, the same requests can now be serviced more efficiently by multiple servers due to workload distribution.
- This results in better scalability.

Autonomous operation.

- In a distributed system that provides file replication as a service to their clients, all files required by a client for operating during a limited time period may be replicated on the file server residing at the client's node.
- This will facilitate temporary autonomous operation of client machines.

Replication Transparency

Replication of files should be designed to be transparent to the users so that multiple copies of a replicated file appear as a single logical file to its users.

Naming of Replicas

- It is the responsibility of the naming system to map a user-supplied identifier into the appropriate replica of a mutable object.

Replication control

- Replication control includes determining the number and locations of replicas of a replicated file.
- Depending on whether replication control is user transparent or not, the replication process is of two types :

1. Explicit replication.
    - In this type, users are given the flexibility to control the entire replication process.

- That is, when a process creates a file, it specifies the server on which the file should be placed.
- Then, if desired, additional copies of the file can be created on other servers on explicit request by the users.

2. Implicit/lazy replication.
    - In this type, the entire replication process is automatically controlled by the system without users' knowledge.
    - That is, when a process creates a file, it does not provide any information about its location.

Explain Atomic Transaction in Distributed System.

A sequence of operations that perform a single logical function, usually used in context of databases. A transaction that happens completely or not at all i.e. No partial results. The Fundamental principles are ACID

- **A**tomicity-to outside world, transaction happens indivisibly
- **C**onsistency-transaction preserves system invariants
- **I**solated-transactions do not interfere with each other
- **D**urable -Once a transaction "commits," the changes are permanent.

Programming in a Transaction System

Begin transaction:

Mark the start of a transaction.

End transaction:

Mark the end of a transaction and try to "commit".

Abort transaction:

Terminate the transaction and restore old values.

Read:

Read data from a file, table, etc., on behalf of the transaction.

Write:

Write data to file, table, etc., on behalf of the transaction

Nested Transactions:

- One or more transactions inside another transaction
- May individually commit, but may need to be undone

Example:

- Planning a trip involving three flights
- Reservation for each flight "commits" individually
- Must be undone if entire trip cannot commit

Tools for Implementing Atomic Transactions (Single System)

Stable storage:

Write to disk "atomically" (ppt, html).

Log File

Record actions in a log before "committing" them (ppt, html), for Log In Stable Storage

Locking Protocols

Serialize Read and Write operations of same data by separate Transactions.

Begin Transaction

Place a begin entry in log

Write

Write updated data to log

Abort Transaction

Place abort entry in log

End transaction (i.e., commit)

Place commit entry in log

Copy logged data to files

Place done entry in log

Crash Recovery –Search Log

If begin entry, look for matching entries

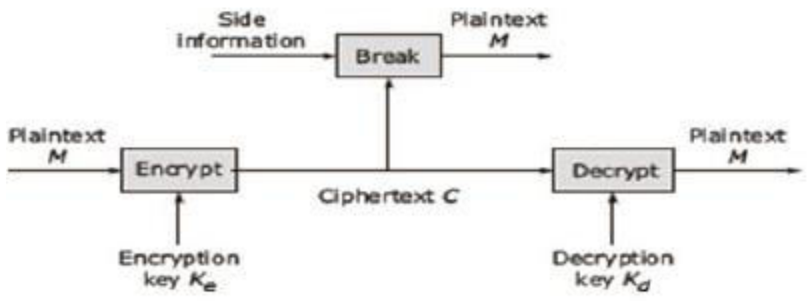
If done, do nothing (all files have been updated)

If abort, undo any permanent changes that transaction may have made

If commit but not done, copy updated blocks from log to files, then add done entry

Explain Cryptography in Distributed Systems.

- Cryptography is defined as a means of protecting private information against unauthorized access in cases where physical security is difficult to achieve.
- Cryptography is carried out using two basic operations: encryption and decryption.
- Encryption is the process of transforming intelligible information (plaintext) into unintelligible form (cipher text).
- Decryption is the reverse process, i.e. the process of transforming the information from cipher text to plaintext.
- The plaintext message is encrypted into cipher text, transmitted, and then reconverted, as shown in Fig



- The encryption algorithm has the following form:

  C = E(P,Ke)

  where P = plaintext to be encrypted Ke= encryption key C = resulting cipher text
- The decryption algorithm is performed by the same matching function which has the following form:

  P =D(C,Ke)

  where D = cipher text to be decrypted Kd = decryption key P = resulting plaintext
- There are two broad classes of cryptosystems based on whether the encryption and decryption keys are the same, viz. symmetric and asymmetric systems.

