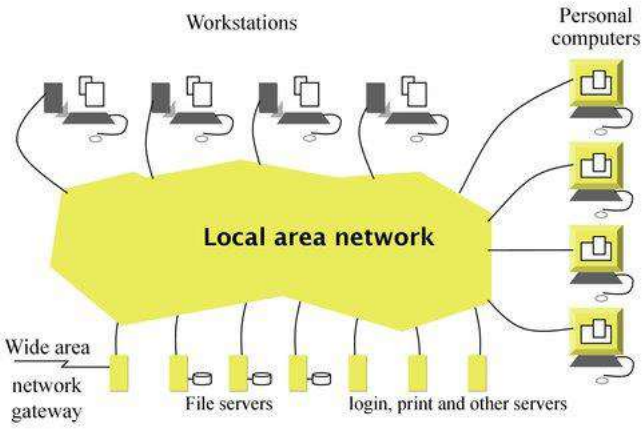
1) Define Distributed Operating System and Explain Goals of Distributed System.

- A distributed system is a collection of independent computers that appear to the users of the system as a single computer.
- Using high performance computers connected by equally high speed communication links, it is possible to build a single system consisting of multiple computer and using it as a single consolidated system.
- 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.
- If user thinks that the entire interlinked system is a single and unified computer, there has to be software envelope that joins multiple operating system together and offers a seamless interface to user.

A Distributed System



The following are the main goals of distributed systems:

- The relative simplicity of the software - each processor has a dedicated function.
- Incremental growth - if we need 10 percent more computing power, we just add 10 percent more processors.
- Reliability and availability - a few parts of the system can be down without disturbing people using the other parts.
- Openness: Offer services according to standard rules that describe the syntax and semantics of those services.

2) Explain Advantage of Distributed system over centralized system.

Economics

- A quarter century ago, according to Grosch's law: the computing power of a CPU is proportional to the square of its price.
- By paying twice as much you could get four times the performance.
- This observation fit the mainframe technology of its time.
- With microprocessor technology, Grosch's law no longer holds.
- For a few hundred dollars you 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.
- For example, with current technology it is possible to build a system from 10,000 modern CPU chips, each of which runs at 50 MIPS (Millions of Instructions Per Second), for a total performance of 500,000 MIPS.
- For a single processor (i.e., CPU) to achieve this, it would have to execute an instruction in 0.002 nsec (2 picosecond).

Inherent Distribution

- Many institutions have applications which are more suitable for distributed computation.
- Assume that there is large company buying and selling goods from different countries.
- Its offices situated in those countries are geographically diverse.
- If a company wishes unified computing system, it should implement a distributed computing system.
- Consider the global employee database of such 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.
- Ideally, if 5 percent of the machines are down at any moment, the system should be able to continue to work with a 5 percent loss in performance.
- 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

- A company will buy a mainframe with the intention of doing all its work on it.
- If the company expands and the workload grows, at a certain point the mainframe will no longer be adequate.
- The only solutions are either to replace the mainframe with a larger one (if it exists) or to add a second mainframe.
- Both of these can cause damage on the company's operations.
- In contrast, with a distributed system, it may be possible simply to add more processors to the system, thus allowing it to expand gradually as the need arises.

3) Explain Advantage of Distributed system over Independent PCs.

Data sharing

- Many users need to share data.
- For example, airline reservation clerks need access to the master data base of flights and existing reservations.
- Giving each clerk his own private copy of the entire data base would not work, since nobody would know which seats the other clerks had already sold.
- Shared data are absolutely essential for this 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.

4) Classify the Network operating system and Distributed operating system.

Network Operating system	Distributed Operating system
A network operating system is made up of software and associated protocols that allow a set of computer network to be used together.	A distributed operating system is an ordinary centralized operating system but runs on multiple independent CPUs.
Environment users are aware of multiplicity of machines.	Environment users are not aware of multiplicity of machines.
Control over file placement is done manually by the user.	It can be done automatically by the system itself.
Performance is badly affected if certain part of the hardware starts malfunctioning.	It is more reliable or fault tolerant i.e. distributed operating system performs even if certain part of the hardware starts malfunctioning.

Remote resources are accessed by either logging into the desired remote machine or transferring data from the remote machine to user's own machines.	Users access remote resources in the same manner as they access local resources.

5) Classify Distributed operating system.

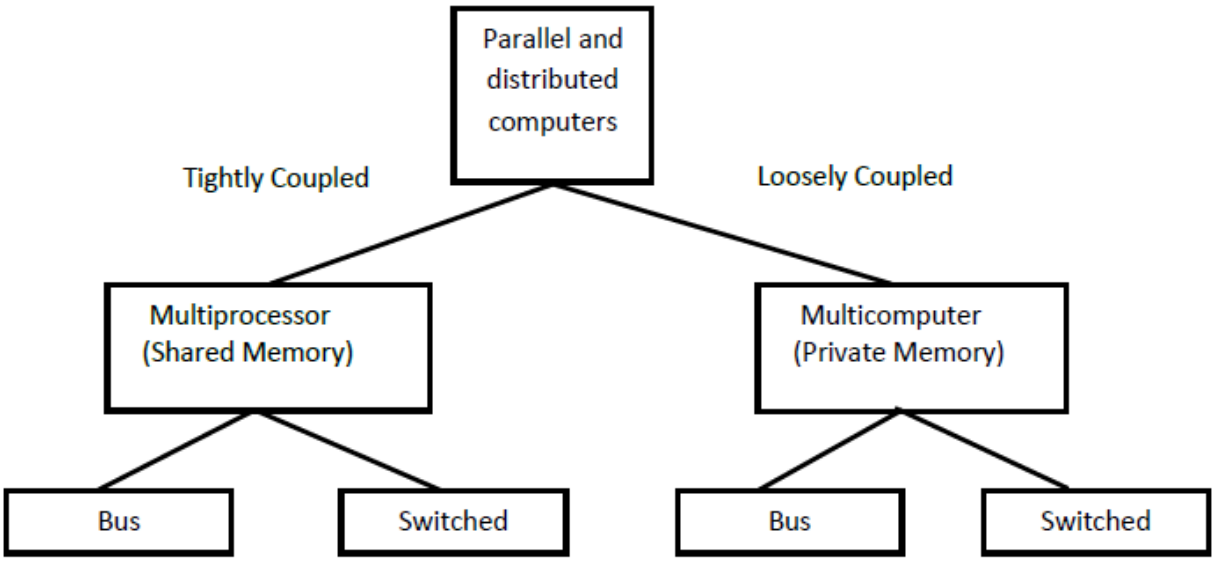


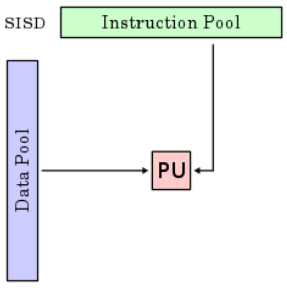
Figure: Classification of distributed system

Hardware Concepts

- Even though all distributed systems consist of multiple CPUs, there are several different ways the hardware can be organized, especially in terms of how they are interconnected and how they communicate.
- Various classification schemes for multiple CPU computer systems have been proposed over the years.
- According to Flynn, classification can be done based on, the number of instruction streams and number of data streams.

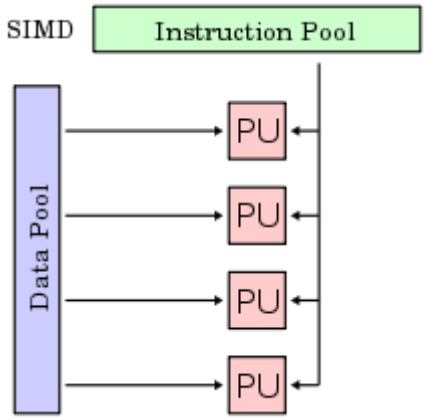
SISD

- A computer with a single instruction stream and a single data stream is called SISD.
- All traditional uniprocessor computers (i.e., those having only one CPU) fall in this category, from personal computers to large mainframes.



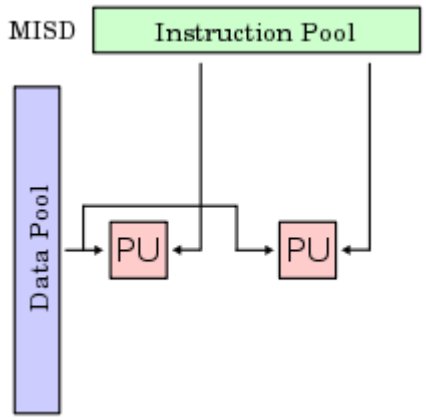
SIMD

- The next category is SIMD, single instruction stream, multiple data stream.
- This type refers to array processors with one instruction unit that fetches an instruction, and then commands many data units to carry it out in parallel, each with its own data.
- These machines are useful for computations that repeat the same calculation on many sets of data, for example, adding up all the elements of 64 independent vectors.
- Some supercomputers are SIMD.



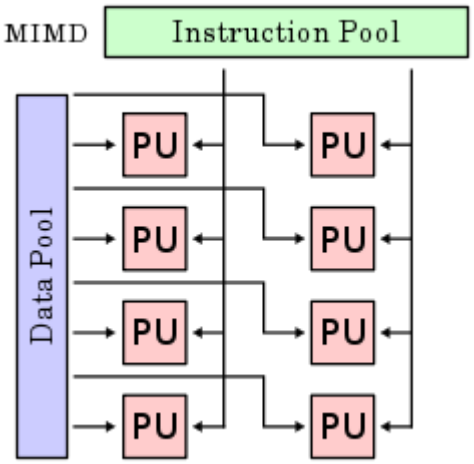
MISD

- The next category is MISD, multiple instruction stream, single data stream, no known computers fit this model.



MIMD

- Finally, comes MIMD, which essentially means a group of independent computers, each with its own program counter, program, and data.
- All distributed systems are MIMD.
- We divide all MIMD computers into two groups: those that have shared memory, usually called multiprocessors, and those that do not, sometimes called multicomputer.



Software Concepts

- The image that a system presents to its users, and how they think about the system, is largely determined by the operating system software, not the hardware.
- There are basic two types of operating system namely (1) Tightly coupled operating system and (2) loosely couple operating system; for multiprocessor and multicomputer.
- Loosely-coupled software allows machines and users of a distributed system to be fundamentally independent of one another.
- Consider a group of personal computers, each of which has its own CPU, its own memory, its own hard disk, and its own operating system, but which share some resources, such as laser printers and data bases, over a LAN.
- This system is loosely coupled, since the individual machines are clearly distinguishable, each with its own job to do.
- If the network should go down for some reason, the individual machines can still continue to run to a considerable degree, although some functionality may be lost.
- For tightly coupled system consider a multiprocessor dedicated to running a single chess program in parallel.
- Each CPU is assigned a board to evaluate, and it spends its time examining that board and all the boards that can be generated from it.
- When the evaluation is finished, the CPU reports back the results and is given a new board to work on.
- The software for this system, both the application program and the operating system required to support it, is clearly much more tightly coupled than in our previous example.

Tightly Coupled System

- Distributed Operating system is a tightly coupled software on loosely coupled hardware.
- The goal of such a system is to create the illusion in the minds of the users that the entire network of computers is a single timesharing system, rather than a collection of distinct machines.

Characteristics:

- There is a single, global inter process communication mechanism so that any process can talk to any other process.

- There is a global protection scheme.
- Process management is also uniform everywhere.
- How processes are created, destroyed, started, and stopped does not vary from machine to machine.
- The file system looks same from everywhere.
- Every file is visible at every location, subject to protection and security constraints.
- Each kernel has considerable control over its own local resources.
- For example, if swapping or paging is used, the kernel on each CPU is the logical place to determine what to do swap or page.

6) Explain Distributed Computing System Models.

- Distributed Computing system models can be broadly classified into five categories.

Minicomputer Model

- It consists of a few minicomputers interconnected by a communication network.
- Each minicomputer usually has multiple users logged on to it simultaneously.
- For this several interactive terminals are connected to each minicomputer.
- Each user is logged onto one minicomputer, with remote access to other minicomputers.
- The network allows the user to access remote resources that are available on some machine other than the user is currently logged.
- Minicomputer model can be used for resource sharing like information databases.
- ARPAnet is the example Distributed computing system based on the minicomputer model.

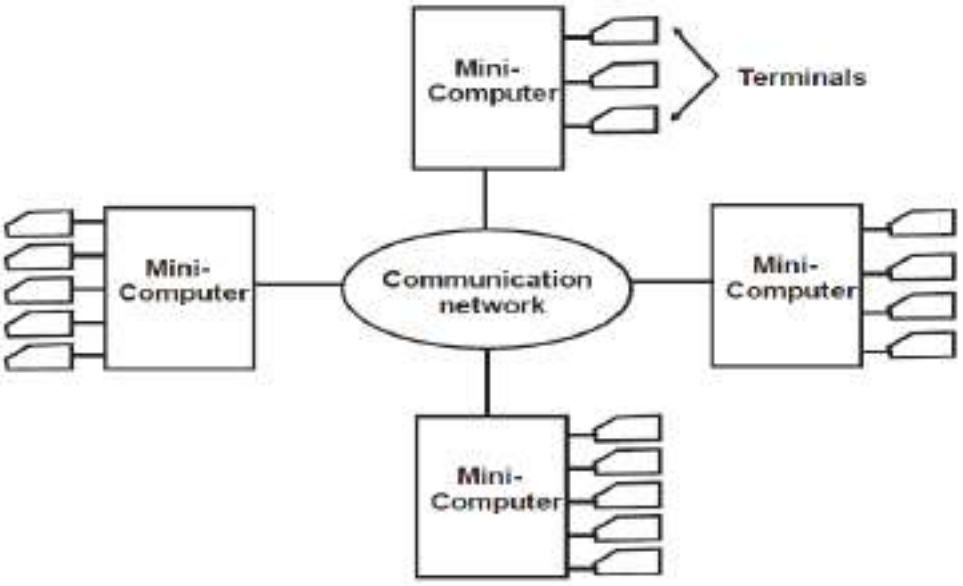


Figure: Minicomputer model

Workstation Model

- A workstation is a high-end microcomputer designed for technical or scientific applications.
- Each workstation is equipped with its own disk and is intended primarily to be used by one person at a time (single-user computer), but they are commonly connected to a local area network and run multi-user operating systems.
- Large number of workstations can lie idle resulting in a waste of CPU time.

- Using a high speed LAN allows idle workstations to process jobs of users who are logged onto other workstations, which have less computing powers.

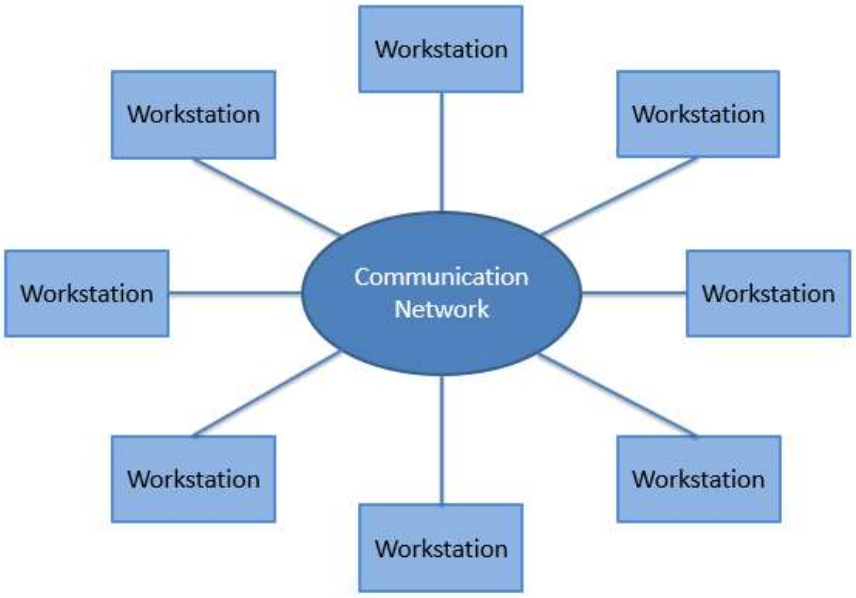


Figure: Workstation Model

Workstation-server Model

- It is a network of personal workstations each with its own disk and local file system.
- Workstation with its own local disk is usually called a disk full workstation and a workstation without a local disk is called a diskless workstation.
- When diskless workstations are used, file system to be used must be implemented by a disk full workstation or by a minicomputer equipped with a disk for file storage.
- Other minicomputers are used for providing other services like database, print etc.

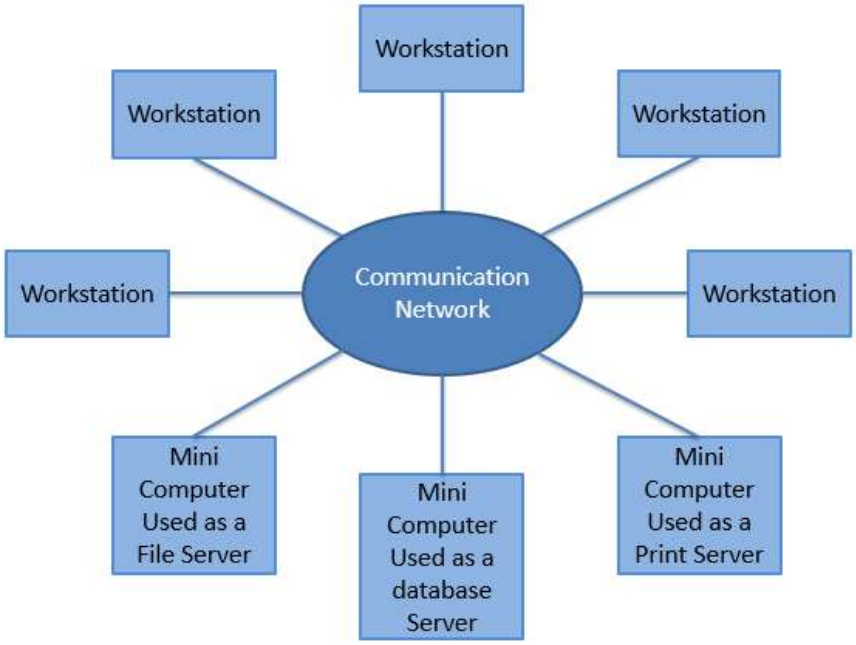


Figure: Workstation-Server Model

Processor-Pool Model

- When a user needs a very large amount of computing power for a short time, instead of allocating a processor to each user, processors are pooled together to be shared by the users as needed.
- Pool of processors contain a large number of microcomputers and mini-computers.
- Each processor in the pool has its own memory to load and run a system program or an application program of the distributed computing system.
- Amoeba, Cambridge Distributed Computing system are examples of process-pool model based Distributed systems.

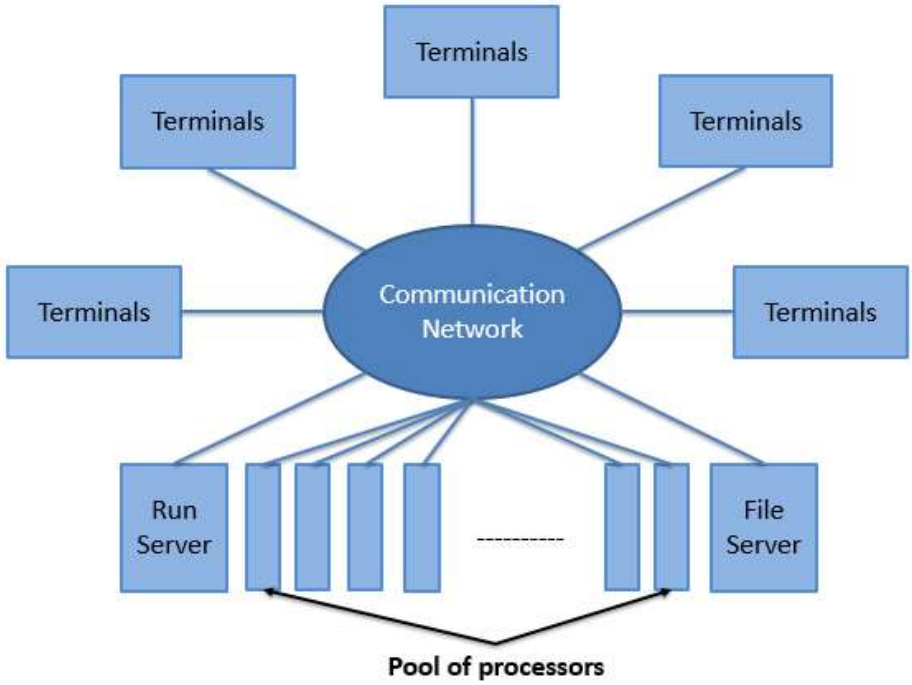


Figure: Process-pool model

Hybrid Model

- Advantages of the workstation-server and processor-pool models are combined to build a hybrid model.
- It is built on the workstation-server model with a pool of processors.
- Processors in the pool can be allocated dynamically for large computations, that cannot be handled by the workstations, and require several computers running concurrently for efficient execution.
- This model is more expensive to implement than the hybrid or the processor-pool model.

7) Explain Design Issues in Distributed Operating System.

The issues in designing a distributed system include:

Transparency

- Main goal of Distributed system is to make the existence of multiple computers invisible (transparent) and provide single system image to user.

- A transparency is some aspect of the distributed system that is hidden from the user (programmer, system developer, application).
- A transparency is provided by including some set of mechanisms in the distributed system at a layer below the interface such that a communication system appears to its users as a virtual uniprocessor.
- While users hit search in google.com……. They never notice that their query goes through a complex process before google shows them a result.

Access Transparency:

- It hides differences in data representation and how a resource is accessed by a user.
- Example, a distributed system may have a computer system that runs different operating systems, each having their own file naming conventions.
- Differences in naming conventions as well as how files can be manipulated should be hidden from the users and applications.

Location Transparency:

- Hides where exactly the resource is located physically.
- Example: by assigning logical names to resources like yahoo.com, one cannot get an idea of the location of the web page's main server.

Migration Transparency:

- Distributed system in which resources can be moved without affecting how the resource can be accessed are said to provide migration transparency.
- It hides that the resource may move from one location to another.

Relocation Transparency:

- This transparency deals with the fact that resources can be relocated while it is being accessed without the user who is using the application to know anything.
- Example: using a Wi-Fi system on laptops while moving from place to place without getting disconnected.

Replication Transparency:

- Hides the fact that multiple copies of a resource could exist simultaneously.
- To hide replication, it is essential that the replicas have the same name.
- Consequently, as system that supports replication should also support location transparency.

Concurrency Transparency:

- It hides the fact that the resource may be shared by several competitive users.
- Example: two independent users may each have stored their file on the same server and may be accessing the same table in a shared database.
- In such cases, it is important that each user should not notice that the others are making use of the same resource.

Failure Transparency:

- Hides failure and recovery of the resources.
- Example: a user cannot distinguish between a very slow or dead resource.

- Same error message come when a server is down or when the network is overloaded of when the connection from the client side is lost.
- So here, the user is unable to understand what has to be done, either the user should wait for the network to clear up, or try again later when the server is working again.

Persistence Transparency:

- It hides if the resource is in memory or disk.
- Example: Object oriented database provides facilities for directly invoking methods on storage objects.
- First the database server copies the object states from the disk i.e. main memory performs the operation and writes the state back to the disk.
- The user does not know that the server is moving between primary and secondary memory.

Reliability

- Reliability in terms of data means that data should be available without any errors.
- Distributed system where multiple processors are available and the system become reliable.
- So on failure, a backup file is available.
- In case of replication all copies should be consistent.

Scalability

- Distributed systems must be scalable as the number of user increases.
- A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity.
- Scalability has 3 dimensions:
  - **Size:** Number of users and resources to be processed. Problem associated is overloading.
  - **Geography:** Distance between users and resources. Problem associated is communication reliability
  - **Administration:** As the size of distributed systems increases, many of the system needs to be controlled. Problem associated is administrative mess.

Flexibility

- The design of Distributed operating system should be flexible due to following reasons:
  - **Ease of Modification:** It should be easy to incorporate changes in the system in a user transparent manner or with minimum interruption caused to the users.
  - **Ease of Enhancement:** New functionality should be added from time to time to make it more powerful and easy to use.
- A group of users should be able to add or change the services as per the comfortability of their use.

Performance

- A performance should be better than or at least equal to that of running the same application on a single-processor system.
- Some design principles considered useful for better performance are as below:
  - **Batch if possible:** Batching often helps in improving performance.