<u>Symmetric cryptography</u>

With symmetric cryptography, the same key is used for both encryption and decryption. A sender and a recipient must already have a shared key that is known to both. Key distribution is a tricky problem and was the impetus for developing asymmetric cryptography.

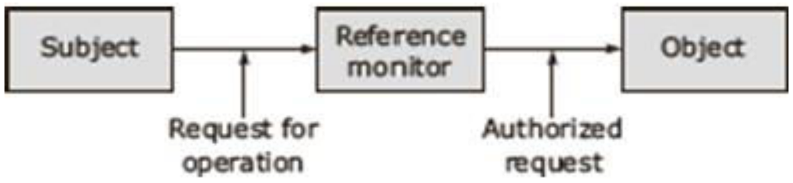
<u>Asymmetric crypto</u>

With asymmetric crypto, two different keys are used for encryption and decryption. Every user in an asymmetric cryptosystem has both a public key and a private key. The private key is kept secret at all times, but the public key may be freely distributed.

Explain Access control in detail

- When a secure channel is set up between a client and a server, the client can issue requests to the server.
- A request can be carried out only if the client has sufficient access rights for that invocation.

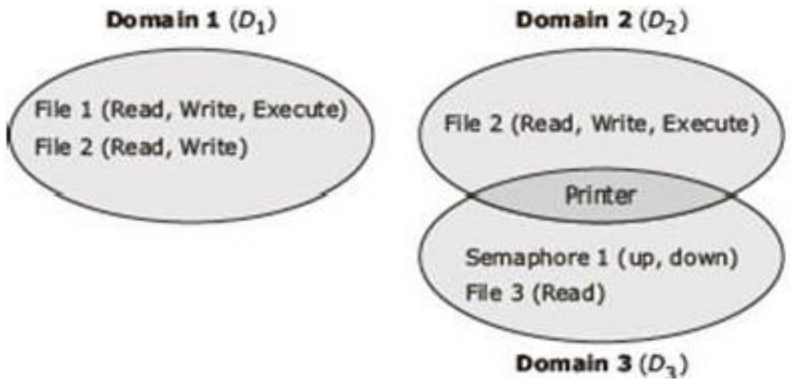
- Verifying access rights is called access control, while authorization is the process of granting access rights.



- The access control comprises subjects that issue a request to access an object.
- The **object** may be an abstract entity like a process, file, database, tree data structure, network site, etc.
- The entities that need to access and perform operations on objects and to which access authorizations are granted are called **subjects**.
- **Protection rules** define the possible ways in which subjects and objects are allowed to interact.
- Associated with a subject-object pair is an **access right** that defines the subset of possible operations for the object type that the subject can perform on that object.

Protection Domains

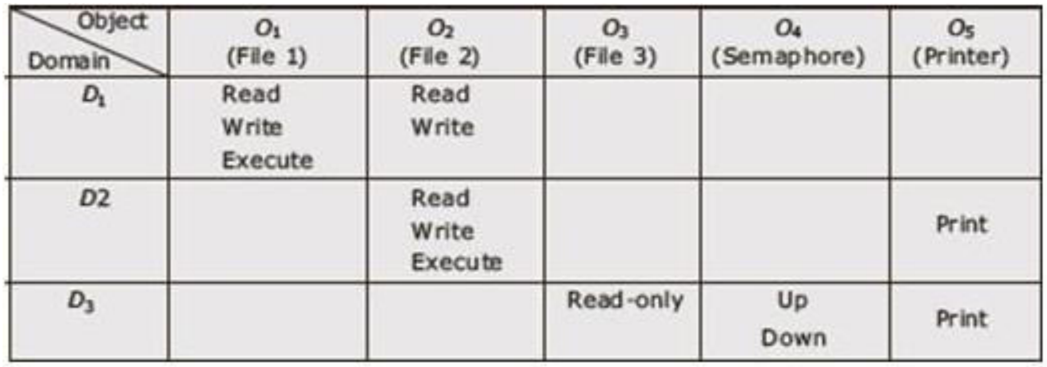
- A domain is an abstract definition of a set of access rights.
- It is defined as a set of objects-rights pair.
- Each pair specifies an object and one or more operations that can be performed on that object.
- Each of these allowed operations is called a right.



Access Matrix

- While implementing a domain-based approach, the system needs to keep track of which rights on which objects belong to a particular domain.