- **Cache whenever possible:** Caching of data at client's side frequently improves overall system performance.
- **Minimize copying of data:** Data copying overhead involves a substantial CPU cost of many operations.
- **Minimize network traffic:** It can be improved by reducing internode communication costs.

Heterogeneity

- This term means the diversity of the distributed systems in terms of hardware, software, platform, etc.
- Modern distributed systems will likely span different:
  - **Hardware devices:** computers, tablets, mobile phones, embedded devices, etc.
  - **Operating System:** MS Windows, Linux, Mac, Unix, etc.
  - **Network:** Local network, the Internet, wireless network, satellite links, etc.
  - **Programming languages:** Java, C/C++, Python, PHP, etc.
  - Different roles of software developers, designers, system managers.

Security

- System must be protected against destruction and unauthorized access.
- Enforcement of Security in a distributed system has the following additional requirements as compared to centralized system:
  - Sender of message should know that message is delivered.
  - Receiver of the message should know that the message was sent by genuine sender.
  - Both sender and receiver should be guaranteed that the content of message were not changed while it is in transfer.

1) What is Computer Network? Give its classification?

- A computer network is a system in which multiple computers are connected to each other to share information and resources.
- The physical connection between networked computing devices is established using either cable media or wireless media.
- The best-known computer network is the Internet.

Figure: Computer Network

Local area network (LAN)

- It is privately-owned networks within a single building or campus of up to a few kilometres in size.
- They are widely used to connect personal computers and workstations in company offices and factories to share resources (e.g., printers) and exchange information.
- LANs are easy to design and troubleshoot
- In LAN, all the machines are connected to a single cable.
- Different types of topologies such as Bus, Ring, Star and Tree are used.
- The data transfer rates for LAN is up to 10 Gbits/s.
- They transfer data at high speeds. High transmission rate are possible in LAN because of the short distance between various computer networks.
- They exist in a limited geographical area.

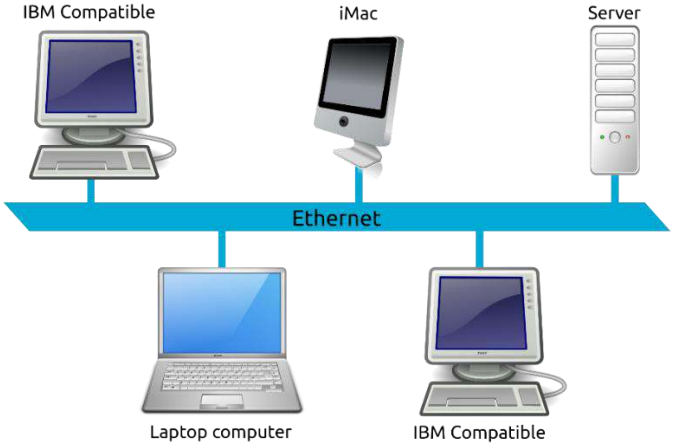


Figure: Local area network (LAN)

Wide area Network (WAN)

- WAN spans a large geographical area, often a country or region.
- WAN links different metropolitan's countries and national boundaries there by enabling easy communication.
- It may be located entirely with in a state or a country or it may be interconnected around the world.
- It contains a collection of machines intended for running user (i.e., application) programs. We will follow traditional usage and call these machines hosts.
- The communication between different users of WAN is established using leased telephone lines or satellite links and similar channels.

Figure: Wide area network (WAN)

Metropolitan area network (MAN)

- MAN is a larger version of LAN which covers an area that is larger than the covered by LAN but smaller than the area covered by WAN.
- A metropolitan area network or MAN covers a city.
- The best-known example of a MAN is the cable television network available in many cities.
- MAN connects two or more LANs.

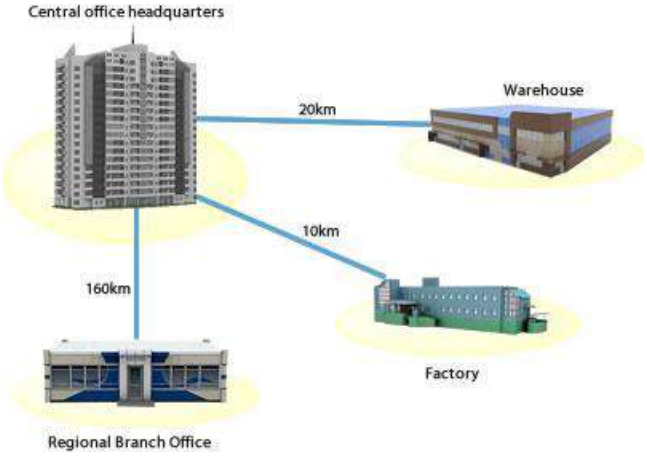


Figure: Metropolitan area network (MAN)

Wireless LAN

- WLAN technology allows computer network devices to have network connectivity without the use of wires, using radio waves.
- These networks use infrared link, Bluetooth or low power radio network operating at a bandwidth of 1-2 Mbps over a distance of 10m.
- At the top level, wireless networks can be classified as wireless LAN, wireless MAN, and wireless WAN.
- The wireless LANs are further classified as Personal Area networks (e.g. Bluetooth, wireless sensor networks) and Business LAN (e.g. 802.11b).

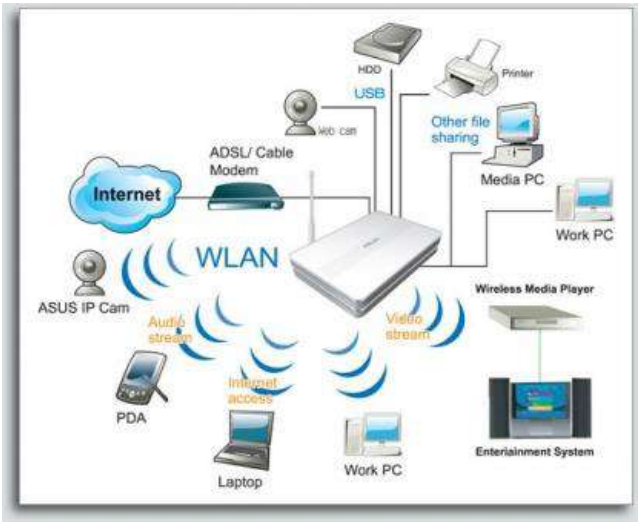


Figure: Wireless LAN

Classification of Network

Basis Of Comparison	LAN	MAN	WAN
Expands to	Local Area Network	Metropolitan Area Network	Wide Area Network
Meaning	A network that connects a group of computers in a small geographical area	It covers relatively large region such as cities, towns	It spans large locality & connects countries together.
Ownership of Network	Private	Private or Public	Private or Public
Design and Maintenance	Easy	Difficult	Difficult
Propagation Delay	Short	Moderate	Long
Speed	High	Moderate	Low
Equipment Needed	Switch, Hub	Modem, Router	Microwave, Radio Transmitters & Receivers
Range(Approx.)	1 to 10 km	In 100 km	Beyond 100 km
Used for	College, School, Hospital	Small towns, City	Country/Continent

2) Explain OSI Model layered architecture.

OSI Layer Architecture

- OSI model is based on a proposal developed by the International Standards Organization (ISO) as a first step toward international standardization of the protocols used in the various layers.
- It was revised in 1995.
- The model is called the OSI (Open Systems Interconnection) Reference Model because it deals with connecting open systems—that is, systems that are open for communication with other systems.
- The OSI model has seven layers.
  - Physical Layer
  - Data Link Layer
  - Network Layer
  - Transport Layer
  - Session Layer
  - Presentation Layer
  - Application Layer

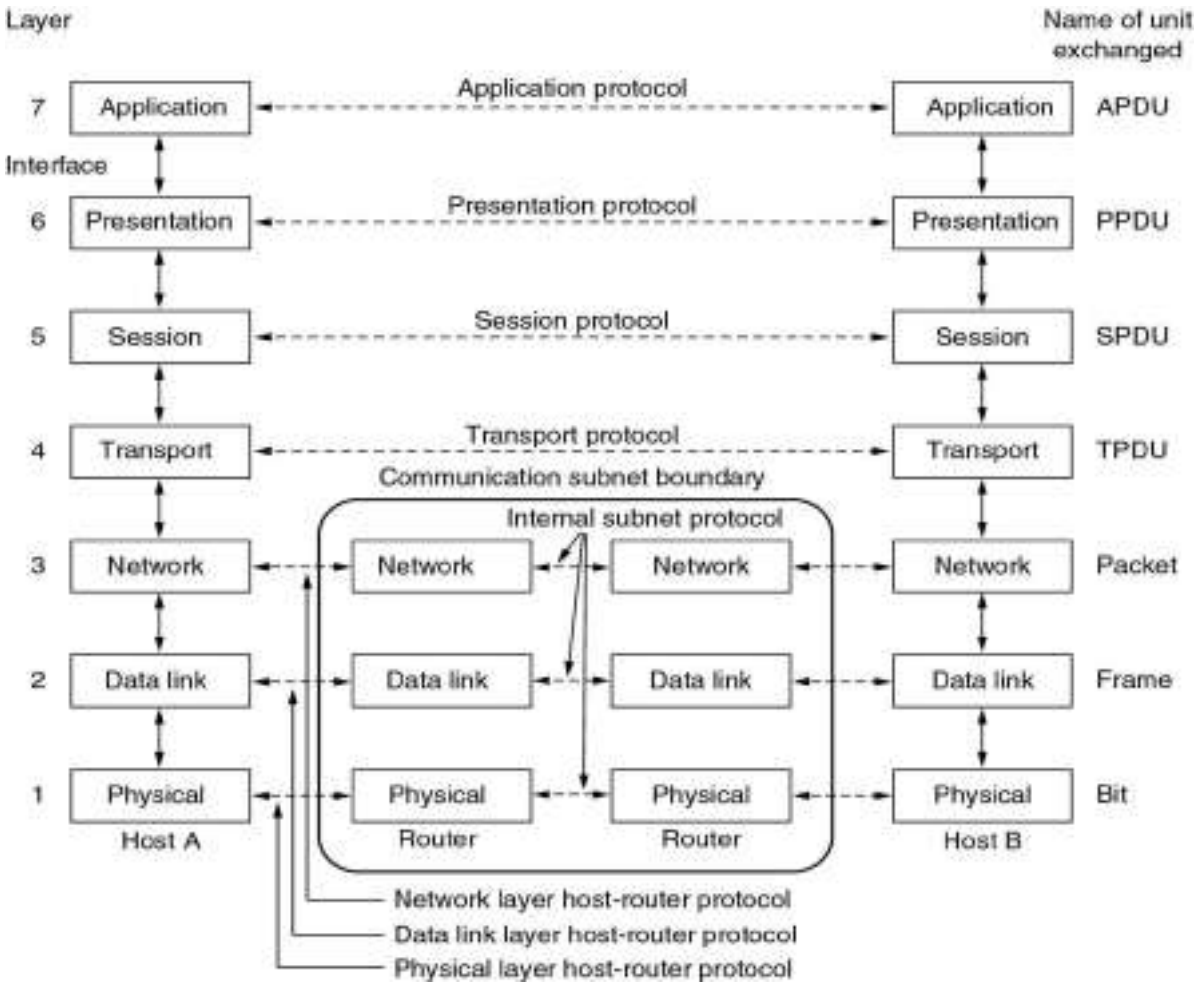


Figure: OSI Reference Model

Physical Layer

- The physical layer, the lowest layer of the OSI model, is concerned with the transmission and reception of the unstructured raw bit stream over a physical medium.
- It describes the electrical/optical, mechanical, and functional interfaces to the physical medium, and carries the signals for all of the higher layers. It provides:
    - Data encoding: modifies the simple digital signal pattern (1s and 0s) used by the PC to better accommodate the characteristics of the physical medium, and to aid in bit and frame synchronization.
    - Transmission technique: determines whether the encoded bits will be transmitted by baseband (digital) or broadband (analog) signalling.
    - Physical medium transmission: transmits bits as electrical or optical signals appropriate for the physical medium.

Data link Layer

- The data link layer provides error-free transfer of data frames from one node to another over the physical layer, allowing layers above it to assume virtually error-free transmission over the link.
- To do this, the data link layer provides:
    - Link establishment and termination: establishes and terminates the logical link between two nodes.
    - Frame traffic control: tells the transmitting node to "back-off" (stop) when no frame buffers are available.
    - Frame sequencing: transmits/receives frames sequentially.
    - Frame acknowledgment: provides/expects frame acknowledgments. Detects and recovers from errors that occur in the physical layer by retransmitting non-acknowledged frames and handling duplicate frame receipt.
    - Frame delimiting: creates and recognizes frame boundaries.
    - Frame error checking: checks received frames for integrity.
    - Media access management: determines when the node "has the right" to use the physical medium.

Network Layer

- The network layer controls the operation of the subnet, deciding which physical path the data should take based on network conditions, priority of service, and other factors.
- To do this, the data link layer provides:
    - Routing: routes frames among networks.
    - Subnet traffic control: routers (network layer intermediate systems) can instruct a sending station to "throttle back" its frame transmission when the router's buffer fills up.
    - Frame fragmentation: if it determines that a downstream router's maximum transmission unit (MTU) size is less than the frame size, a router can fragment a frame for transmission and re-assembly at the destination station.
    - Logical-physical address mapping: translates logical addresses or names, into physical addresses.

- Subnet usage accounting: has accounting functions to keep track of frames forwarded by subnet intermediate systems, to produce billing information.

Transport Layer

- The transport layer ensures that messages are delivered error-free, in sequence, and with no losses or duplications. It relieves (release) the higher layer protocols from any concern with the transfer of data between them and their peers.
- The size and complexity of a transport protocol depends on the type of service it can get from the network layer. For a reliable network layer with virtual circuit capability, a minimal transport layer is required. If the network layer is unreliable and/or only supports datagrams, the transport protocol should include extensive error detection and recovery.
- The transport layer provides:
  - Message segmentation: accepts a message from the (session) layer above it, splits the message into smaller units (if not already small enough), and passes the smaller units down to the network layer. The transport layer at the destination station reassembles the message.
  - Message acknowledgment: provides reliable end-to-end message delivery with acknowledgments.
  - Message traffic control: tells the transmitting station to "back-off" when no message buffers are available.
- Typically, the transport layer can accept relatively large messages, but there are strict message size limits imposed by the network (or lower) layer. Consequently, the transport layer must break up the messages into smaller units, or frames, prepending a header to each frame.
- The transport layer header information must then include control information, such as message start and message end flags, to enable the transport layer on the other end to recognize message boundaries.
- In addition, if the lower layers do not maintain sequence, the transport header must contain sequence information to enable the transport layer on the receiving end to get the pieces back together in the right order before handing the received message up to the layer above.

Session Layer

- The session layer allows session establishment between processes running on different stations. It provides:
  - Session establishment, maintenance and termination: allows two application processes on different machines to establish, use and terminate a connection, called a session.
  - Session support: performs the functions that allow these processes to communicate over the network, performing security, name recognition, logging, and so on.

Presentation Layer

- The presentation layer formats the data to be presented to the application layer. It can be viewed as the translator for the network. This layer may translate data from a format used by the application layer into a common format at the sending station, then translate the common format to a format known to the application layer at the receiving station.
- The presentation layer provides:
  - Character code translation: for example, ASCII to EBCDIC.
  - Data conversion: bit order, CR-CR/LF, integer-floating point, and so on.

- Data compression: reduces the number of bits that need to be transmitted on the network.
- Data encryption: encrypt data for security purposes. For example, password encryption.

Application Layer

- The application layer serves as the window for users and application processes to access network services.
- This layer contains a variety of commonly needed functions:
  - Resource sharing and device redirection
  - Remote file access
  - Remote printer access
  - Inter-process communication
  - Network management
  - Directory services
  - Electronic messaging (such as mail)
  - Network virtual terminals

3) Explain TCP/IP Reference model.

- Transmission Control Protocol/Internet Protocol (TCP/IP) protocol suite is the engine for the Internet and networks worldwide.
- TCP/IP either combines several OSI layers into a single layer, or does not use certain layers at all.
- TCP/IP is a set of protocols developed to allow cooperating computers to share resources across the network.
- The TCP/IP model has five layers.
  - Application Layer
  - Transport Layer
  - Internet Layer
  - Data Link Layer
  - Physical Network

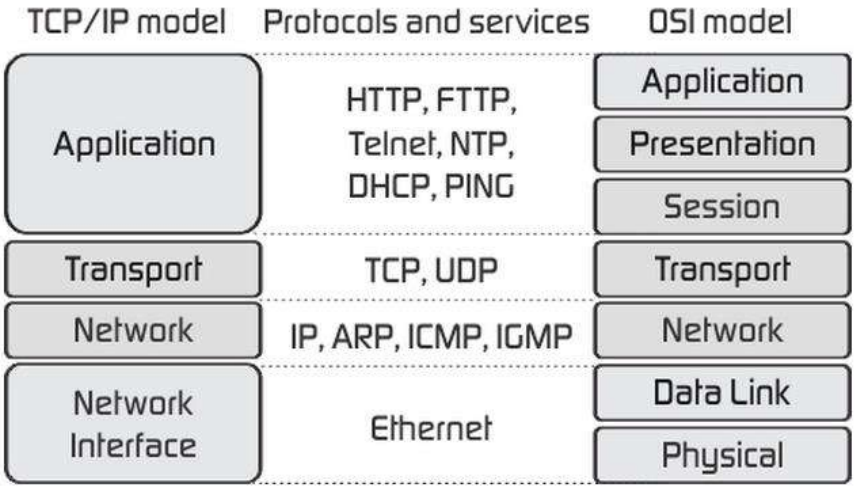


Figure: TCP/IP Reference Model

- As we can see from the above figure, presentation and session layers are not there in TCP/IP model. Also note that the Network Access Layer in TCP/IP model combines the functions of Data Link Layer and Physical Layer.

Application Layer

- Application layer is the top most layer of four layer TCP/IP model.
- Application layer is present on the top of the Transport layer.
- Application layer defines TCP/IP application protocols and how host programs interface with Transport layer services to use the network.
- Application layer includes all the higher-level protocols like DNS (Domain Naming System), HTTP (Hypertext Transfer Protocol), Telnet, SSH, FTP (File Transfer Protocol), TFTP (Trivial File Transfer Protocol), SNMP (Simple Network Management Protocol), SMTP (Simple Mail Transfer Protocol), DHCP (Dynamic Host Configuration Protocol), X Windows, RDP (Remote Desktop Protocol) etc.

Transport Layer

- The purpose of Transport layer is to permit devices on the source and destination hosts to carry on a conversation.
- Transport layer defines the level of service and status of the connection used when transporting data.
- The transport layer provides the end-to-end data transfer by delivering data from an application to its remote peer.
- The most-used transport layer protocol is the Transmission Control Protocol (TCP), which provides:
    - Reliable delivery data
    - Duplicate data suppression
    - Congestion control
    - Flow control
- Another transport layer protocol is the User Datagram Protocol (UDP), which provides:
    - Connectionless
    - Unreliable
    - Best-effort service
- UDP is used by applications that need a fast transport mechanism and can tolerate the loss of some data.

Network Layer (Internet Layer)

- The internet layer also called the network layer.
- Internet layer pack data into data packets known as IP datagrams, which contain source and destination address (logical address or IP address) information that is used to forward the datagrams between hosts and across networks.
- The Internet layer is also responsible for routing of IP datagrams.
- Internet Protocol (IP) is the most important protocol in this layer.
- It is a connectionless protocol that does not assume reliability from lower layers. IP does not provide reliability, flow control or error recovery.

- IP provides a routing function that attempts to deliver transmitted messages to their destination.
- These message units in an IP network are called an IP datagram.
- Example: IP, ICMP, IGMP, ARP, and RARP.

Network Interface Layer (Network Access Layer)

- Network Access Layer defines details of how data is physically sent through the network, including how bits are electrically or optically signalled by hardware devices that interface directly with a network medium, such as coaxial cable, optical fiber, or twisted pair copper wire.
- The protocols included in Network Access Layer are Ethernet, Token Ring, FDDI, X.25, Frame Relay etc.

OSI(Open System Interconnection)	**TCP/IP (Transmission Control Protocol/ Internet Protocol)**
OSI provides layer functioning and also defines functions of all the layers.	TCP/IP model is more based on protocols and protocols are not flexible with other layers.
In OSI model the transport layer guarantees the delivery of packets	In TCP/IP model the transport layer does not guarantees delivery of packets.
Follows horizontal approach	Follows vertical approach.
OSI model has a separate presentation layer	TCP/IP doesn't have a separate presentation layer
OSI is a general model.	TCP/IP model cannot be used in any other application.
Network layer of OSI model provide both connection oriented and connectionless service.	The Network layer in TCP/IP model provides connectionless service.
OSI model has a problem of fitting the protocols in the model	TCP/IP model does not fit any protocol
Protocols are hidden in OSI model and are easily replaced as the technology changes.	In TCP/IP replacing protocol is not easy.
OSI model defines services, interfaces and protocols very clearly and makes clear distinction between them.	In TCP/IP it is not clearly separated its services, interfaces and protocols.
It has 7 layers	It has 4 layers

4) Explain Inter process communication (IPC) in message passing system.

- Inter process communication (IPC) basically requires information sharing among two or more processes.

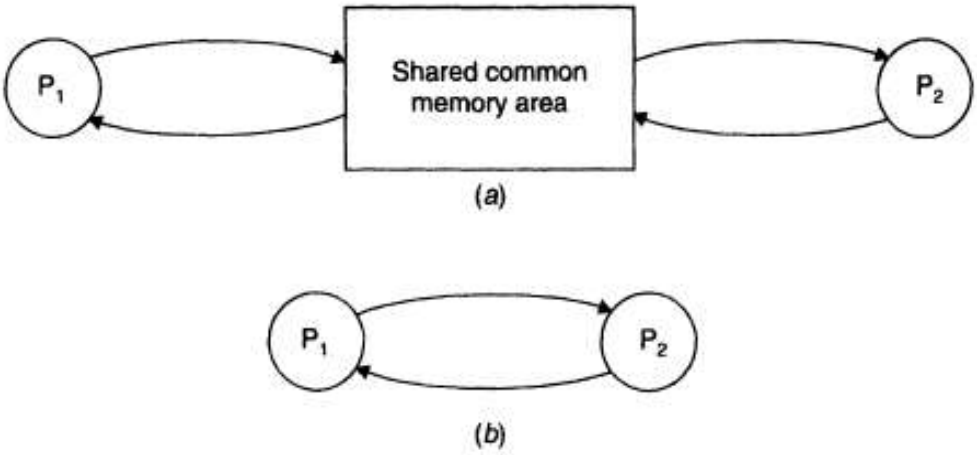


Figure: Inter process communication (a) Shared data approach (b) Message passing approach

- 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.

5) Explain 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 inter process 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.
- Some optimizations normally adopted for efficiency include the following:
- Avoiding the costs of establishing and terminating connections between the same pair of processes for each and every message exchange between them.
- Minimizing the costs of maintaining the connections;
- Piggybacking of acknowledgement of previous messages with the next message during a connection between a sender and a receiver that involves several message exchanges.

Reliability

- A good IPC protocol can cope with failure problems and guarantees the delivery of a message.
- Handling of lost messages involves acknowledgement and retransmission on the basis of timeouts.
- 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: It ensures that every message sent to a group of receivers will be delivered to either all of them or none of them.
    - Ordered delivery: It ensures that messages arrive to all receivers in an order acceptable to the application.
    - Survivability: It 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.
- The IPC primitives should be such that user have the flexibility to choose and specify types and levels of reliability and correctness requirement of their applications.
- IPC primitives must also have the flexibility to permit any kind of control flow between the cooperating processes, including synchronous and asynchronous send/receive.

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.
- The application written by using primitives of IPC protocols of the message passing system should be portable.
- This may require use of an external data representation format for the communication taking place between two or more processes running on computers of different architectures.

6) What are 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.

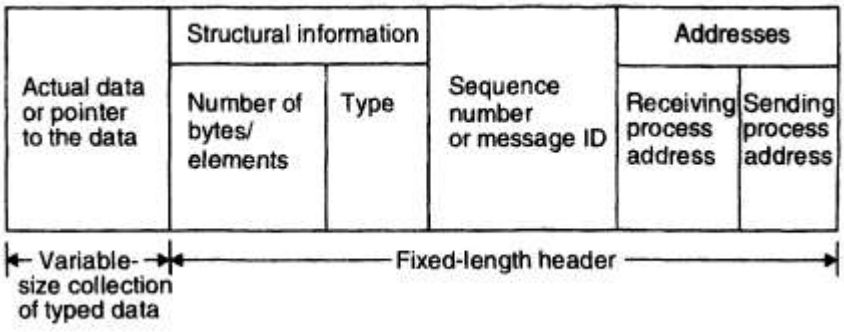


Figure: A typical message structure.

- The header usually consists of the following elements:
- 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?

7) Explain Buffering in standard message passing model.

- 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.

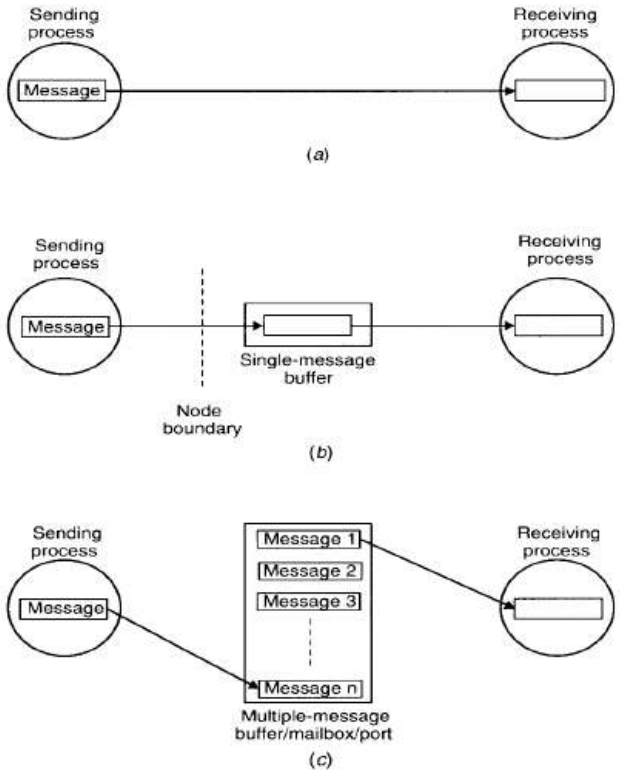


Figure: Message transfer with (a) no buffering (b) single message buffer (c) multi buffer

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.
- The buffer overflow problem can be dealt with in one of the following two ways:
- Unsuccessful communication.
    - In this method, message transfers simply fail, whenever there is no more buffer space and an error is returned.
- Flow-controlled communication.
    - The sender is blocked until the receiver accepts some messages, thus creating space in the buffer for new messages.
    - This method introduces a synchronization between the sender and the receiver and may result in unexpected deadlocks.
    - Due to the synchronization imposed, the asynchronous send does not operate in the truly asynchronous mode for all send commands.

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.

5) Explain Failure Handling in IPC.

- During inter process communication partial failures such as a node crash or communication link failure may lead to the following problems:

Loss of request message

- This may happen either due to the failure of communication link between the sender and receiver or because the receiver's node is down at the time the request message reaches there.

Loss of response message

- This may happen either due to the failure of communication link between the sender and receiver or because the sender's node is down at the time the response message reaches there.

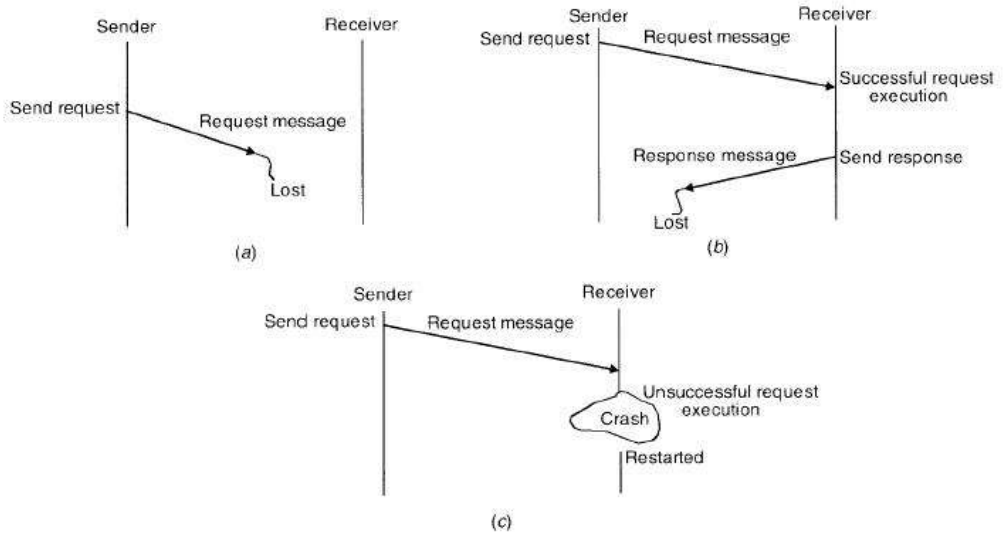


Figure: Possible problems in IPC when (a) Request message is lost (b) Response message is lost (c) Receiver's computer crashed.

Four-message reliable IPC protocol for client-server communication between two processes

- The client sends a request message to the server.
- When the request message is received at the server's machine, the kernel of that machine returns an acknowledgment message to the kernel of the client machine.

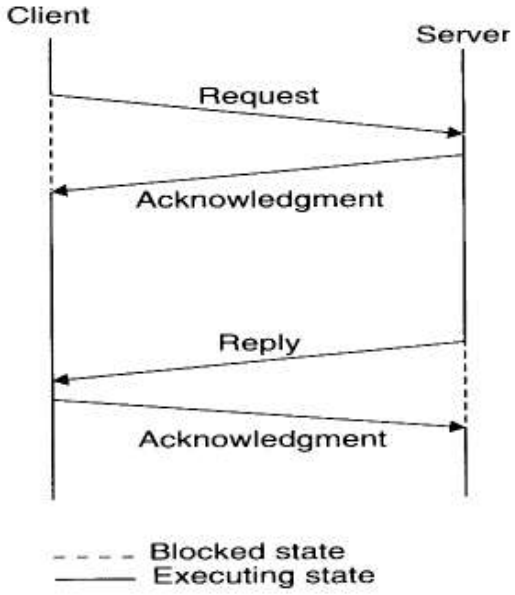


Figure: Four message reliable IPC protocol for client server

- If the acknowledgment is not received within the timeout period, the kernel of the client machine retransmits the request message.
- When the server finishes processing the client's request it returns a reply message to the client.
- When the reply is received at client machine, the kernel of that machine returns an acknowledgment message to the kernel of the server machine.

- If the acknowledgment message is not received within the timeout period, the kernel of the server machine retransmits the reply message.

Three message reliable IPC message protocol

- The client sends a request message to the server.
- When the server finishes processing the client's request, it returns a reply message (containing the result of processing) to the client.
- The client remains blocked until the reply is received.

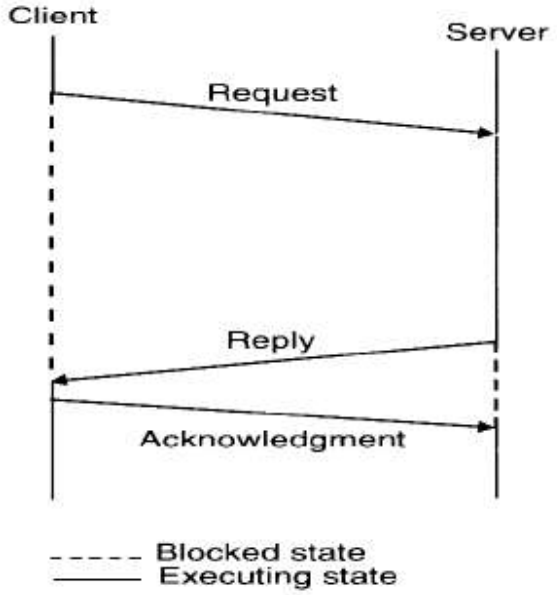


Figure: Three message reliable IPC protocol for client server

- If the reply is not received within the timeout period, the kernel of the client machine retransmits the request message.
- When the reply message is received at the client's machine, the kernel of that machine returns an acknowledgment message to the kernel of the sever machine.
- If the acknowledgment message is not received within the timeout period, the kernel of the server machine retransmits the reply message.
- If the request message is lost, it will be retransmitted only after the timeout period, which has been set to a large value to avoid unnecessary retransmission of the request message.
- On the other hand, if the timeout value is not set properly taking into consideration the long time needed for request processing, unnecessary retransmissions of the request message will take place.

Two message reliable IPC message protocol

- The client sends a request message to the server and remains blocked until a reply is received from the server.

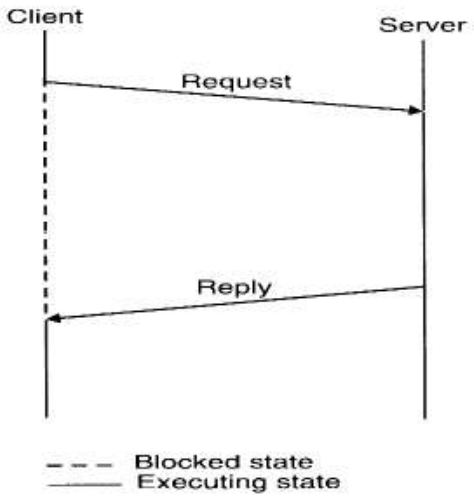


Figure: Two message reliable IPC protocol for client server

- When the server finishes processing the client's request, it returns a reply message (containing the result of processing) to the client.
- If the reply is not received within the timeout period, the kernel of the client machine retransmits the request message.

6) Explain basic concept of client server model.

- The structure of the operation system is like a group of cooperating processes called the servers, which offer services to the users called the clients.
- Both run on the same microkernel as the user processes.
- A machine can run as single/multiple clients or single/multiple servers, or a mix of both.
- This model depicted in Figure2.23 uses the connectionless Request Reply protocol thus reducing the overhead of the connection oriented TCP/IP or OSI protocol.

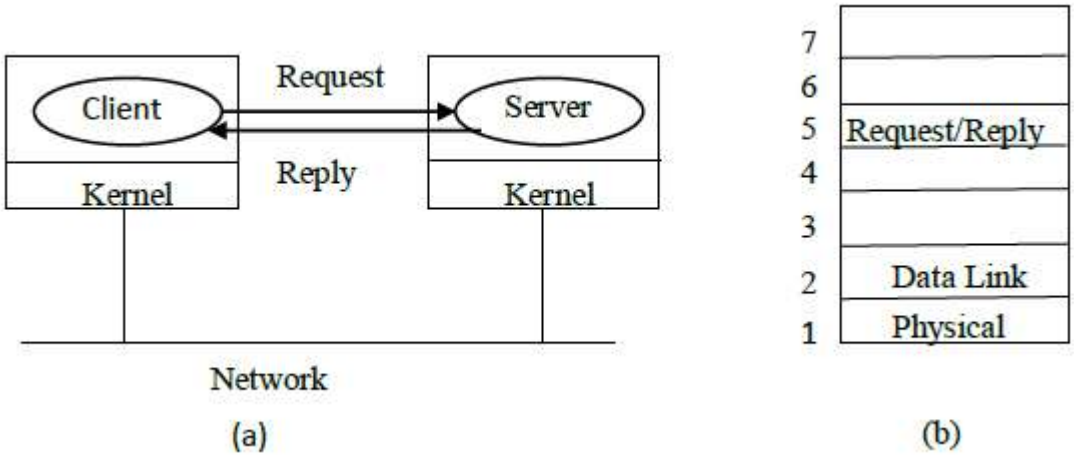


Figure: Client Server model

- The Request Reply protocol works as follows:
  - The client requesting the service sends a request message to the server.
  - The server completes the task and returns the result indicating that it has performed the task or returns an error message indicating that it has not performed the task.

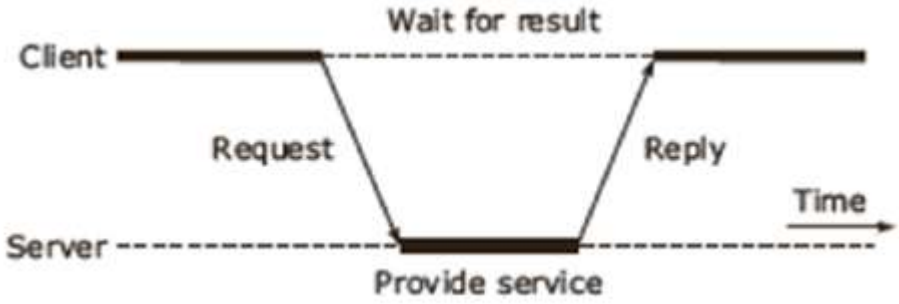


Figure: The Client server interaction

  - Figure shows the client server interaction.
  - Here reply serves as acknowledgement to the request.
- As seen in the above figure(b), only three layers of the protocol are used.
- The physical and the data link layer are responsible for transferring packets between the client and the server through hardware.
- There is no need for routing and establishment of connection.
- The Request Reply protocol defines the set of request and replies to the corresponding requests. The session management and other higher layers are not required.

7) Write a note on Implementation of Client Server Model.

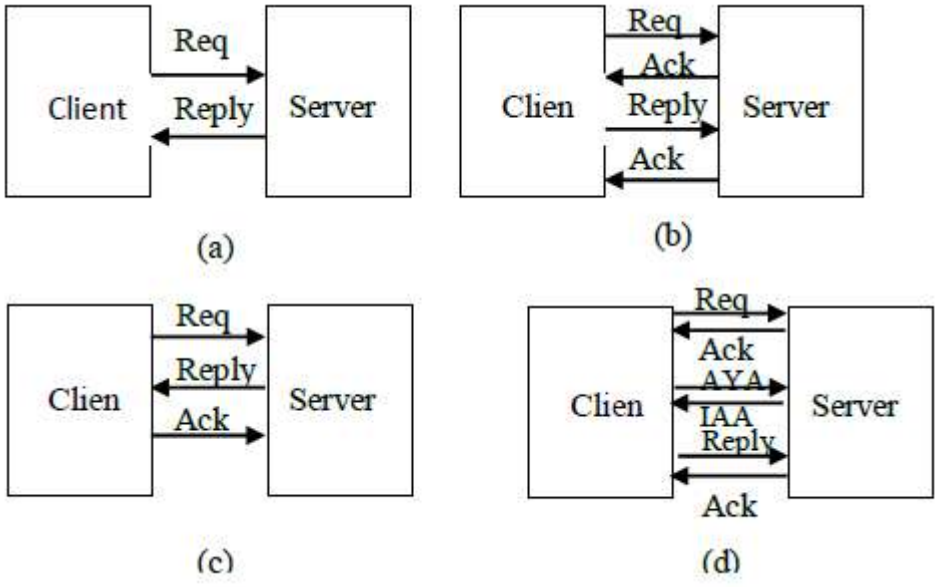


Figure: Common packet message sequences

- All networks have a maximum packet size.
- If the message is large, it is split into multiple packets and each packet is transmitted individually.
- What happens if the packets are lost or garbled or arrive at the server out of sequence?

- Solution is assigning a sequence number to each packet depending on the order of the packets.
- In case of multiple packets per message, should the receiver acknowledge each packet or just the last packet?
- If an acknowledgement is sent for each packet, then only lost packets are retransmitted, but it increases the message traffic.
- The alternative is to send a single acknowledgement for all the packets.
- This results in transmission of fewer acknowledgements.
- The client may not understand which packet is lost and hence may end up retransmitting all packets.
- The choice of the method depends on the loss rate of the network.
- Following are different types of packets transmitted across the network
    - REQ: Request packet is used to send the request from the client to the server.
    - Reply: This message is used to carry the result from the server to the client.
    - ACK: Acknowledgement packet is used to send the correct receipt of the packet to the sender.
    - Are You Alive (AYA)? This packet is sent in case the server takes a long time to complete the client's request.
    - I am Alive (IAA): The server, if active, replies with this packet.
- It implies that the client kernel should wait for the reply.
- If no response is received for an AYA message, the client resends the message.
- Figure shows some of the common packet message sequences.

8) Explain Remote procedure call (RPC) and its implementation.

- 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.

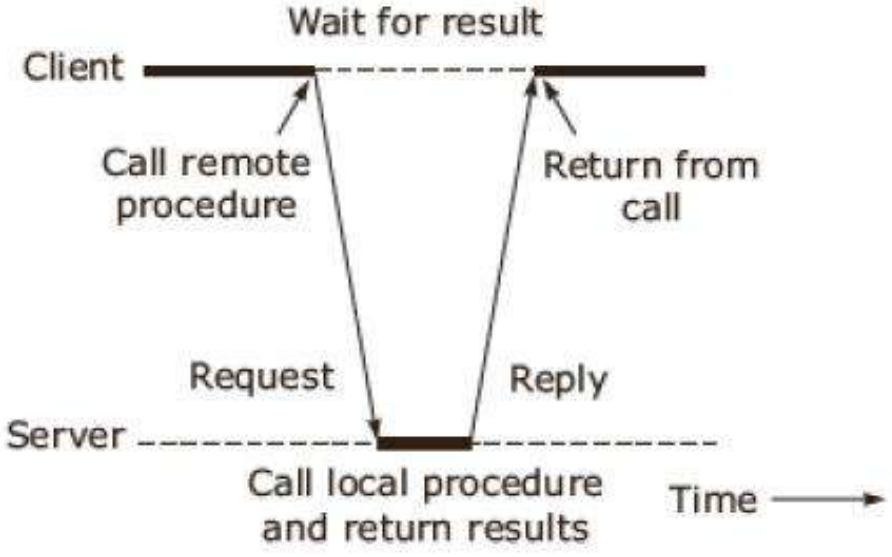


Figure: RPC Model

- It includes mainly five elements:
  - The Client
  - The Client stub (stub: Piece of code used for converting parameters)
  - The RPCRuntime (RPC Communication Package)
  - The Server stub
  - The Server

The Client

- It is user process which initiates a remote procedure call.
- The client makes a perfectly normal call that invokes a corresponding procedure in the client stub.

The Client stub

- On receipt of a request it packs a requirement into a message and asks to RPCRuntime to send.
- On receipt of a result it unpacks the result and passes it to client.

RPCRuntime

- It handles transmission of messages between client and server.

The Server stub

- It unpacks a call request and make a perfectly normal call to invoke the appropriate procedure in the server.
- On receipt of a result of procedure execution it packs the result and asks to RPCRuntime to send.

The Server

- It executes an appropriate procedure and returns the result from a server stub.

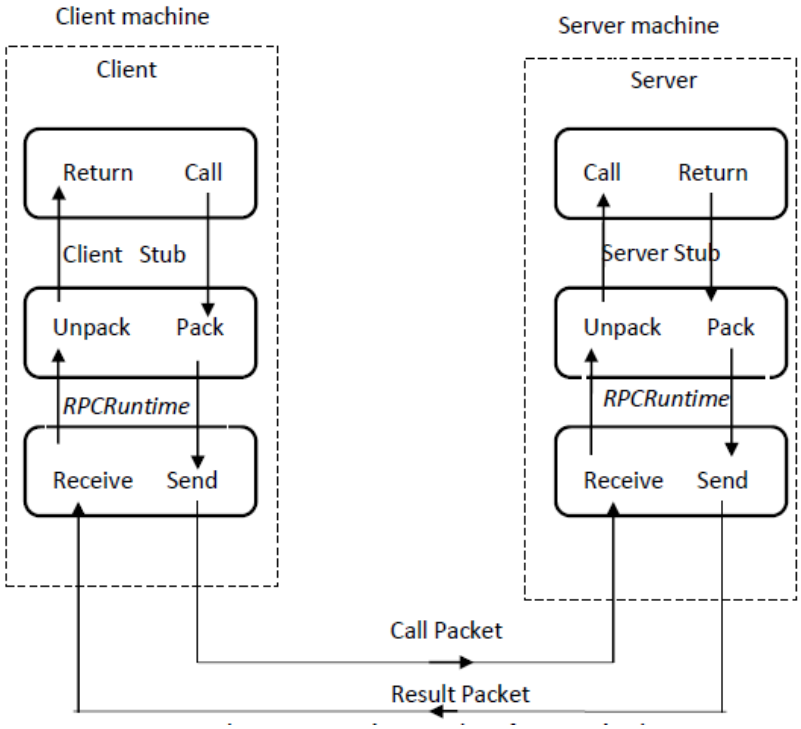


Figure: Implementation of RPC mechanism

Steps of Implementation are as follows:

1) The client calls a local procedure, called the client stub.
2) Network messages are sent by the client stub to the remote system (via a system call to the local kernel using sockets interfaces).
3) Network messages are transferred by the kernel to the remote system via some protocol (either connectionless or connection-oriented).
4) A server stub, sometimes called the skeleton, receives the messages on the server. It unmarshals the arguments from the messages.
5) The server stub calls the server function, passing it the arguments that it received from the client.
6) When server function is finished, it returns to the server stub with its return values.
7) The server stub converts the return values, if necessary, and marshals them into one or more network messages to send to the client stub.
8) Messages get sent back across the network to the client stub.
9) The client stub reads the messages from the local kernel.
10) The client stub then returns the results to the client function, converting them from the network representation to a local one if necessary.

9) Explain Implementation Issues in Remote procedure call (RPC).

RPC protocols

- The first issue is the choice of the RPC protocol.

  Connection oriented protocol

  - The client is bound to the server and a connection is established between them.
  - The advantage: communication becomes much easier.
  - The disadvantage: especially over a LAN, is the performance loss.

  Connectionless protocol

  - IP and UDP are easy to use and fit in well with existing UNIX systems and networks such as the Internet, but downside is performance.
- Packet and message length is another issue.
- Doing an RPC has a large, fixed overhead, independent of the amount of data sent.
- Thus reading a 64K file in a single 64K RPC is vastly more efficient than reading it in 64 1K RPCs.
- It is therefore important that the protocol and network should allow large transmissions.
- Some RPC systems are limited to small sizes.
- In addition, many networks cannot handle large packets so a single RPC will have to be split over multiple packets, causing extra overhead.

Acknowledgement

- If RPCs are broken up into many small packets, a new issue arises: Should individual packets be acknowledged or not?

- Suppose, for example, that a client wants to write a 4K block of data to a file server, but the system cannot handle packets larger than 1K.

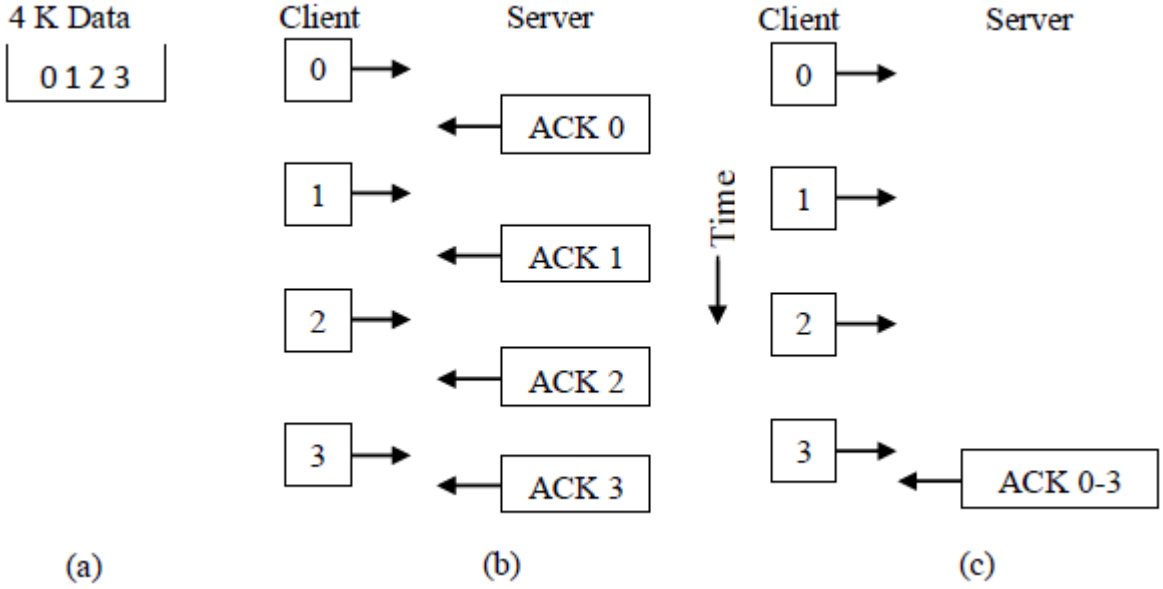


Figure: Acknowledgement to RPC messages

- One strategy, known as a stop-and-wait protocol, is for the client to send packet 0 with the first 1K, then wait for an acknowledgement from the server, as illustrated in Figure (b).
- Then the client sends the second 1K, waits for another acknowledgement, and so on.
- The alternative, often called a blast protocol, is simply for the client to send all the packets as fast as it can.
- With this method, the server acknowledges the entire message when all the packets have been received, not one by one.
- The blast protocol is illustrated in Figure (c).