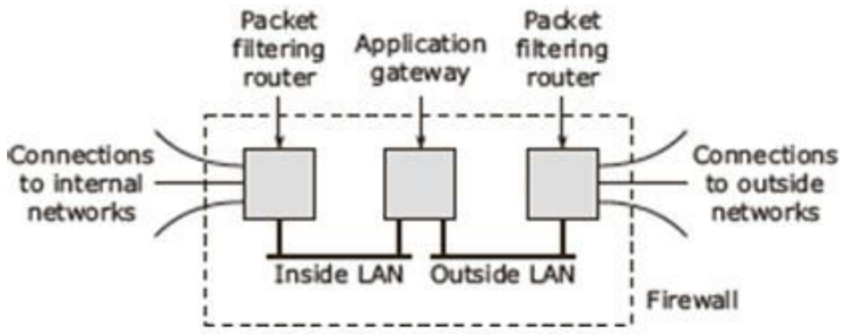
- In an access matrix model, this information is represented as an access matrix.
- Each subject in an access matrix is represented by a row and each column represents an object.
- Each entry in the matrix consists of a set of access rights.



Object / Domain	O_1 (File 1)	O_2 (File 2)	O_3 (File 3)	O_4 (Semaphore)	O_5 (Printer)
D_1	Read Write Execute	Read Write			
$D2$		Read Write Execute			Print
D_3			Read-only	Up Down	Print

Firewalls

- External access to any part of a distributed system is controlled with a special kind of reference monitor called a firewall.
- This monitor isolates the distributed system from the external world.
- Thus, a firewall is a set of related programs located at a network gateway server that protects the resources of a private network from the users of other networks.
- Basically, a firewall works closely with a router program and examines each network packet to determine whether or not to forward it towards its destination.
- A firewall is often installed in a specially designated computer separate from the rest of the network so that no incoming request can get directly at private network resources.



Secure Mobile Code

- An important requirement of distributed systems is their capability to migrate codes between hosts instead of just passive data. Mobile codes lead to various security threats.

- While sending an agent across the network, there is a need to protect the code from malicious hosts which may try to steal or modify the information carried by the agent.
- The hosts also need to be protected against malicious agents.

Protecting an agent

- When the agent moves to a host, the host should not steal the information nor should the agent modify the host's data.
- It is require the protection of an agent against attacks from the host include destroying an agent, tampering with the agent, etc.

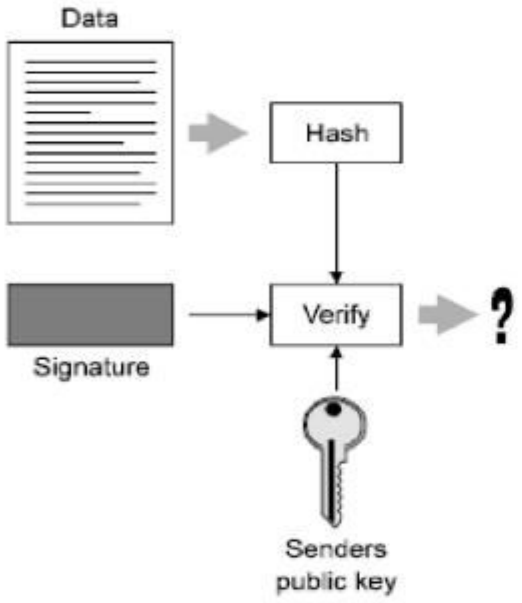
Protecting the target

- When we allow a mobile code to enter a host, it is equally important to protect the host from a malicious code.
- A user has to allow an agent into the system and then the host should be able to exercise full control over what the agent can do. There are two suggested approaches.
  - ❖ Sandbox
  - ❖ Playground.
- A sandbox is a technique by which the execution of the downloaded code can be fully controlled. Any attempt by the foreign code to execute a forbidden instruction halts the execution.
- A playground is a separate, designated machine exclusively reserved for running mobile codes.

Explain Digital Signature

- Digital signatures are used to authenticate the identity of the sender. It is like signing a message in electronic form.
- A digital signature is a protocol that produces the same effect as a real signature.
- It is a mark that only the sender can make and other people can easily recognize that it belongs to the sender. A digital signature is also used to confirm agreement to a message.
- A digital signature must be unforgettable and authentic.

- In a digital signature process, the sender uses a signing algorithm to sign the message. The message and the signature are sent to the receiver. The receiver receives the message and the signature and applies the verifying algorithm to the combination.
- If the result is true, the message is accepted otherwise it is rejected.



- A valid digital signature gives a recipient reason to believe that the message was created by a known sender (authentication), that the sender cannot deny having sent the message (non-repudiation), and that the message was not altered in transit (integrity).
- Digital signatures have assumed great significance in the modern world of web-commerce. Many countries have made provisions for recognizing digital signature as a valid authorization mechanism like paper-based signatures.