Critical path

- The sequence of instructions that is executed on every RPC is called the critical path, and is shown in Figure.

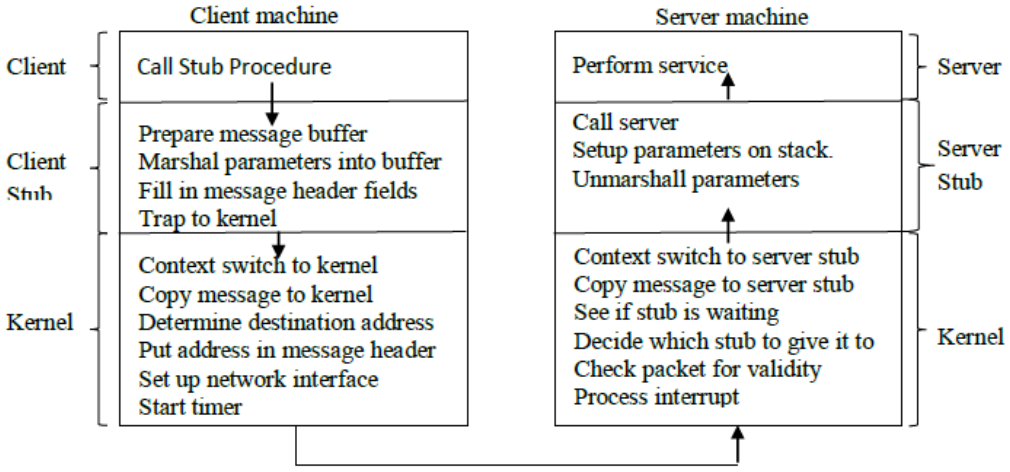


Figure: Critical path from client to server

- Even for the null RPC, the dominant costs are the context switch to the server stub when a packet arrives, the interrupt service routine, and moving the packet to the network interface for transmission.
- For the 1440-byte RPC, the picture changes considerably, with the Ethernet transmission time now being the largest single component, with the time for moving the packet into and out of the interface coming in close behind.

Copying

- An issue that frequently dominates RPC execution times is copying.
- The number of times a message must be copied varies from 1 to about 8, depending on the hardware, software and type of call.
- In the best case, the network chip can DMA the message directly out of the client stub's address space onto the network (copy 1), depositing it in the server kernel's memory in real.
- In the worst case, the kernel copies the message from the client stub into a kernel buffer for subsequent transmission, either because it is not convenient to transmit directly from user space or the network is currently busy (copy 1).
- Later, the kernel copies the message, in software, to a hardware buffer on the network interface board (copy 2).
- At this point, the hardware is started, causing the packet to be moved over the network to the interface board on the destination machine (copy 3).
- When the packet-arrived interrupt occurs on the server's machine, the kernel copies it to a kernel buffer (copy 4).
- Finally, the message has to be copied to the server stub (copy 5).
- Suppose that it takes an average of 500 nsec to copy a 32-bit word; then with eight copies, each word needs 4 msec, no matter how fasts the network itself is.

Timer management

- Huge amount of machine time is wasted in managing the timers.
- Setting a timer requires building a data structure specifying when the timer is to expire and what is to be done after timer expires.
- The data structure is then inserted into a list consisting of the other pending timers.

1) What is Synchronization? Explain Clock Synchronization and implementation of computer clock.

- Synchronization is coordination with respect to time, and refers to the ordering of events and execution of instructions in time.
- It is often important to know when events occurred and in what order they occurred.
- Clock synchronization deals with understanding the temporal ordering of events produced by concurrent processes.
- It is useful for synchronizing senders and receivers of messages, controlling joint activity, and the serializing concurrent access to shared objects.
- The goal is that multiple unrelated processes running on different machines should be in agreement with and be able to make consistent decisions about the ordering of events in a system.
- Another aspect of clock synchronization deals with synchronizing time-of-day clocks among groups of machines.
- In this case, we want to ensure that all machines can report the same time, regardless of how imprecise their clocks may be or what the network latencies are between the machines.
- A computer clock usually consists of three components-a quartz crystal that oscillates at a well-defined frequency, a counter register, and a constant register.
- The constant register is used to store a constant value that is decided based on the frequency of oscillation of the quartz crystal.
- The counter register is used to keep track of the oscillations of the quartz crystal.
- That is, the value in the counter register is decremented by 1 for each oscillation of the quartz crystal.
- When the value of the counter register becomes zero, an interrupt is generated and its value is reinitialized to the value in the constant register.
- Each interrupt is called a clock tick.
- To make the computer clock function as an ordinary clock used by us in our day-today life, the following things are done:
    - The value in the constant register is chosen so that 60 clock ticks occur in a second.
    - The computer clock is synchronized with real time (external clock). For this, two more values are stored in the system-a fixed starting date and time and the number of ticks.
    - For example, in UNIX, time begins at 0000 on January 1, 1970.
    - At the time of initial booting, the system asks the operator to enter the current date and time.
    - The system converts the entered value to the number of ticks after the fixed starting date and time.
    - At every clock tick, the interrupt service routine increments the value of the number of ticks to keep the clock running.

2) Explain Drifting of Clock.

- A clock always runs at a constant rate because its quartz crystal oscillates at a well-defined frequency.
- However, due to differences in the crystals, the rates at which two clocks run are normally different from each other.

- The difference in the oscillation period between two clocks might be extremely small, but the difference accumulated over many oscillations leads to an observable difference in the times of the two clocks, no matter how accurately they were initialized to the same value.
- Therefore, with the passage of time, a computer clock drifts from the real-time clock that was used for its initial setting.
- For clocks based on a quartz crystal, the drift rate is approximately 10-6, giving a difference of 1 second every 1,000,000 seconds, or 11.6 days.
- Hence a computer clock must be periodically resynchronized with the real-time clock to keep it non faulty.
- Even non faulty clocks do not always maintain perfect time.
- A clock is considered non faulty if there is a bound on the amount of drift from real time for any given finite time interval.
- As shown in Figure, after synchronization with a perfect clock, slow and fast clocks drift in opposite directions from the perfect clock.
- This is because for slow clocks dC/dt < 1 and for fast clocks dC/dt > 1.

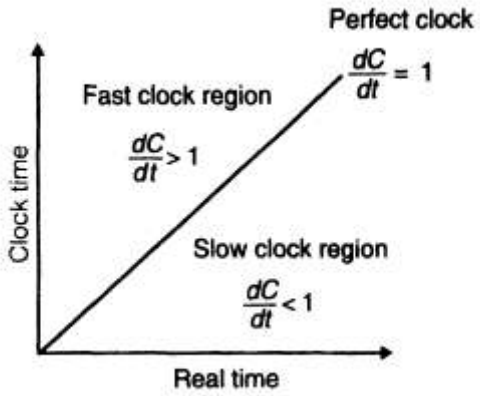


Figure: Slow, perfect, and fast clocks.

- A distributed system requires the following types of clock synchronization:
  - Synchronization of the computer clocks with real-time (or external) clocks.
  - This type of synchronization is mainly required for real-time applications.
  - That is, external clock synchronization allows the system to exchange information about the timing of events with other systems and users.
  - An external time source that is often used as a reference for synchronizing computer clocks with real time is the Coordinated Universal Time (UTC).
  - Mutual (or internal) synchronization of the clocks of different nodes of the system.
  - This type of synchronization is mainly required for those applications that require a consistent view of time across all nodes of a distributed system as well as for the measurement of the duration of distributed activities that terminate on a node different from the one on which they start.

3) Classify and explain clock synchronization algorithms.

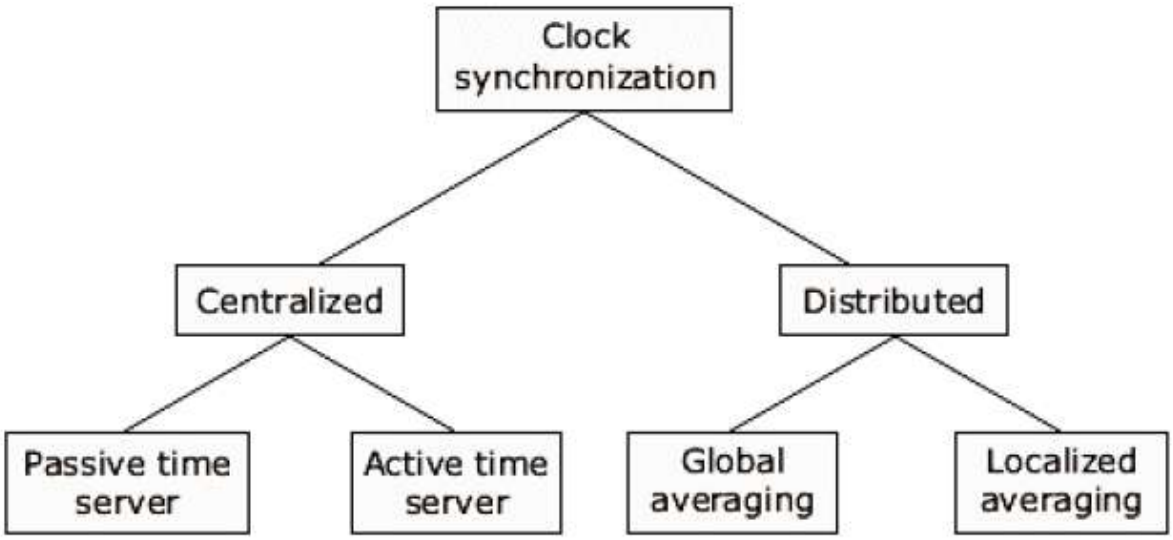


Figure: Classification of Synchronization Algorithms

Centralized clock synchronization algorithms

- In centralized clock synchronization algorithms one node has a real-time receiver.
- This node, called the time server node whose clock time is regarded as correct and used as the reference time.
- The goal of these algorithms is to keep the clocks of all other nodes synchronized with the clock time of the time server node.
- Depending on the role of the time server node, centralized clock synchronization algorithms are again of two types –
    - Passive Time Sever algorithm
    - Active Time Server algorithm

Distributed clock synchronization algorithms

- Externally synchronized clocks are also internally synchronized
- Each node's clock is independently synchronized with real time.
- All clocks of system remain mutually synchronized.
- Each node is equipped with real time receiver so that each node can be independently synchronized.
- Theoretically internal synchronization of clock is not required.
- In practice, due to inherent inaccuracy of real time clocks, different real time clocks produce different time.
- Internal synchronization is performed for better accuracy.
- Types of internal Synchronization:
    - Global averaging
    - Localized Averaging

4) Explain Passive time server algorithms.

Passive time server Algorithm

- Each node periodically sends a message called 'time=?' to the time server.
- When the time server receives the message, it responds with 'time=T' message.
- Assume that client node has a clock time of T0 when it sends 'time=?' and time T1 when it receives the 'time=T' message.
- T0 and T1 are measured using the same clock, thus the time needed in propagation of message from time server to client node would be (T1- T0)/2
- When client node receives the reply from the time server, client node is readjusted to Tserver+(T1-T0)/2.
- Two methods have been proposed to improve estimated value
    - Let the approximate time taken by the time server to handle the interrupt and process the message request message 'time=?' is equal to I.
    - Hence, a better estimate for time taken for prapogation of response message from time server node to client node is taken as (T1- T0-I)/2
    - Clock is adjusted to the value Tserver+(T1- T0-I)/2

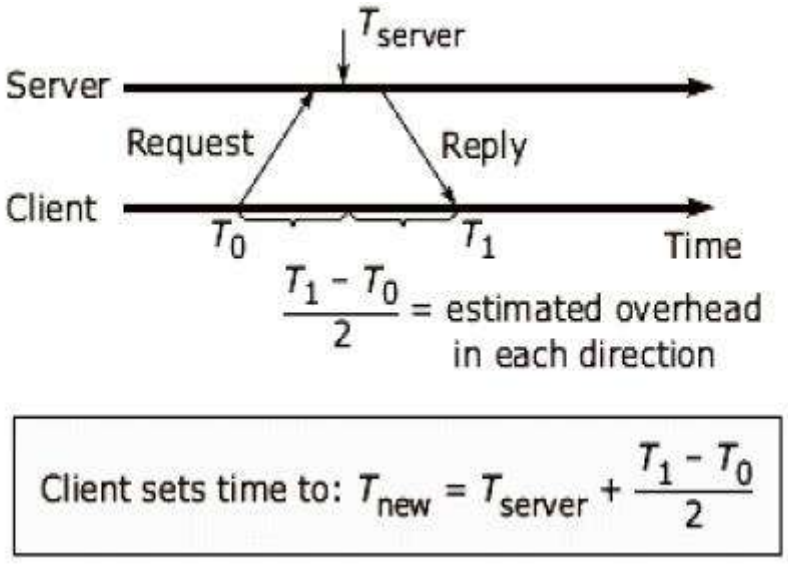


Figure: Time approximation using passive time server algorithm

- Christian method.
    - This method assumes that a certain machine, the time server is synchronized to the UTC in some fashion called T.
    - Periodically all clock is synchronized with time server.
    - Other machines send a message to the time server, which responds with T in a response, as fast as possible.
    - The interval (T1- T0) is measured many times.
    - Those measurements in which (T1- T0) exceeds a specific threshold values are considered to be unreliable and are discarded.
    - Only those values that fall in the range (T1- T0-2Tmin) are considered for calculating the correct time.

- For all the remaining measurements, an average is calculated which is added to T.
- Alternatively, measurement for which value of (T1- T0) is minimum, is considered most accurate and half of its value is added to T.

5) Explain Active time server algorithms.

- It is also called Berkeley Algorithm.
- An algorithm for internal synchronization of a group of computers.
- A master polls to collect clock values from the others (slaves).
- The master uses round trip times to estimate the slaves' clock values.
- It obtains average from participating computers.
- It sends the required adjustment to the slaves.
- If master fails, can elect a new master to take over.
- It Synchronizes all clocks to average.

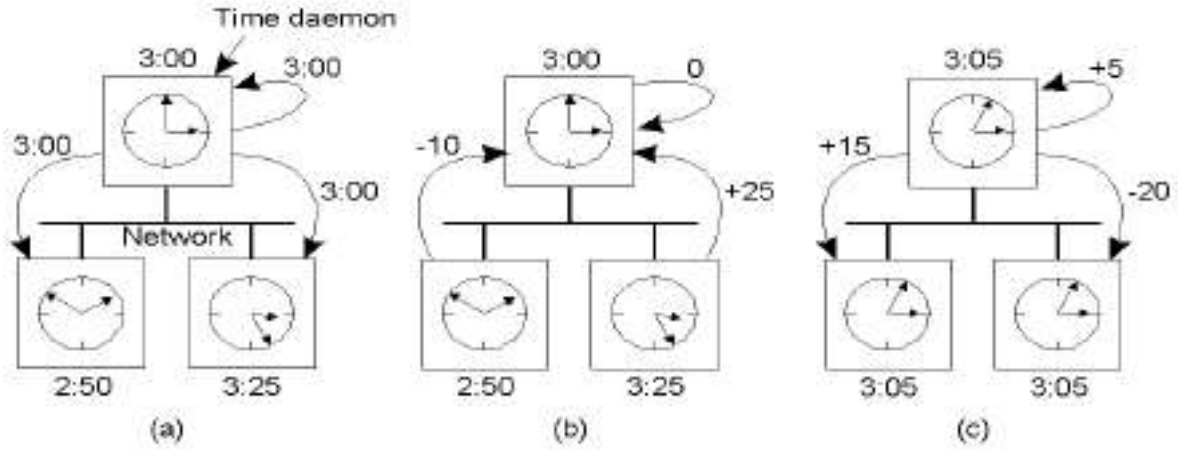


Figure: (a) The time daemon asks all the other machines for their clock values. (b) The machines answer. (c) The time daemon tells everyone how to adjust their clock.

- Here the time server (actually, a time daemon) is active, polling every machine periodically to ask what time it is there.
- Based on the answers, it computes an average time and tells all the other machines to advance their clocks to the new time or slow their clocks down until some specified reduction has been achieved.
- In Figure (a), at 3:00, the time daemon tells the other machines its time and asks for theirs.
- In Figure (b), they respond with how far ahead or behind the time daemon they are.
- Armed with these numbers, the time daemon computes the average and tells each machine how to adjust its clock Figure (c).

6) Explain Global averaging algorithm and Local averaging algorithm.

Global averaging algorithm

- One class of decentralized clock synchronization algorithms works by dividing time into fixed-length resynchronization intervals.
- The ith interval starts at T0+iR and runs until T0+(i+1) R, where T0 is an agreed upon moment in the past, and R is a system parameter.
- At the beginning of each interval, every machine broadcasts the current time according to its clock.
- Because the clocks on different machines do not run at exactly the same speed, these broadcasts will not happen precisely simultaneously.
- After a machine broadcasts its time, it starts a local timer to collect all other broadcasts that arrive during some interval 5.
- When all the broadcasts arrive, an algorithm is run to compute a new time from them.
- The simplest algorithm is just to average the values from all the other machines.
- A slight variation on this theme is first to discard the m highest and m lowest values, and average the rest.
- Discarding the extreme values can be regarded as self-defense against up to m faulty clocks sending out nonsense.
- Another variation is to try to correct each message by adding to it an estimate of the propagation time from the source.
- This estimate can be made from the known topology of the network, or by timing how long it takes for probe messages to be enclosed.

Local averaging algorithm

- This algorithm attempt to overcome the drawbacks of the global averaging algorithm.
- Nodes of a distributed system are logically arranged in some kind of pattern such as Ring.
- Periodically each node exchanges its clock time with his neighbors in the ring.
- Then its sets clock time by taking Average.
- Average is calculated by, its own clock time and clock times of its neighbors.

7) What is Mutual Exclusion?

- Mutual Exclusion is a process that prevents multiple threads or processes from accessing shared resources at the same time.
- Concurrent access of processes to a shared resource or data is executed in mutually exclusive manner.
- Only one process is allowed to execute the critical section (CS) at any given time.
- A critical section is a section in a program that accesses shared resources.
- In a distributed system, shared variables or a local kernel cannot be used to implement mutual exclusion.
- Message passing is the sole means for implementing distributed mutual exclusion.

8) Explain Centralized Mutual Exclusion algorithm.

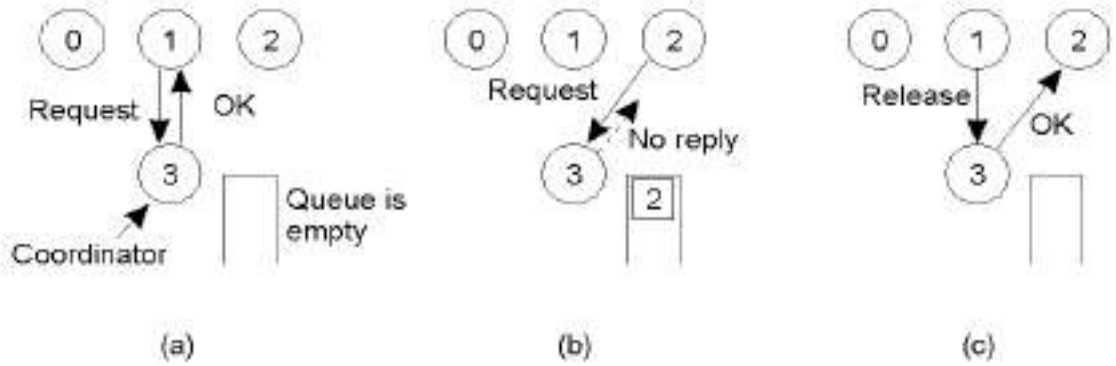


Figure: (a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted. (b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply. (c) When process 1 exits the critical region, it tells the coordinator, which then replies to 2.

- The most straight forward way to achieve mutual exclusion in a distributed system is to simulate how it is done in a one-processor system.
- One process is elected as the coordinator (e.g., the one running on the machine with the highest network address).
- Whenever a process wants to enter a critical region, it sends a request message to the coordinator stating which critical region it wants to enter and asking for permission.
- If no other process is currently in that critical region, the coordinator sends back a reply granting permission.
- When the reply arrives, the requesting process enters the critical region.
- Now suppose that another process 2 in figure asks for permission to enter the same critical region.
- The coordinator knows that a different process is already in the critical region, so it cannot grant permission.
- The exact method used to deny permission is system dependent.
- In Fig.(b), the coordinator just refrains from replying, thus blocking process 2, which is waiting for a reply.
- Alternatively, it could send a reply saying ‘‘permission denied.’’ Either way, it queues the request from 2 for the time being and waits for more messages.
- When process 1 exits the critical region, it sends a message to the coordinator releasing its exclusive access, as shown in Fig.(c).
- The coordinator takes the first item off the queue of deferred requests and sends that process a grant message.
- If the process was still blocked (i.e., this is the first message to it), it unblocks and enters the critical region.
- If an explicit message has already been sent denying permission, the process will have to poll for incoming traffic or block later.
- Either way, when it sees the grant, it can enter the critical region.

9) Explain Distributed Mutual Exclusion algorithm.

- When a process wants to enter a critical region, it builds a message containing the name of the critical region it wants to enter, its process number, and the current time.
- It then sends the message to all other processes, conceptually including itself.
- The sending of messages is assumed to be reliable; that is, every message is acknowledged.
- Reliable group communication if available, can be used instead of individual messages.
- When a process receives a request message from another process, the action it takes depends on its state with respect to the critical region named in the message.
- Three cases have to be distinguished:
    - If the receiver is not in the critical region and does not want to enter it, it sends back an OK message to the sender.
    - If the receiver is already in the critical region, it does not reply. Instead, it queues the request.
    - If the receiver wants to enter the critical region but has not yet done so, it compares the timestamp in the incoming message with the one contained in the message that it has sent everyone.
    - The lowest one wins. If the incoming message is lower, the receiver sends back an OK message.
    - If its own message has a lower timestamp, the receiver queues the incoming request and sends nothing.

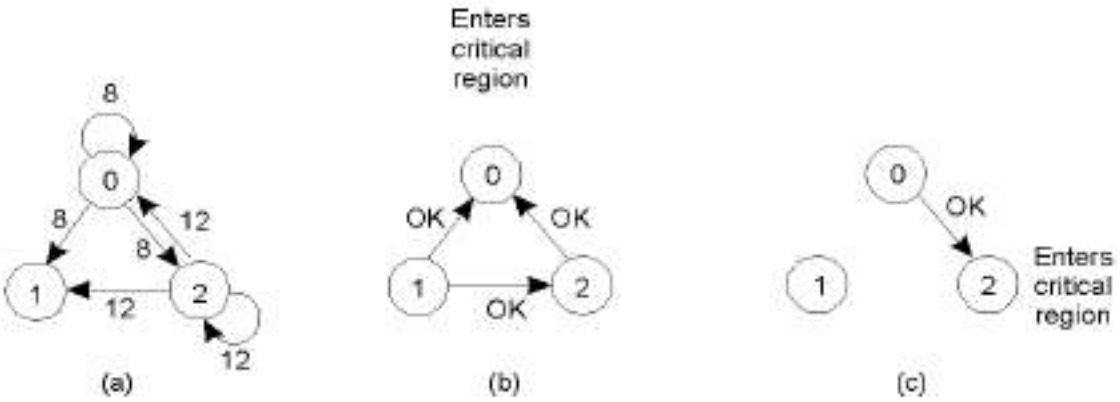


Figure (a) Two processes want to enter the same critical region at the same moment. (b) Process 0 has the lowest timestamp, so it wins. (c) When process 0 is done, it sends an OK also, so 2 can now enter the critical region.

- Process 0 sends everyone a request with timestamp 8.
- while at the same time, process 2 sends everyone a request with timestamp 12.
- Process 1 is not interested in entering the critical region, so it sends OK to both senders.
- Processes 0 and 2 both see the conflict and compare timestamps.
- Process 2 sees that it has lost, so it grants permission to 0 by sending OK.
- Process 0 now queues the request from 2 for later processing and enters the critical region.
- When it is finished, it removes the request from 2 from its queue and sends an OK message to process 2, allowing the latter to enter its critical region.

10) Explain Token Ring Algorithm in Mutual Exclusion.

- Here 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).
- The ring positions may be allocated in numerical order of network addresses or some other means.
- It does not matter what the ordering is. All that matters is that each process knows who is next in line after itself.

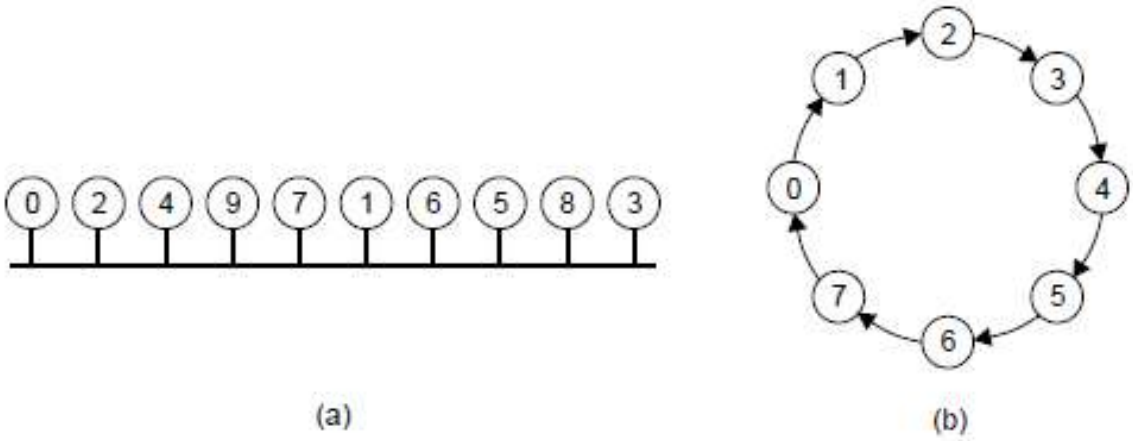


Figure (a) An unordered group of processes on a network. (b) A logical ring constructed in software.

- 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 neighbor 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.

11) What is Deadlock?

Deadlock

- A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.
- Suppose process D holds resource T and process C holds resource U.
- Now process D requests resource U and process C requests resource T but none of process will get this resource because these resources are already hold by other process so both can be blocked, with neither one be able to proceed, this situation is called deadlock.
- Example:

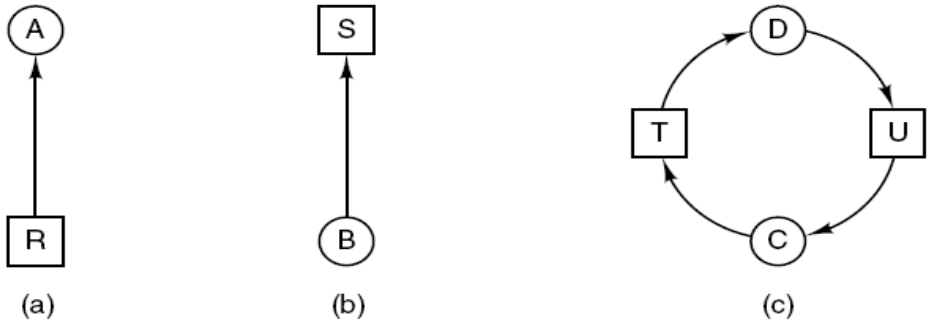


Figure 4-1 Resource allocation graphs. (a) Holding a resource. (b) Requesting a resource. (c) Deadlock.

- As shown in figure 4-1, resource T assigned to process D and resource U is assigned to process C.
- Process D is requesting / waiting for resource U and process C is requesting / waiting for resource T.
- Processes C and D are in deadlock over resources T and U.

12) List the conditions that lead to deadlock.

Conditions that lead to deadlock

- There are four conditions that must hold for deadlock:

1) Mutual exclusion condition
   - Each resource is either currently assigned to exactly one process or is available.
2) Hold and wait condition
   - Process currently holding resources granted earlier can request more resources.
3) No preemption condition
   - Previously granted resources cannot be forcibly taken away from process.
4) Circular wait condition
   - There must be a circular chain of 2 or more processes. Each process is waiting for resource that is held by next member of the chain.

- **All four of these conditions must be present for a deadlock to occur.**

13) Explain deadlock detection and recovery.

Deadlock detection with single resource of each type.

- Algorithm for detecting deadlock for single resource
  1. For each node, N in the graph, perform the following five steps with N as the starting node.
  2. Initialize L to the empty list, designate all arcs as unmarked.
  3. Add current node to end of L, check to see if node now appears in L two times. If it does, graph contains a cycle (listed in L), algorithm terminates.
  4. From given node, see if any unmarked outgoing arcs. If so, go to step 5; if not, go to step 6.
  5. Pick an unmarked outgoing arc at random and mark it. Then follow it to the new current node and go to step 3.
  6. If this is initial node, graph does not contain any cycles, algorithm terminates. Otherwise, dead end. Remove it, go back to previous node, make that one current node, go to step 3.

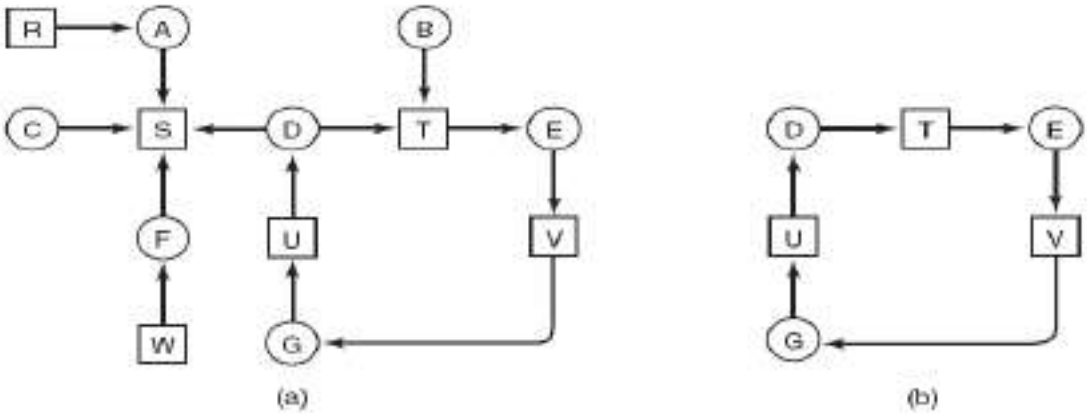


Figure 4-2 (a) A resource graph. (b) A cycle extracted from (a)

- For example as shown in figure 4-2 (a),
  - We are starting from node D.
  - Empty list L = ()
  - Add current node so Empty list = (D).
  - From this node there is one outgoing arc to T so add T to empty list.
  - So Empty list become L = (D, T).
  - Continue this step….so we get empty list as below
  - L = (D, T, E)…… L = (D, T, E, V, G, U, D)
  - In the above step in empty list the node D appears twice, so deadlock.

Deadlock detection for multiple resource

- When multiple copies of some of the resources exist, a matrix-based algorithm can be used for detecting deadlock among n processes.
- Let the number of resource classes be m, with E1 resources of class 1, E2 resources of class 2, and generally, Ei resources of class i (1 < i < m).
- E is the existing resource vector. It gives the total number of instances of each resource in existence.
- Let A be the available resource vector, with Ai giving the number of instances of resource i that are currently available (i.e., unassigned).

- There are two matrices used in the algorithm. C is a current allocation matrix, and R, the request matrix.

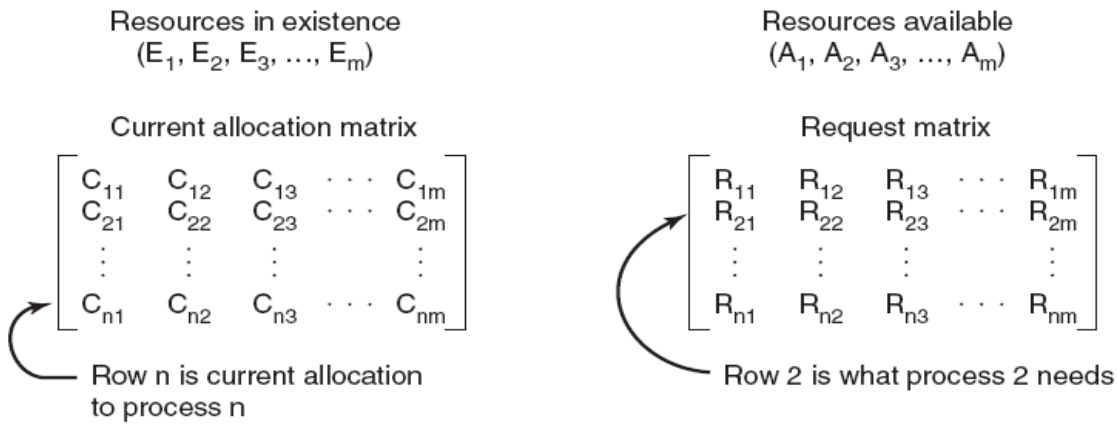


Figure 4-3. The four data structures needed by the deadlock detection algorithm.

- The i-th row of C tells how many instances of each resource class Pi currently holds.
- Thus Cij is the number of instances of resource j that are held by process i.
- Similarly, Rij is the number of instances of resource j that Pi wants.
- These four data structures are shown in Fig. 4-3.
- The deadlock detection algorithm can now be given, as follows.
    1. Look for an unmarked process, Pi, for which the i-th row of R is less than or equal to A.
    2. If such a process is found, add the i-th row of C to A, mark the process, and go back to step 1.
    3. If no such process exists, the algorithm terminates.

When the algorithm finishes, all the unmarked processes, if any, are deadlocked.

Example of deadlock detection in multiple resource

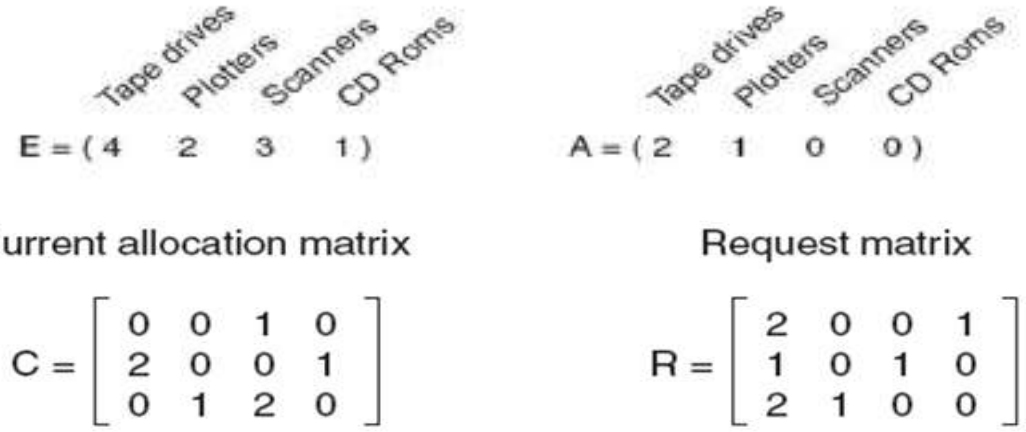


Figure 4-4. An example for the deadlock detection algorithm.

- E is the total no of each resource, C is the number of resources held by each process, A is the number of resources that are available (free), and R is the number of resources still needed by each process to proceed.

- At this moment, we run the algorithm:
- P3 can be satisfied, so P3 completes, returns resources A = (2 2 2 0)
- P2 can be satisfied, so P2 completes, returns resources A = (4 2 2 1)
- P1 can be satisfied, so P1 completes.
- Now all processes are marked, so no deadlock.

Deadlock recovery

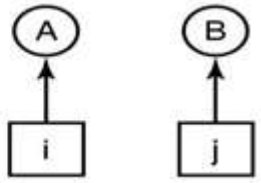
- **Recovery through preemption**
    - In some cases, it may be possible to temporarily take a resource away from its current owner and give it to another process.
    - The ability to take a resource away from a process, have another process use it, and then give it back without the process noticing it is highly dependent on the nature of the resource.
    - Recovering this way is frequently difficult or impossible.
    - Choosing the process to suspend depends largely on which ones have resources that can easily be taken back.
- **Recovery through rollback**
    - Create a checkpoint.
    - Checkpoint a process periodically.
    - Checkpointing a process means that its state is written to a file so that it can be restarted later.
    - The checkpoint contains not only the memory image, but also the resource state, that is, which resources are currently assigned to the process.
    - When a deadlock is detected, it is easy to see which resources are needed.
    - To do the recovery, a process that owns a needed resource is rolled back to a point in time before it acquired some other resource by starting one of its earlier checkpoints.
    - In effect, the process is reset to an earlier moment when it did not have the resource, which is now assigned to one of the deadlocked processes.
    - If the restarted process tries to acquire the resource again, it will have to wait until it becomes available.
- **Recovery through killing processes**
    - The crudest, but simplest way to break a deadlock is to kill one or more processes.
    - One possibility is to kill a process in the cycle. With a little luck, the other processes will be able to continue.
    - If this does not help, it can be repeated until the cycle is broken.
    - Alternatively, a process not in the cycle can be chosen as the victim in order to release its resources.
    - In this approach, the process to be killed is carefully chosen because it is holding resources that some process in the cycle needs.

14) How deadlock can be prevented? OR Explain deadlock prevention.

Deadlock Prevention

- Deadlock can be prevented by attacking the one of the four conditions that leads to deadlock.
  - **1) Attacking the Mutual Exclusion Condition**
    - No deadlock if no resource is ever assigned exclusively to a single process.
    - Some devices can be spooled such as printer, by spooling printer output; several processes can generate output at the same time.
    - Only the printer daemon process uses physical printer.
    - Thus deadlock for printer can be eliminated.
    - Not all devices can be spooled.
    - Principle:
      - Avoid assigning a resource when that is not absolutely necessary.
      - Try to make sure that as few processes as possible actually claim the resource.
  - **2) Attacking the Hold and Wait Condition**
    - Require processes to request all their resources before starting execution.
    - A process is allowed to run if all resources it needed is available. Otherwise nothing will be allocated and it will just wait.
    - Problem with this strategy is that a process may not know required resources at start of run.
    - Resource will not be used optimally.
    - It also ties up resources other processes could be using.
    - Variation: A process must give up all resources before making a new request. Process is then granted all prior resources as well as the new ones only if all required resources are available.
    - Problem: what if someone grabs the resources in the meantime how can the processes save its state?
  - **3) Attacking the No Preemption Condition**
    - This is not a possible option.
    - When a process P0 request some resource R which is held by another process P1 then resource R is forcibly taken away from the process P1 and allocated to P0.
    - Consider a process holds the printer, halfway through its job; taking the printer away from this process without having any ill effect is not possible.
  - **4) Attacking the Circular Wait Condition**
    - To provide a global numbering of all the resources.
    - Now the rule is this: processes can request resources whenever they want to, but all requests must be made in numerical order.
    - A process need not acquire them all at once.
    - Circular wait is prevented if a process holding resource n cannot wait for resource m, if m > n.
    - No way to complete a cycle.

1. Imagesetter
2. Scanner
3. Plotter
4. Tape drive
5. CD Rom drive



- A process may request 1st a CD ROM drive, then tape drive. But it may not request 1st a plotter, then a Tape drive.
- Resource graph can never have cycle.

15) What is Election Algorithms.

- Several distributed algorithms require a coordinator process in the entire System.
- It performs coordination activity needed for the smooth running of processes.
- Two examples of such coordinator processes encountered here are:
  - The coordinator in the centralized algorithm for mutual exclusion.
  - The central coordinator in the centralized deadlock detection algorithm.
- If the coordinator process fails, a new coordinator process must be elected to take up the job of the failed coordinator.
- Election algorithms are meant for electing a coordinator process from the currently running processes.
- Election algorithms are based on the following assumptions:
  - Each process in the system has a unique priority number.
  - Whenever an election is held, the process having the highest priority number among the currently active processes is elected as the coordinator.
  - On recovery, a failed process can take appropriate actions to rejoin the set of active processes.

16) Explain Bully Algorithm for Election.

- Bully algorithm specifies the process with the highest identifier will be the coordinator of the group. It works as follows:
- When a process p detects that the coordinator is not responding to requests, it initiates an election:
  - p sends an election message to all processes with higher numbers.
  - If nobody responds, then p wins and takes over.
  - If one of the processes answers, then p's job is done.
- If a process receives an election message from a lower-numbered process at any time, it:
  - sends an OK message back.
  - holds an election (unless its already holding one).
- A process announces its victory by sending all processes a message telling them that it is the new coordinator.
- If a process that has been down recovers, it holds an election.

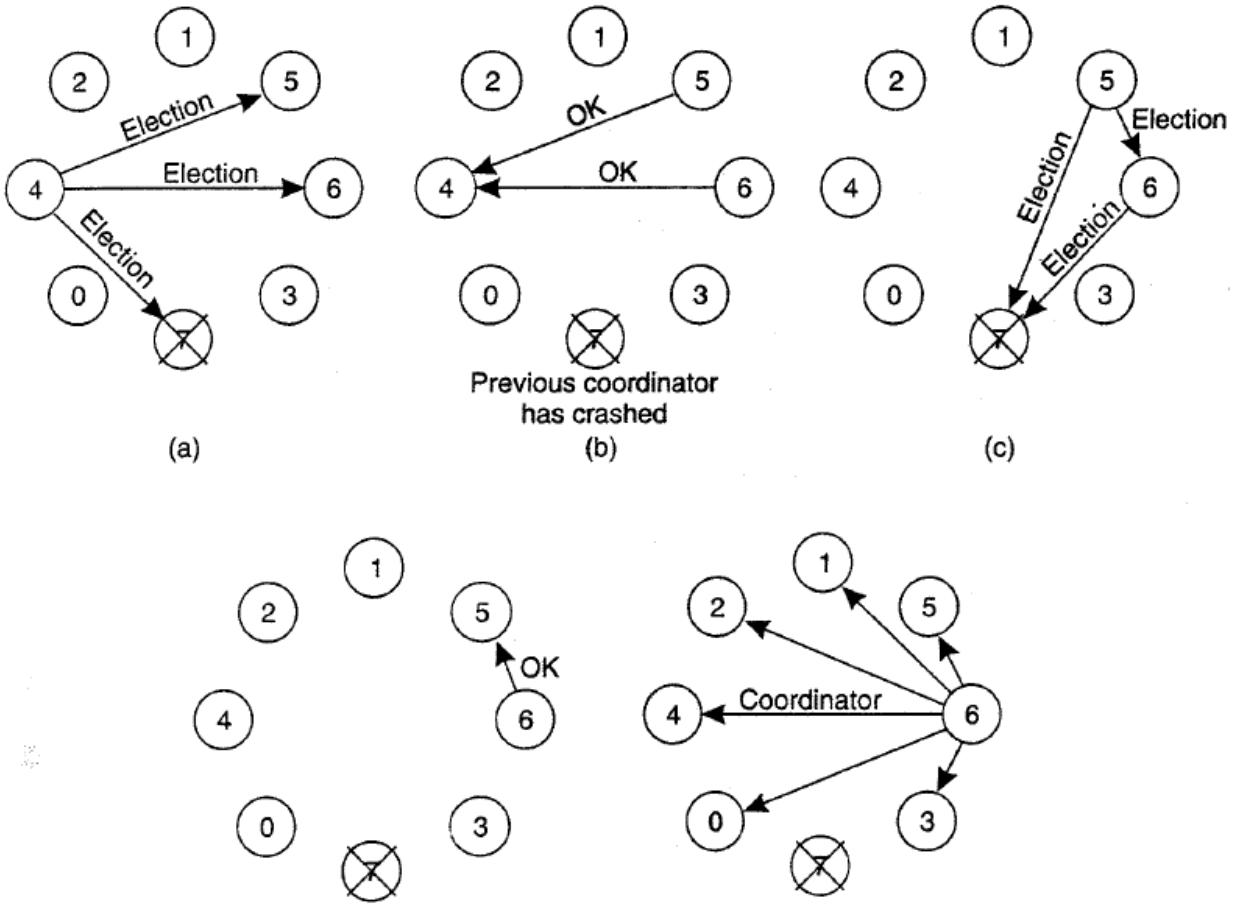


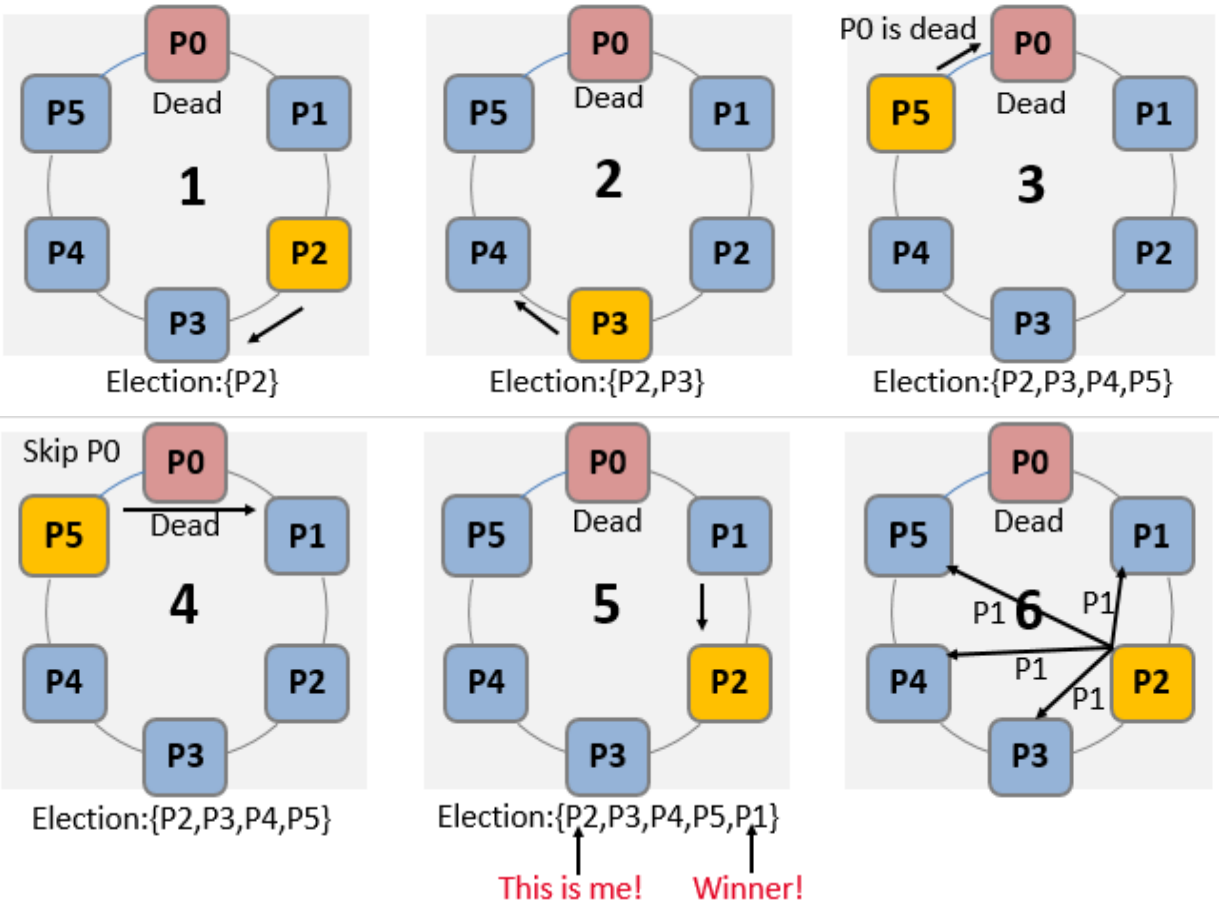
Figure: Bully Algorithm. (a) Process 4 holds an election. (b) Processes 5 and 6 respond, telling 4 to stop. (c) Now 5 and 6 each hold an election. (d) Process 6 tells 5 to stop. (e) Process 6 wins and tells everyone.

- The group consists of eight processes, numbered from 0 to 7.
- Previously process 7 was the coordinator, but it has just crashed.
- Process 4 is the first one to notice this, so it sends ELECTION messages to all the processes higher than it, namely 5, 6, and 7, as shown in Figure (a).
- Processes 5 and 6 both respond with OK, as shown in Figure (b).
- Upon getting the first of these responses, 4 knows that its job is over.
- It knows that one of these bigwigs will take over and become coordinator.
- It just sits back and waits to see who the winner will be.
- In Figure (c), both 5 and 6 hold elections, each one only sending messages to those processes higher than itself.
- In Figure (d) process 6 tells 5 that it will take over.
- At this point 6 knows that 7 is dead and that it (6) is the winner.
- If there is state information to be collected from disk or elsewhere to pick up where the old coordinator left off, 6 must now do what is needed.
- When it is ready to take over, 6 announces this by sending a COORDINATOR message to all running processes.

- When 4 gets this message, it can now continue with the operation it was trying to do when it discovered that 7 was dead, but using 6 as the coordinator this time.
- In this way the failure of 7 is handled and the work can continue.
- If process 7 is ever restarted, it will just send all the others a COORDINATOR message and bully them into submission.

17) Explain Ring Election Algorithm.

- The ring algorithm assumes that the processes are arranged in a logical ring and each process is knowing the order of the ring of processes.
- If any process detects failure, it constructs an election message with its process ID (e.g., its network address and local process ID) and sends it to its neighbor.
- If the neighbor is down, the process skips over it and sends the message to the next process in the ring.
- This process is repeated until a running process is located.
- At each step, the process adds its own process ID to the list in the message and sends the message to its living neighbor.
- Eventually, the election message comes back to the process that started it.
- The process then picks either the highest or lowest process ID in the list and sends out a message to the group informing them of the new coordinator.



- Fig.1 shows a ring of six processes. P2 detects that the coordinator P0 is dead.
- It starts an election by sending an election message containing its process ID to its neighbor P3. (Fig.1)
- P3 receives an election message and sends an election message to its neighbor with the ID of P3 suffixed to the list. (Fig.2)
- The same sequence of events occurs with P4, which adds its ID to the list it received from P3 and sends an election message to P5.
- P5 then tries to send an election message to P0. (Fig. 3)
- P0 is dead and the message is not delivered, P5 tries again to its neighbor P1. (Fig. 4)
- The message is received by P2, the originator of the election. (Fig. 5)
- P2 recognizes that it is the initiator of the election because its ID is the first in the list of processes.
- It then picks a leader. it chooses the lowest-numbered process ID, which is that of P1.
- It then informs the rest of the group of the new coordinator (Fig. 6).

1) What is thread? Explain thread structure. OR Explain thread in brief with its structure.

Thread

- A program has one or more locus of execution. Each execution is called a thread of execution.
- In traditional operating systems, each process has an address space and a single thread of execution.
- It is the smallest unit of processing that can be scheduled by an operating system.
- A thread is a single sequence stream within in a process. Because threads have some of the properties of processes, they are sometimes called lightweight processes. In a process, threads allow multiple executions of streams.

Thread Structure

- Process is used to group resources together and threads are the entities scheduled for execution on the CPU.
- The thread has a program counter that keeps track of which instruction to execute next.
- It has registers, which holds its current working variables.
- It has a stack, which contains the execution history, with one frame for each procedure called but not yet returned from.
- Although a thread must execute in some process, the thread and its process are different concepts and can be treated separately.
- What threads add to the process model is to allow multiple executions to take place in the same process environment, to a large degree independent of one another.
- Having multiple threads running in parallel in one process is similar to having multiple processes running in parallel in one computer.

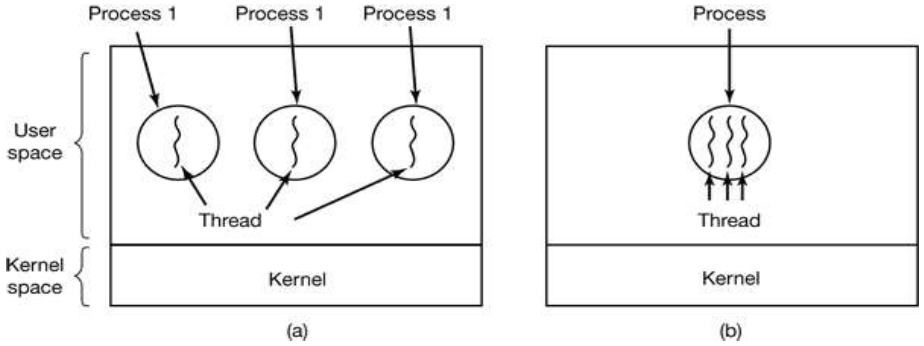


Figure: (a) Three processes each with one thread. (b) One process with three threads.

- In former case, the threads share an address space, open files, and other resources.
- In the latter case, process share physical memory, disks, printers and other resources.
- In Fig.(a) we see three traditional processes. Each process has its own address space and a single thread of control.
- In contrast, in Fig.(b) we see a single process with three threads of control.
- Although in both cases we have three threads, in Fig.(a) each of them operates in a different address space, whereas in Fig.(b) all three of them share the same address space.
- Like a traditional process (i.e., a process with only one thread), a thread can be in any one of several states: running, blocked, ready, or terminated.

- When multithreading is present, processes normally start with a single thread present. This thread has the ability to create new threads by calling a library procedure **thread_create.**
- When a thread has finished its work, it can exit by calling a library procedure **thread_exit.**
- One thread can wait for a (specific) thread to exit by calling a procedure **thread_join.** This procedure blocks the calling thread until a (specific) thread has exited.
- Another common thread call is **thread_yield**, which allows a thread to voluntarily give up the CPU to let another thread run.

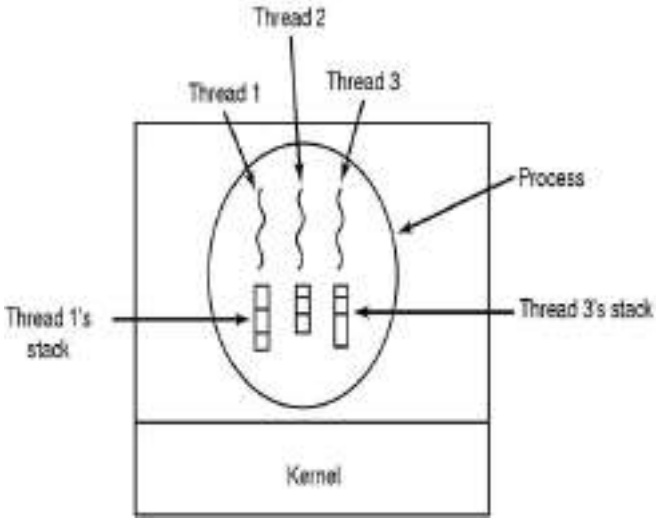


Figure: Each thread has its own stack.

2) What are the Similarities and Dissimilarities between Process and Thread?

Similarities between Process and Thread

- Like processes threads share CPU and only one thread is running at a time.
- Like processes threads within a process execute sequentially.
- Like processes thread can create children.
- Like a traditional process, a thread can be in any one of several states: running, blocked, ready or terminated.
- Like process threads have Program Counter, Stack, Registers and State.

Dissimilarities between Process and Thread

- Unlike processes threads are not independent of one another.
- Threads within the same process share an address space.
- Unlike processes all threads can access every address in the task.
- Unlike processes threads are design to assist one other. Note that processes might or might not assist one another because processes may be originated from different users.

3) Give difference between User level threads and Kernel level threads.

USER LEVEL THREAD	KERNEL LEVEL THREAD
User thread are implemented by users.	Kernel threads are implemented by OS.
OS doesn't recognized user level threads.	Kernel threads are recognized by OS.
Implementation of User threads is easy.	Implementation of Kernel thread is complex.
Context switch time is less.	Context switch time is more.
Context switch requires no hardware support.	Context switch requires hardware support.
If one user level thread performs blocking operation, then entire process will be blocked.	If one kernel thread performs blocking operation, then another thread with in same process can continue execution.
Example: Java thread, POSIX threads.	Example: Window Solaris

4) Explain Thread models. OR Explain type of Thread Usage.

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.

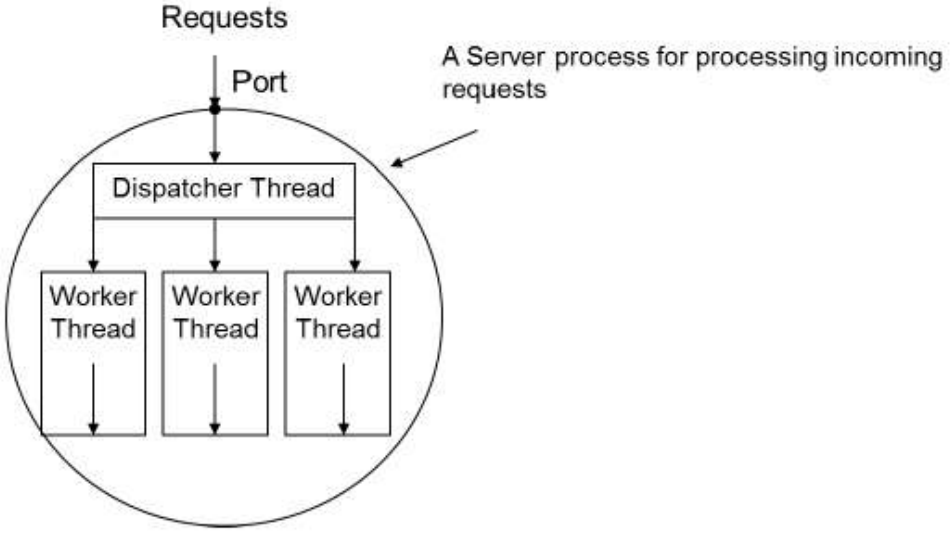


Figure: Dispatcher worker model

Team model

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

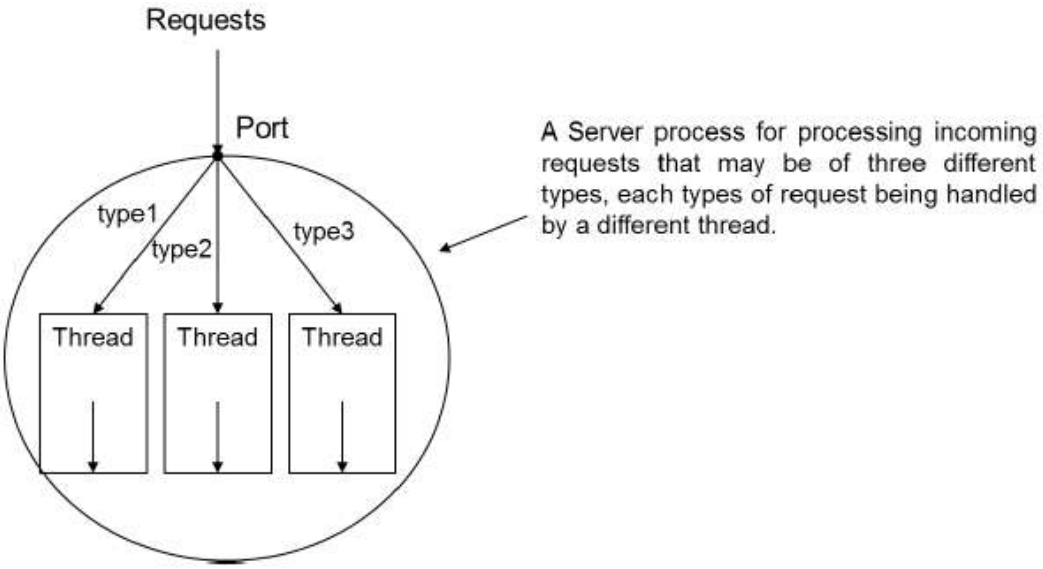


Figure: Team model

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.

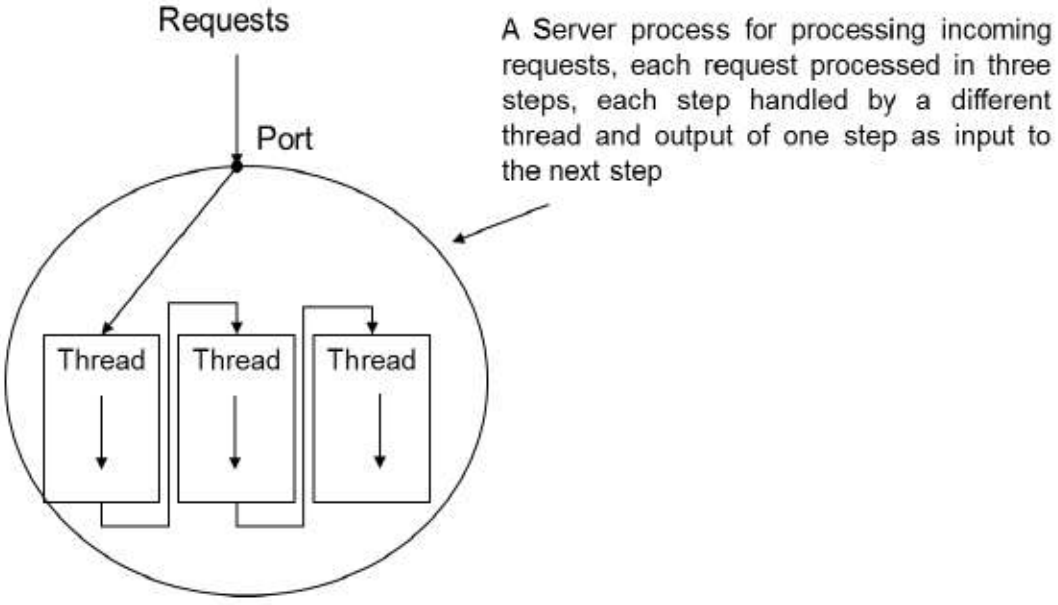


Figure: Pipeline model

5) Explain Designing issues in Thread package.

- A system that supports thread facility must provide a set of primitives to its users for threads-related operations.
- These primitives of the system are said to form a thread package.
- Some of the important issues in designing a thread package are:
  1. Thread Creation
  2. Thread Termination
  3. Thread Synchronization
  4. Thread Scheduling
  5. Signal Handling

Thread Creation

- Threads can be created either statically or dynamically.
  1. **Static:**
     - The number of threads of a process remains fixed for its entire lifetime.
     - Memory space is allocating to each thread.
  2. **Dynamic:**
     - The number of threads of a process keeps changing dynamically.
     - Threads are created as and when it is needed during the process life cycle.
     - It exits when task is completed.
     - Here the stack size for the threads is specified as parameter to the system call for thread creation.

Thread Termination

- Threads may terminate or never terminate until life cycle of process.
- Thread termination can be done as follows:
  - Thread destroys itself on task completion by making an EXIT call.
  - Thread is killed from outside using KILL command with thread identifier as parameter.
- In some process, all its threads are created immediately after the process start and then these threads are never killed until the process terminates.

Thread Synchronization

- Since threads belongs to same process share the same address space, thread synchronization is required to ensure that multiple threads don't access the same data simultaneously.
- For example:
  - If two threads want to increment the same global variable with in the same process.
  - One thread should exclusive access to shared variable, increment it, and then pass control to the other thread.
  - It means that only one thread can execute in critical region at any instance of time.

Thread Scheduling

- Another important issue in designing threads package is to decide an appropriate scheduling algorithm.
- Thread packages provide calls to give the users the flexibility to specify the scheduling policy to be used for their applications.
- Some of the special features for threads scheduling that may be supported by a threads package are as follows:
    - Priority assignment facility
    - Flexibility to vary quantum size dynamically
    - Handoff scheduling
    - Affinity scheduling

Signal Handling

- Signals provide software-generated interrupts and exceptions.
- Interrupts are externally generated disruptions of a thread or process.
- Exceptions are caused by the occurrence of unusual conditions 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 when another signal of the same type occurs in some other thread before the first one is handled by the thread in which it occurred.

6) What is Scheduling in distributed systems? Explain Desirable features of a good Global scheduling algorithm.

- A resource manager schedules the processes in a distributed system to make use of the system resources.
- Scheduling is to optimize resource usage, response time, network congestion. It can be broadly classified into three types:

- **Task assignment approach**
    - In which each process is viewed as a collection of related tasks.
    - These tasks are scheduled to suitable nodes so as to improve performance.

- **Load-balancing approach**
    - In which all the processes submitted by the users are distributed among the nodes of the system to equalize the workload among the nodes.

- **Load-sharing approach**
    - Which simply attempts to conserve the ability of the system to perform work by assuring that no node is idle while processes wait for being processed.

Desirable Features of a Good Global Scheduling Algorithm:

1. **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.
- Obtaining prior knowledge poses an extra burden upon the users who must specify this information while submitting their processes for execution.

2. **Dynamic in Nature**

- Process assignment decisions should be based on the current load of the system and not on some fixed static policy.
- Thus system support pre-emptive process migration facility in which a process can be migrated from one node to another during the course of its execution.

3. **Quick Decision making capability**

- A good process scheduling algorithm must make quick decisions about the assignment of processes to processors.
- For example, an algorithm that models the system by a mathematical program and solves it on line is unsuitable because it does not meet this requirement.
- Heuristic methods requiring less computational effort while providing near optimal results are therefore normally preferable to exhaustive solution methods.

4. **Balance System Performance and Scheduling Overhead**

- Several global scheduling algorithms collect global state information and use this information in making process assignment decisions.
- A common intuition is that greater amounts of information describing global system state allow more intelligent process assignment decisions to be made that have a positive effect on the system as a whole.
- In a distributed environment, however, information regarding the state of the system is typically gathered at a higher cost than in a centralized system.
- Hence algorithms that provide near optimal system performance with a minimum of global state information gathering overhead are desirable.

5. **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.
- Processor thrashing can occur in situations where each node of the system has the power of scheduling its own processes and scheduling decisions are based on relatively old data to transmission delay between nodes.

- For example, it may happen that node n1 and n2 both observe that node n3 is idle and then both offload a portion of their work to node n3 without being aware of the offloading decision made by the other.
- Now if node n3 becomes overloaded due to the processes received from both nodes n1 and n2, then it may again start transferring its processes to other nodes.
- This entire cycle may be repeated again and again, resulting in an unstable state.

6. Scalable

- Algorithm should be scalable and able to handle workload inquiry from any number of machines in the network.
- The N2 nature of the algorithm creates more network traffic and quickly consumes network bandwidth.
- A simple approach to make an algorithm scalable is to probe only m of N nodes for selecting a host.
- The value of m can be dynamically adjusted depending upon the value of N.

7. 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.

8. 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.

7) What is Processor Allocation OR Task Assignment in Distributed system? Explain Process allocation OR Task assignment algorithms.

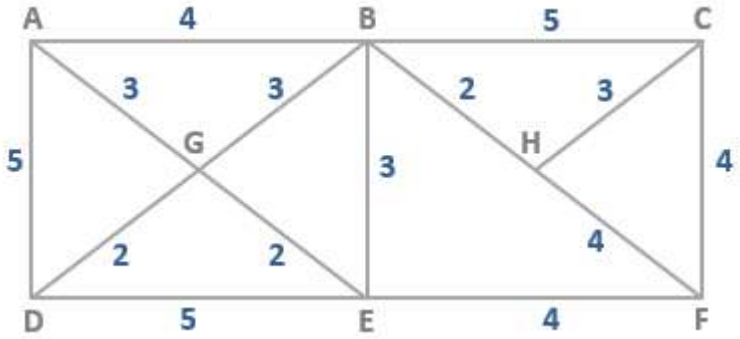
- Each process is divided into multiple tasks.
- These tasks are scheduled to suitable processor to improve performance.
- It requires characteristics of all the processes to be known in advance.
- This approach does not take into consideration the dynamically changing state of the system.
- In this approach, a process is considered to be composed of multiple tasks and the goal is to find an optimal assignment policy for the tasks of an individual process.

Task Assignment Approach algorithms

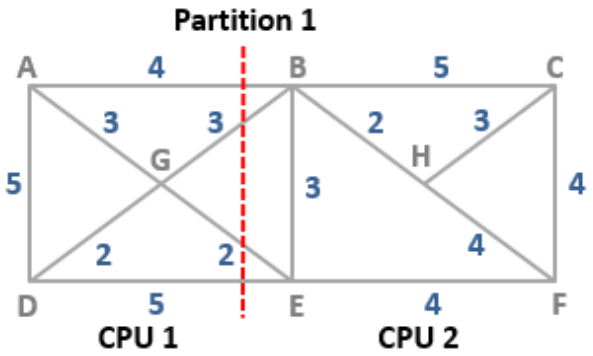
1. Graph Theoretic Deterministic Algorithm

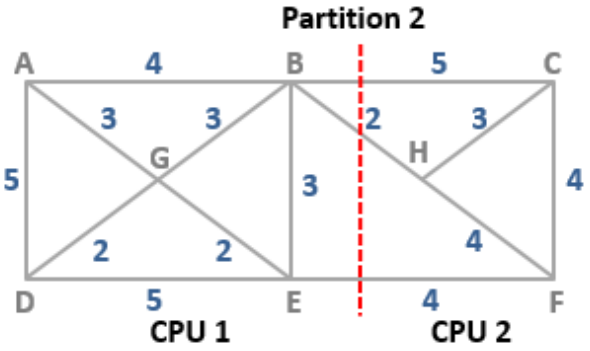
- A system with m CPUs and n processes has any of the following three cases:
- m=n: Each process is allocated to one CPU.
- M>n: Some CPUs may remain idle (free) or work on earlier allocated processes.

- M<n: There is a need to schedule processes on CPUs, and several processes may be assigned to each CPU.
- The main objective of performing CPU assignment is to:
  - Minimize IPC cost.
  - Obtain quick turnaround time.
  - Achieve high degree of parallelism for efficient utilization.
  - Minimize network traffic.



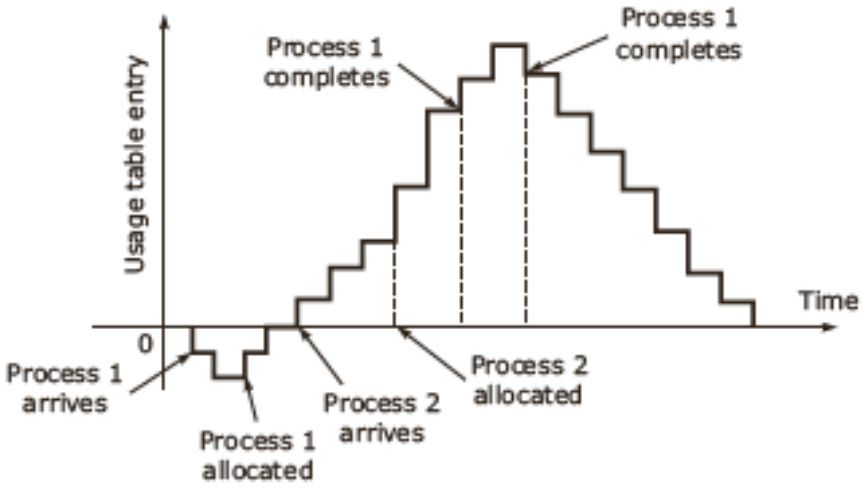
- Processes are represented as nodes A, B, C, D, E, F, G and H.
- Arcs between sub-graphs represent network traffic and their weights represent IPC costs.
- Total network traffic is the sum of the arcs intersected by the dotted cut lines.





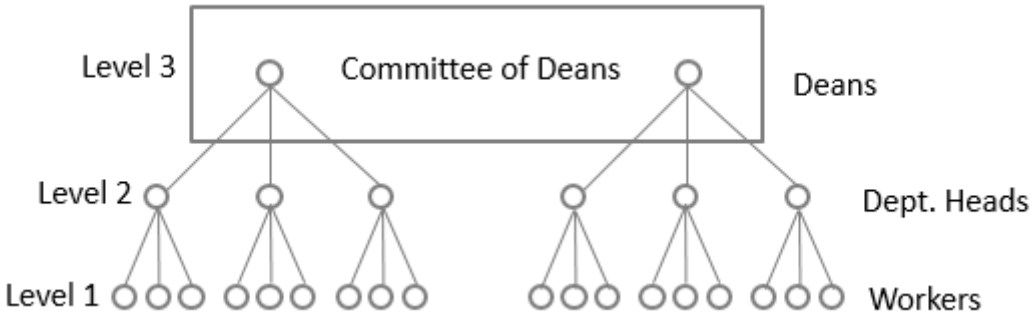
- Partition 1
  - CPU 1 runs A, D, G
  - CPU 2 runs B, E, F, H and C
  - Network traffic= 4+3+2+5=14
- Partition 2
  - CPU 1 runs processes A, D, E, G, B
  - CPU2 runs processes H, C and F
  - Network traffic= 5+2+4=11
- Thus partition 2 communication generates less network traffic as compared to partition 1.

Centralized Heuristic Algorithm



- It is also called Top down algorithm.
- It doesn't require advance information.
- Coordinator maintains the usage table with one entry for every user (processor) and this is initially zero.
- Usage table entries can either be zero, positive, or negative.
    - Zero value indicates a neutral state.
    - Positive value implies that the machine is using system resources.
    - Negative value means that the machine needs resources.

Hierarchical Algorithm



- Process hierarchy is modelled like an organization hierarchy.
- For each group of workers, one manager machine (department head) is assigned the task of keeping track of who is busy and who is idle.
- If the system is large, there will be number of department heads, so some machines will function as "deans".
- Each processor maintains communication with one superior and few subordinates.
- When a dean or department head stops functioning (crashes), promote one of the direct subordinates of the faulty manager to fill in for the boss.
- The choice of which can be made by the subordinates themselves.

8) Classify Load balancing approaches and Explain each in detail. Write advantages of Load balancing.

- The distribution of loads to the processing elements is simply called the load balancing.
- Load balancing approaches can be classified as follows:

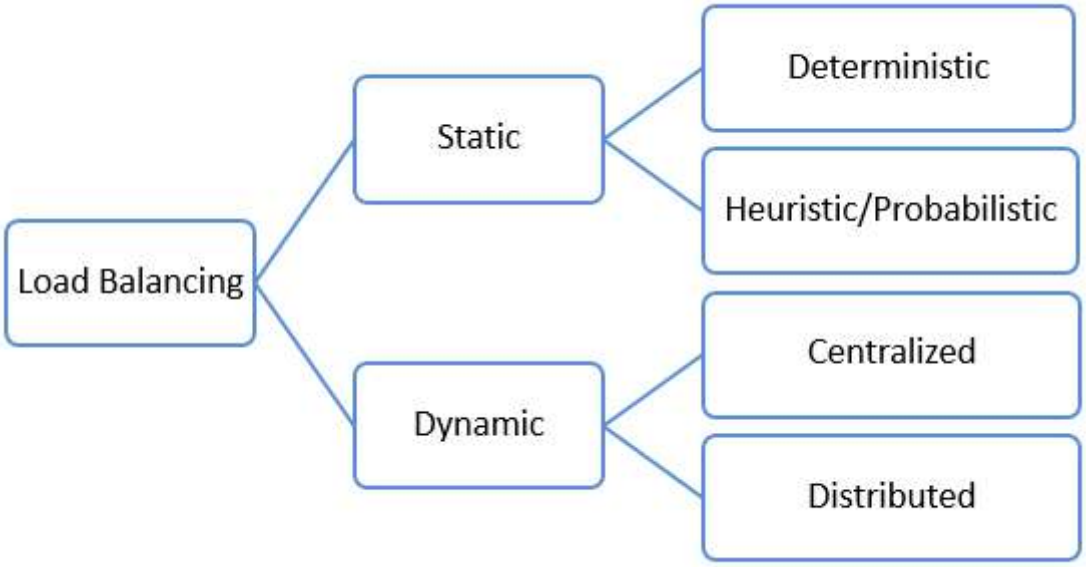


Figure: Classification of Load Balancing approaches

Static Load Balancing

- In static algorithm the processes are assigned to the processors at the compile time according to the performance of the nodes.
- Once the processes are assigned, no change or reassignment is possible at the run time.
- Number of jobs in each node is fixed in static load balancing algorithm.
- Static algorithms do not collect any information about the nodes.
- The static load balancing algorithms can be divided into two sub classes:
    1. **Optimal Static Load Balancing Algorithm**
        - If all the information and resources related to a system are known optimal static load balancing can be done.
    2. **Sub optimal static load balancing Algorithm**
        - Sub-optimal load balancing algorithm will be mandatory for some applications when optimal solution is not found.

Dynamic Load Balancing

- In dynamic load balancing algorithm assignment of jobs is done at the runtime.
- In DLB jobs are reassigned at the runtime depending upon the situation.
- The load will be transferred from heavily loaded nodes to the lightly loaded nodes.
- No decision is taken until the process gets executed.
- This strategy collects the information about the system state and about the job information.
- As more information is collected, the algorithm can make better decision.

Deterministic vs Probabilistic

- 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 Probabilistic/heuristic processor allocation is preferred.

Centralized Vs Distributed

CENTRALIZED LOAD BALANCING	DISTRIBUTED LOAD BALANCING
Scheduling decision is carried out at one single node called the centralized node.	Scheduling decision is carried out at different nodes called the Distributed nodes.
Every information is available at a single node.	Information is distributed at different nodes.
Centralized approach leads to a bottleneck as number of requests increases.	Distributed approach handle the multiple requests by physically distributing among various nodes.
If centralized server fails, all scheduling in the system will fails.	If distributed server fails, all scheduling will be handled by other servers.

Advantages of Load balancing

- Load balancing improves the performance of each node and hence the overall system performance will improve.
- Load balancing reduces the job idle time.
- Low cost but high gain.
- Small jobs do not suffer from long starvation.
- Extensibility and incremental growth.
- Maximum utilization of resources.
- Higher throughput.
- Response time becomes shorter.

9) Explain issues in Designing Load Balancing Algorithms.

- **Load estimation:** determines how to estimate the workload of a node in a distributed system.
- **Process transfer:** decides whether the process can be executed locally or remotely.
- **Static information exchange:** determines how the system load information can be exchanged among the nodes.
- **Location policy:** determines the selection of a destination node during process migration.
- **Priority assignment:** determines the priority of execution of a set of local and remote processes on a particular node.

- **Migration limiting policy:** determines the total number of times a process can migrate from one node to another.

10) What is Load sharing in Distributed system? Explain Location policies in Designing Load Sharing Algorithms.

- Load sharing algorithms ensure that no node is idle or heavily loaded.
- Policies for load sharing approach are the same as load balancing polices.
- They include load estimation policy, process transfer policy, location policy and state information exchange.
- They differ in location policy.

Location Policies

- 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.
- Depending on the type of node that takes the initiative to globally search for a suitable node for the process, the location policies are of the following types:

1. **Sender-initiated policy:**
    - In which the sender node of the process decides where to send the process.
    - The heavily loaded nodes search for lightly loaded nodes to which work may be transferred.
    - when a node's load becomes more than the threshold value, it either broadcasts a message or randomly probes the other nodes one by one to find a lightly loaded node that can accept one or more of its processes.
    - If a suitable receiver node is not found, the node on which the process originated must execute that process.

2. **Receiver-initiated policy:**
    - In which the receiver node of the process decides from where to get the process.
    - In this policy lightly loaded nodes search for heavily loaded nodes from which processes can be accepted for execution.
    - When the load on a node falls below a threshold value, it broadcasts a probe message to all nodes or probes nodes one by one to search for a heavily loaded node.
    - Some heavily loaded node may transfer one of its process if such a transfer does not reduce its load below normal threshold.

11) What is process migration? List the steps for it. What are the advantages of process migration?

- Process migration is a specialized form of process management whereby processes are moved from one computing environment to another.
- Process migration is classified into two types:
    - Non-preemptive: Process is migrated before execution start in source node.
    - Preemptive: Process is migrated during its execution.

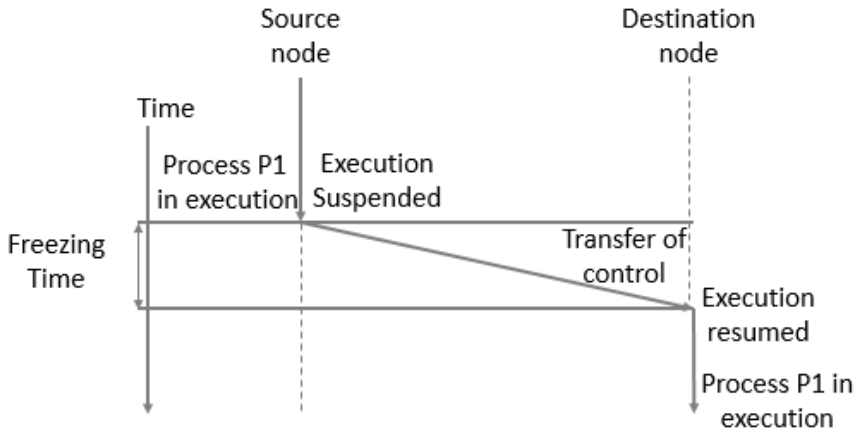


Figure: Flow of execution of a migrating process

- 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

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

2. **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.

3. **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 throughput of the system.

4. **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 (databases, files with different capabilities are optimally used.
   - Depending nature of process, it can be appropriately migrated to utilize the system resource efficiently.

5. **Reducing network traffic**
   - Migrating a process closer to the resource it is using most heavily may reduce network traffic in the system if the decreased cost of accessing its favorite resources offsets the possible increased cost of accessing its less favored ones.
   - Another way to reduce network traffic by process migration is to migrate and cluster two or more processes, which frequently communicate with each other, on the same node of the system.

6. **Improving system reliability**
   - Simply migrate a critical process to a node whose reliability is higher than other nodes in the system.
   - Migrate a copy of a critical process to some other node and to execute both the original and copied processes concurrently on different nodes.
   - In failure modes such as manual shutdown, which manifest themselves as gradual degradation of a node, the process of the node, for their continued execution, may be migrated to another node before the dying node completely fails.

7. **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.

12) What are the desirable features of a good process migration mechanism?

Desirable Features of a Good process migration mechanism:

1. **Transparency**

- Transparency is an important requirement for a system that supports process migration.
- Object access level transparency
  - Object access level transparency is the minimum requirement for a system to support non preemptive 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 inter process communication transparency
  - System call and inter process communication level transparency is must for a system to support preemptive 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.

2. **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.

3. 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 since, otherwise, the following will occur.

4. 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.

5. 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.

6. Communication between coprocesses of a job

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

13) What is Fault Tolerance. Explain classification of Component faults. Explain technique to handle Redundancy.

- The main objective of designing a fault tolerance system is to ensure that the system operates correctly even in presence of faults.
- Distributed system made up of independent computers connected over the network, can fail due to fault in any of hardware components or software components.
- **Component faults can be classified as:**

1. Transient faults

- It can occur suddenly, disappear, and may not occur again if the operation is repeated.
- For example:
    - During heavy traffic in network, a telephone call is misroute but if call is retried, it will reach destination correctly.

2. Intermittent faults

- These faults occur often, but may not be periodic in nature.
- For example:
    - The loose connection to network switch may cause intermittent connection problems.
    - Such faults are difficult to diagnose.
    - During debugging process, it is possible that the fault may not be detected at all.

3. Permanent faults

- These faults can be easily identified and the components can be replaced.
- For example:
    - Software bug or disk head crash causes permanent faults.
    - These faults can be identified and corresponding action can be taken.

- Technique to handle fault tolerance is **Redundancy**.
- Types of Redundancy are as follows:

1. Information redundancy
- Extra bits are added to data to handle fault tolerance by detecting errors.

2. Time redundancy
- An action performed once is repeated if needed after a specific time period.
- For example, if an atomic transaction aborts, it can be executed without any side effects.
- Time redundancy is helpful when the faults are transient or intermittent.

3. Physical redundancy
- Extra equipment's are added to enable the system to tolerate faults due to loss or malfunction of some components.
- For example, extra stand by processors can be used in the system.

14) What is Real time distributed system? Explain the types of Real time distributed system.

- It is developed by collection of computers in a network.
- Some of these are connected to external devices that produce or accept data or expect to be controlled in real time.
- Computers may be tiny microcontrollers or stand-alone machines.
- In both cases they usually have sensors for receiving signals from the devices and actuators for sending signals to them.
- The sensors and actuators may be digital or analog.

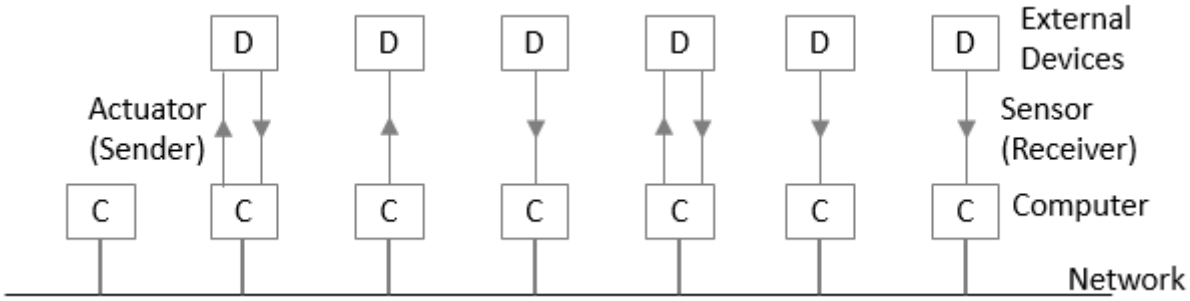


Figure: A Distributed real-time computer system

- Real-time systems are generally split into two types depending on how serious their deadlines are and the consequences of missing one.
  1. **Soft real time system:** missing an occasional deadline is all right. For example, a telephone switch might be permitted to lose or misroute one call in 10^5 under overload conditions and still be within specification.
  2. **Hard real time system:** even a single missed deadline in a hard real-time system is unacceptable, as this might lead to loss of life or an environmental catastrophe

1) What is Distributed file system? Explain Benefits and Services of distributed file system.

- 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 (DFS) is a file system with data stored on a server.
- The data is accessed and processed as if it was stored on the local client machine.
- The DFS makes it convenient to share information and files among users on a network in a controlled and authorized way.
- Server allows the client users to share files and store data just like they are storing the information locally.
- Servers have full control over the data and give access control to the clients.

Benefits of distributed file system

1. **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.
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. **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.
4. **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.

Services of distributed file system

1. **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.
   - It is also known as disk service and block service.
2. **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.
3. **Name/Directory service**
   - It provides mapping between text names for files to their IDs.
   - 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.

2) Explain feature and goals of distributed file system.

1. Transparency

- Structure transparency
  - Clients should not know the number or locations of file servers and the storage devices.
- Access transparency
  - Both local and remote files should be accessible in the same way.
  - The file system should automatically locate an accessed files and arrange for the transport of data to the client's site.
- Naming transparency
  - The name of a file should give no hint as to where the file is located.
  - The name of the file must not be changed when moving from one node to another.
- Replication transparency
  - If a file is replicated on multiple nodes, both the existence of multiple copies and their locations should be hidden from the clients.

2. User mobility

- The user should not be forced to work on a specific node but should have the flexibility to work on different nodes at different times.
- This can be achieved by automatically bringing the users environment to the node where the user logs in.

3. Performance

- Performance is measured as the average amount of time needed to satisfy client requests.
- This time includes CPU time + time for accessing secondary storage + network access time.
- It is desirable that the performance of a distributed file system should be comparable to that of a centralized file system.

4. Simplicity and ease of use

- User interface to the file system must be simple and the number of commands should be as small as possible.

5. Scalability

- A good distributed file system should be designed to easily cope with the growth of nodes and users in the system.
- Such growth should not cause serious disruption of service or significant loss of performance to users.

6. High availability

- A distributed file system should continue to function even when partial failures occur due to the failure of one or more components.
- It should have multiple and independent file servers controlling multiple and independent storage devices.

7. High reliability

- The probability of loss of stored data should be minimized.
- System should automatically generate backup copies of critical files that can be used in the event of loss of the original ones.

8. Data integrity

- Concurrent access requests from multiple users must be properly synchronized by the use of some concurrency control mechanism.
- Atomic transactions are a high-level concurrency control mechanism provided to users by a file system for data integrity.

9. Security

- Users should be confident about the privacy of their data.
- Necessary security mechanisms must be implemented against unauthorized access of files.

10. Heterogeneity

- There should be easy access to shared data on diverse platforms (e.g. Unix workstation, Wintel platform etc.).

3) Explain File models of a distributed file system.

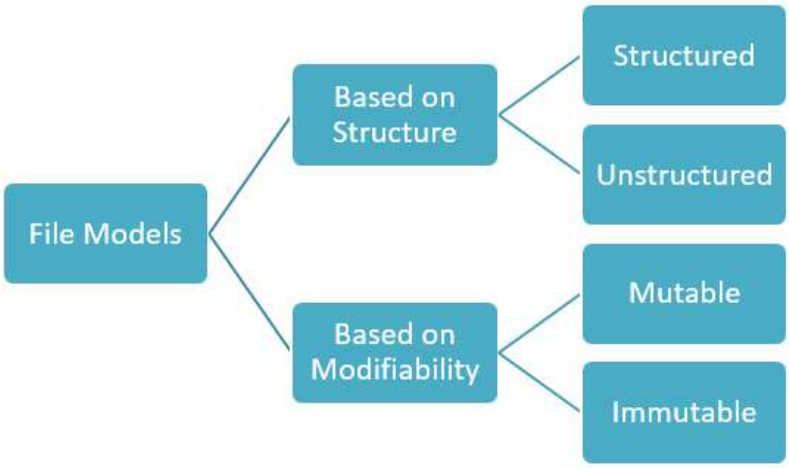


Figure: Classification of File models

Unstructured files

- In this model, there is no substructure known to the file server.
- Contents of each file of the file system appears to the file server as an uninterpreted sequence of bytes.
- Interpretation of the meaning and structure of the data stored in the files are entirely up to the application programs.
- UNIX, MS-DOS and other modern operating systems use this file model.
- This is mainly because sharing of a file by different applications is easier compared to the structured file model.

Structured files

- In structured files (rarely used now) a file appears to the file server as an ordered sequence of records.
- Records of different files of the same file system can be of different sizes.
- NetBSD operating system use this file model.
- Two types of structured files are:
  1. **Files with non-indexed records**
     - File record is accessed by specifying its position within the file.
     - For example, the fifth record from the beginning of the file or the second record from the end of the file.
  2. **Files with indexed records**
     - Records have one or more key fields and can be addressed by specifying the values of the key fields.
     - File is maintained as B-tree or other suitable data structure or hash table to locate records quickly.

Mutable and Immutable files

MUTABLE FILES	IMMUTABLE FILES
Update performed on a file overwrites on its old contents to produce the new contents.	Rather than updating the same file, a new version of the file is created each time a change is made to the file contents.
A file can be modified by each update operation.	A file cannot be modified once it has been created except to be deleted.
File overwrites its old contents to produce the new contents.	File versioning approach is used to implement file updates.
Most existing operating systems use the mutable file model.	It is rarely used now a day.

4) Explain File accessing models of a distributed file system.

- The file accessing model of a distributed file system mainly depends on two factors:
  1. The method used for accessing remote files
  2. The unit of data access

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

Advantages

- 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.
- It has better scalability because it requires fewer accesses to file servers, resulting in reduced server load and network traffic.
- 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.
- Once an entire file is cached at a client's site, it becomes immune to server and network failures.

Disadvantage

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

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

Advantages

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

Disadvantage

- 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

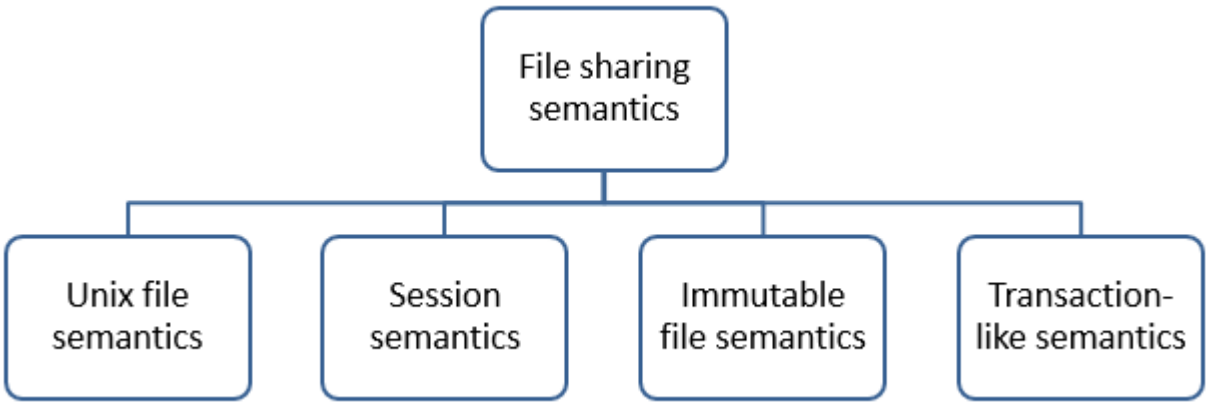
- **Byte level transfer model**
  - In this model, file data transfers across the network between a client and a server take pace in units of bytes.

  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.

- **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.

5) Explain File sharing semantics in distributed file system.

- A shared file may be simultaneously accessed by multiple users.
- In such a situation, an important design issue for any file system is to clearly define when modifications of file data made by a user are observable by other users.



1. Unix semantics

- This semantics enforces an absolute time ordering on all operations.
- Every read operation on a file sees the effects of all previous write operations performed on that file.
- This semantics can be achieved in a distributed system by disallowing files to be cached at client nodes.
- Allowing a shared file to be managed by only one file server that processes all read and write requests for the file strictly in the order in which it receives them.
- There is a possibility that, due to network delays, client requests from different nodes may arrive and get processed at the server node out of order.

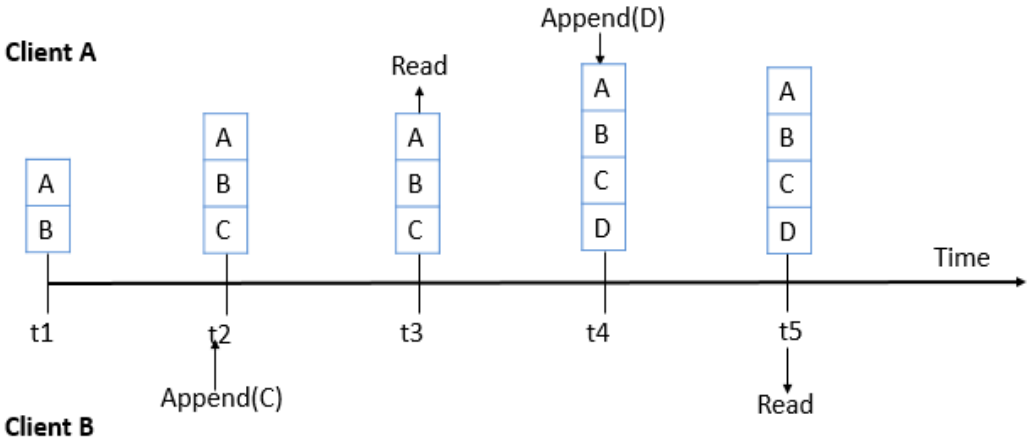


Figure: Unix semantics

2. Session semantic

- A client opens a file, performs a series of read/write operations on the file, and finally closes the file when he or she is done with the file.
- 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.

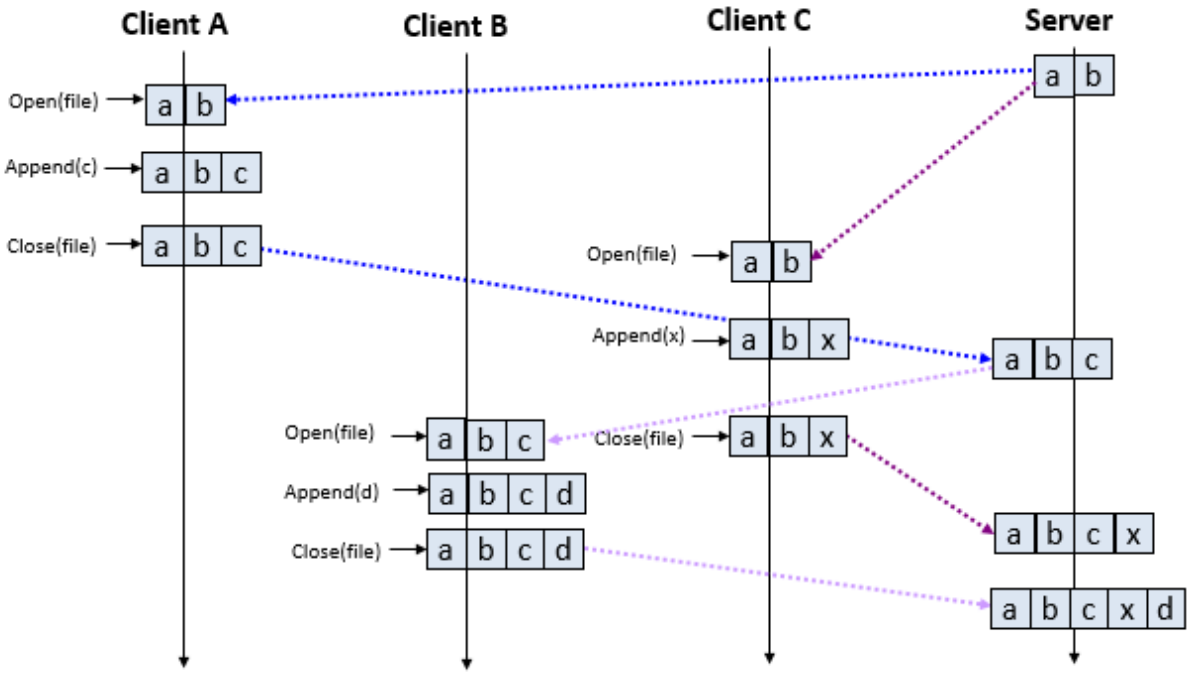


Figure: Session Semantics

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 are handled by creating a new updated version of the file.
- Therefore, the semantics allows files to be shared only in the read-only mode.

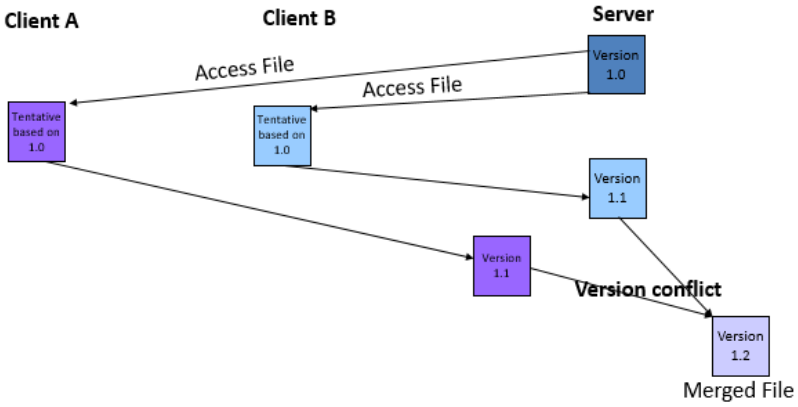


Figure: Immutable File semantics

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.

6) What is File caching scheme? Explain key decisions of File caching in Distributed File System.

- The idea in file caching is to retain recently accessed file data in main memory, to avoid repeated accesses.
- File caching reduces disk transfers substantially, resulting in better overall performance of the file system.
- A file-caching scheme for a distributed file system should address the following key decisions:
  - Cache location
  - Modification propagation
  - Cache validation

1. 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 locations in a distributed file system.
- **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.

Disadvantages

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

- **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.

Disadvantages

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

- **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.

Disadvantages

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

2. Modification propagation

- **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.

Advantages

- 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.

Disadvantages

- 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.

- **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.

  Advantages
  - 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.

3. 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:

a) Client Initiated Approach

- In this approach, a client contacts the server and checks whether its locally cached data is consistent with the master copy.
- **Checking before every access**
  - This approach defeats the main purpose of caching because the server has to be contacted on every access.
  - It is suitable for supporting UNIX-like semantics.
- **Periodic checking**
  - In this method, a check is initiated every fixed interval of time.
- **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.

b) 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 opens 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.

7) What is File replication? Write the Advantages of File Replication.

- File replication is the primary mechanism for improving file availability.
- A replicated file is a file that has multiple copies, with each copy located on a separate file server.
- The main problem in Replication is consistency, i.e. when one copy changes, how do other copies reflect that change?
- Often there is a tradeoff: consistency versus availability and performance.

Advantages of Replication

1. **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 copies of the data will be accessible increases.
2. **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.
3. **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.

4. **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.
5. **Improved system throughput**
   - Replication also enables several client's requests for access to the same file to be serviced in parallel by different servers, resulting in improved system throughput.
6. **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.
7. **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.

8) Give difference between replication and caching.

REPLICATION	CACHING
Replica is associated with a server.	**Cached copy is normally associated with a client.**
Existence of a replica depends on availability and performance requirement.	**Existence of a cached copy is dependent on the locality in file access patterns.**
More persistent.	**Less persistent.**
Widely known, secure, available, complete and accurate.	**Less known, secure, available, complete, and accurate.**
A replica does not change upon cached.	**A cached copy is subject to change upon a replica.**

9) Write a note on 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.

1. **Naming of Replicas**
   - Assignment of a single identifier to all replicas of an object seems reasonable for immutable objects because a kernel can easily support this type of object.
   - All copies are immutable and identical; there is only one logical object with a give identifier
   - However, in mutable objects, different copies of a replicated object may not be the same (consistent) at a particular instance of time.
   - In this case, if the same identifier is used for all replicas of the object, the kernel cannot decide which replica is the most up-to-date one.

- Therefore, the consistency control and management of the various replicas of a mutable object should be performed outside the kernel.
- It is the responsibility of the naming system to map a user-supplied identifier into the appropriate replica of a mutable object.

2. 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:

a) 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.
- Users also have the flexibility to delete one or more replicas of a replicated file.

b) Implicit/lazy replication

- In this type, the entire replication process is automatically controlled by the system without user's knowledge.
- That is, when a process creates a file, it does not provide any information about its location.
- The system automatically selects one server for the placement of the file.
- Later the system automatically creates replicas of the file on other servers, based on some replication policy used by the system.

10) Explain Multicopy Update Problem in File Replication and Approaches to handle it.

- As soon as a file system allows multiple copies of the same (logical) file to exist on different servers, it is faced with the problem of keeping them mutually consistent.

1. Read Only Replication

- This approach allows the replication of only immutable files.
- Files known to be frequently read and modified only once in a while such as files containing the object code of system programs, can be treated as immutable files for replication using this approach.

2. Read Any Write All Protocol

- In this method a read operation on a replicated file is performed by reading any copy of the file and a write operation by writing to all copies of the file.
- Before updating any copy, all copies are locked, then they are updated, and finally the locks are released to complete the write.

3. Available Copies Protocol

- This method allows write operations to be carried out even when some of the servers having a copy of the replicated file are down.
- In this method, a read operation is performed by reading any available copy, but a write operation is performed by writing to all available copies.
- When a server recovers after a failure, it brings itself up to date by copying from other servers before accepting any user request.
- Failed servers are dynamically detected by high-priority status management routines and configured out of the system while newly recovered sites are configured back in.

4. Primary Copy Protocol

- In this protocol, for each replicated file, one copy is designated as the primary copy and all the others are secondary copies.
- Read operations can be performed using any copy, primary or secondary.
- But all write operations are directly performed only on the primary copy.
- Each server having a secondary copy updates its copy either by receiving notification of changes from the server having the primary copy or by requesting the updated copy from it.

5. Quorum-Based Protocols

- Suppose that there are a total of n copies of a replicated file F.
- To read the file, a minimum r copies of F have to be consulted.
- This set of r copies is called a read quorum.
- To perform a write operation on the file, a minimum w copies of F have to be written.
- This set of w copies is called a write quorum.
- The restriction on the choice of the values of r and w is (r+w>n).
- That is, there is at least one common copy of the file between every pair of read and write operations resulting in at least one up-to-date copy in any read/write quorum.
- The version number of a copy is updated every time the copy is modified.
- A copy with the largest version number in a quorum is current.
- A read is executed as follows:
    - Retrieve a read quorum (any r copies) of F.
    - Of the r copies retrieved, select the copy with the largest version number.
    - Perform the read operation on the selected copy.
- A write is executed as follows:
    - Retrieve a write quorum (any w copies) of F.
    - Of the w copies retrieved, get the version number of the copy with the largest version number.
    - Increment the version number.
    - Write the new value and the new version number to all the w copies of the write quorum.

- **Examples of quorum consensus algorithm:(n=8, r=4, w=5)**

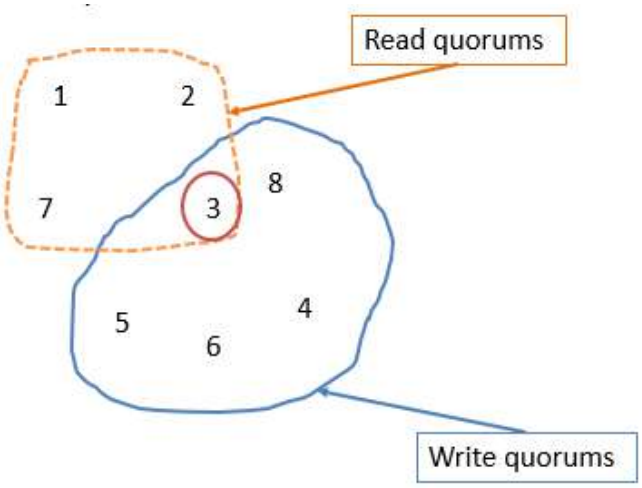


Figure: Example of Quorum based protocol

- There are total of eight copies of the replicated file (n = 8).
- The values of read and write quorums are 4 and 5, respectively (r =4, w= 5).
- The condition r +w > n is satisfied.
- Now suppose a write operation is performed on the write quorum comprised of copies 3, 4, 5, 6, and 8.
- All these copies get the new version and the new version number.
- Now any subsequent read operation will require a read quorum of four copies because r=4.
- any read quorum will have to contain at least one copy of the previous write quorum.

11) Explain file and Storage properties that directly influence the ability of a distributed file system to tolerate faults.

- The primary file properties that directly influence the ability of a distributed file system to tolerate faults are as follow:

1. Availability

- Availability of a file refers to the fraction of time for which the file is available for use.
- If a network is partitioned due to a communication link failure, a file may be available to the clients of some nodes, but at the same time, it may not be available to the clients of other nodes.

2. Robustness

- Robustness of a file refers to its power to survive crashes of the storage device and decays of the storage medium on which it is stored.
- Storage devices that are implemented by using redundancy techniques, such as a stable storage device, are often used to store robust files.

3. **Recoverability**
   - Recoverability of file refers to its ability to be rolled back to an earlier, consistent state when an operation on the file fails or is aborted by the client.

- In context of crash resistance capability, storage may be broadly classified into three types:

1. **Volatile Storage (RAM)**
   - It cannot withstand power failure or machine crash.
   - Data is lost in case of power failure or machine crash.

2. **Nonvolatile storage (Disk)**
   - It can withstand CPU failures but cannot withstand transient I/O faults and decay of the storage media.

3. **Stable Storage**
   - It can withstand transient I/O fault and decay of storage media.

12) Explain Effect of Service Paradigm on Fault Tolerance.

- A server may be implemented by using any one of the following two service paradigms stateful and stateless.

1. **Stateful File Servers**
   - A stateful server maintains client's state information from one remote procedure call to the next.
   - In case of two subsequent calls by a client to a stateful server, some state information pertaining to the service performed for the client as a result of the first call execution is stored by the server process.
   - These clients state information is subsequently used at the time of executing the second call.

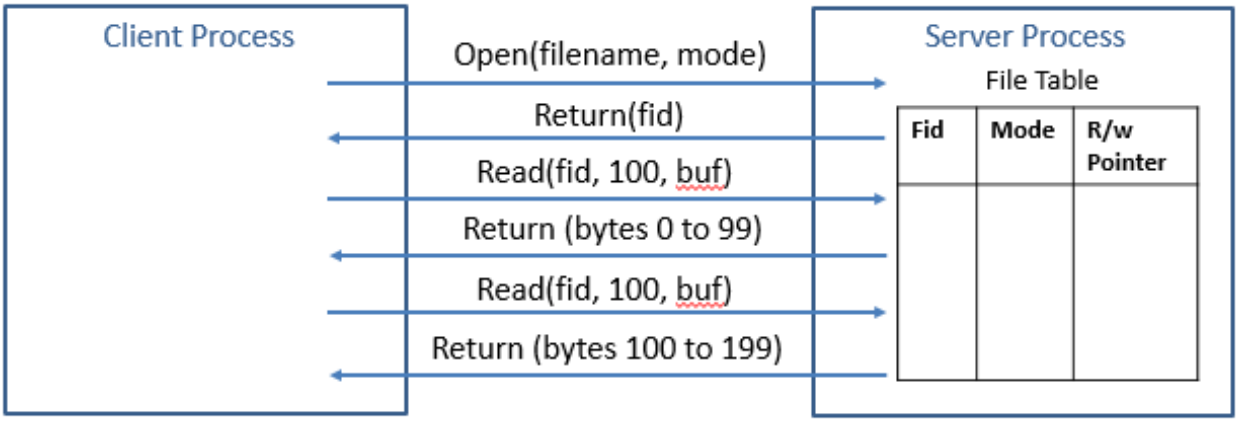


Figure: Stateful File Server

   - For example, let us consider a server for byte-stream files that allows the following operations on files.

- Following Figure shows example of stateful server.

1. **Open(filename,mode)**
   - This operation is used to open a file identified by filename in the specified mode.
   - When the server executes this operation, it creates an entry for this file in file table that it uses for maintaining the file state information of all the open files.
   - The file state information normally consists of the identifier of the file, the open mode, and the current position of a nonnegative integer pointer, called the read writer pointer.
   - When a file is opened, it read-write pointer is set to zero and the server returns to the client a file identifier(fid), which is used by the client for subsequent accesses to that file.

2. **Read(fid,n,buffer)**
   - This operation is used to get n bytes of data from the file identified by fid into the buffer named buffer.
   - When the server executes this operation, it returns to the client n bytes of file data starting from the byte currently addressed by the read write pointer and then increments the read writer pointer by n.

3. **Write(fid,n,buffer)**
   - On execution of this operation, the server takes n bytes of data from the specified buffer, writes it into file identified by fid at the byte position currently addressed by the read write pointer and then increments the read write pointer by n.

4. **Seek(fid,position)**
   - This operation causes the server to change the value of the read-write pointer of the file identified by fid to the new value specified as position.

5. **Close(fid)**
   - This statement causes the server to delete from its file table the file state information of the file identified by fid.

2. Stateless File server

- A stateless server does not maintain any client state information.
- Every request from client is accompanied with all the necessary parameters to successfully carry out desired operation.
- For example, a server for byte stream files that allow the following operations on file is stateless.

1. **Read(filename,position,n,buffer)**
   - On execution of this operation, the server returns to the client n bytes of data of the file identified by filename.
   - The returned data is placed in the buffer named buffer.
   - The value of actual number of bytes read is also returned to the client.

- The position within the file from where to being reading is specified as the position parameter.

2. **Write(filename,position,n,buffer)**
   - When the server executes this operation, it takes n bytes of data from the specified buffer and writes it into the file identified by filename.
   - The position parameter specifies the byte position within the file from where to start writing.
   - The server returns to the client the actual number of bytes written.

- If a client wishes to have similar effect as above Figure, the following two read operation must be carried out:
  - Read(filename,0,100,buf)
  - Read(filename,100,100,buf)

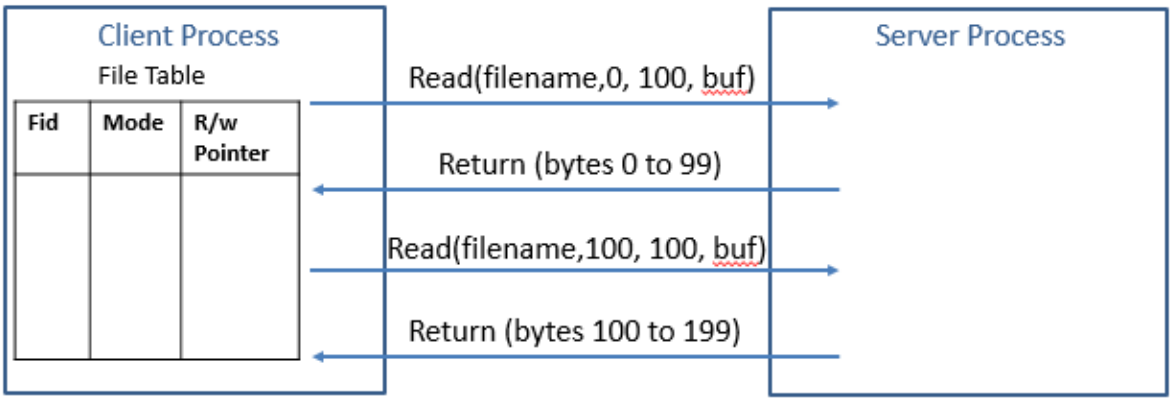


Figure: Stateless File Server

13) Give difference between Stateful File Server and Stateless File server.

PARAMETERS	STATEFUL SERVER	STATELESS SERVER
State	A Stateful server remember client data (state) from one request to the next.	A Stateless server does not remember state information.
Programming	Stateful server is harder to code.	Stateless server is straightforward to code.
Efficiency	More because clients do not have to provide full file information every time they perform an operation.	Less because information needs to be provided.
Crash recovery	Difficult due to loss of information.	Can easily recover from failure because there is no state that must be restored.
Information transfer	The client can send less data with each request.	The client must specify complete file names in each request.
Operations	Open, Read, Write, Seek, Close	Read, Write

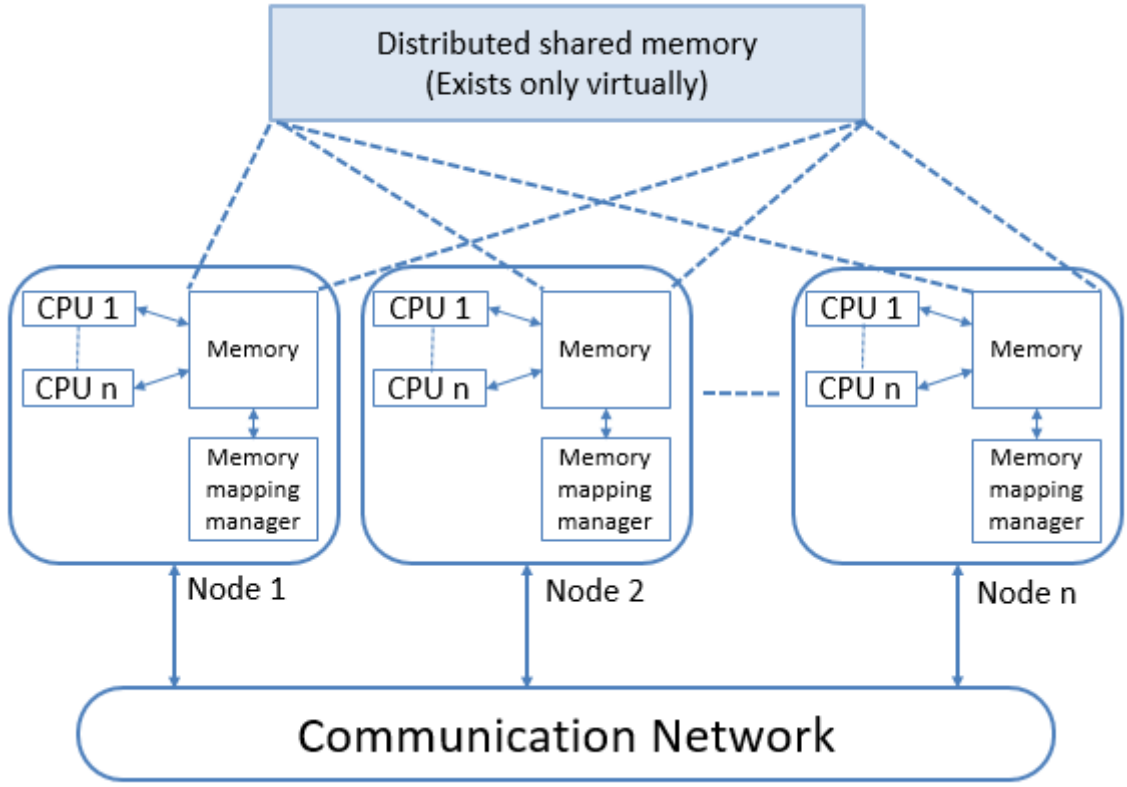
14) Write a note on Trends in Distributed File System.

- Some of the changes that can be expected in the future and some of the implications these changes may have for file systems are:

1. **New Hardware**
   - Within few years memory may become so cheap that the file system may permanently reside in memory, and no disks will be needed.
   - With an in-core file system, it may be much simpler to store each file contiguously in memory, rather than breaking it up into blocks.
2. **Scalability**
   - Algorithms that work well for systems with 100 machines may work poorly for systems with 1000 machines and not at all for systems with 10,000 machines.
   - If opening a file requires contacting a single centralized server to record the fact that the file is open, that server will eventually become a bottleneck as the system grows.
   - A general way to deal with this problem is to partition the system into smaller units and try to make each one relatively independent of the others.
3. **Wide Area Networking**
   - In the future, many LAN-based distributed systems will be interconnected to form transparent distributed systems covering countries and continents.
   - An inherent problem with massive distributed systems is that the network bandwidth is extremely low.
   - Bringing fiber optics into everyone's house will take decades and cost billions.
4. **Mobile Users**
   - A large fraction of the time, the user is off-line, disconnected from the file system.
   - Solution is probably going to have to be based on caching.
   - While connected, the user downloads to the portable those files expected to be needed later.
   - When reconnect occurs, the files in the cache will have to be merged with those in the file tree.
   - Since disconnect can last for hours or days, the problems of maintaining cache consistency are much more severe than in online systems.
5. **Multimedia**
   - Text files are rarely more than a few megabytes long, but video files can easily exceed a gigabyte.
   - To handle applications such as video-on-demand, completely different file systems will be needed.
6. **Fault Tolerance**
   - As distributed systems become more and more widespread, the demand for systems that essentially never fail will grow.
   - File replication, in current distributed systems, will become an essential requirement in future ones.
   - Systems will also have to be designed that manage to function when only partial data are available, since insisting that all the data be available all the time does not lead to fault tolerance.

1) Explain the DSM system architecture.

- Distributed shared memory (DSM) is one of the basic paradigm for inter process communication.
- The shared-memory paradigm provides shared address space to processes in a system.
- Processes access data in the shared address space through the following two basic primitives.
- Data = Read (address)
- Write (address, data)
- **Read** returns the data item referenced by address.
- **Write** sets the contents referenced by address to the value of data.



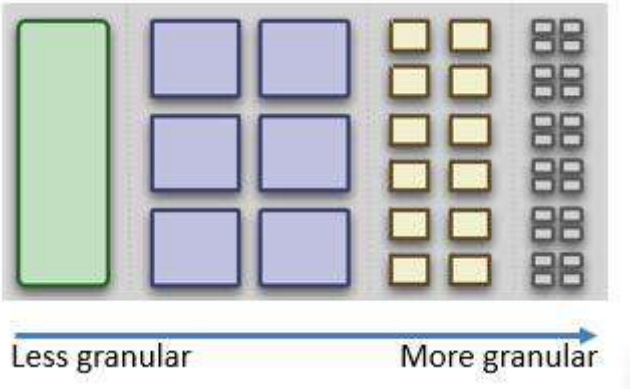
- Each node of the system consists of one or more CPUs and memory unit.
- Nodes are connected by high speed communication network.
- Simple message passing system for nodes to exchange information.
- Main memory of individual nodes is used to cache pieces of shared memory space.
- Memory mapping manager routine maps local memory to shared virtual memory.
- Shared memory of DSM exist only virtually.
- Shared memory space is partitioned into blocks.
- Data caching is used in DSM system to reduce network latency.
- Data block keep migrating from one node to another on demand but no communication is visible to the user processes.
- If data is not available in local memory network block fault is generated.

2) Explain Design and Implementation Issues of DSM.

- **Granularity**
  - 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.
  - Possible units are a few words, a page, or a few pages.
  - 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.
  -

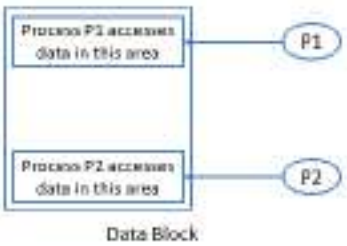
3) What is Granularity? Explain Criteria for choosing granularity parameter.

- Granularity is a block or unit of data transfer across the network.
- Selecting proper block size is an important part of the design of a DSM system.



Criteria for choosing granularity parameter are as follows.

1. **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.
   - This is usually due to the typical software protocols and overhead of the virtual memory layer of the operating system.
   - This fact favors relatively large block sizes.
2. **Paging overhead**
   - A process is likely to access a large region of its shared address space in a small amount of time.
   - Therefore, the paging overhead is less for large block size as compared to the paging overhead for small block size.
3. **Directory size**
   - The larger the block size, the smaller the directory.
   - Ultimately result in reduced directory management overhead for larger block size.
4. **Thrashing**
   - The problem of thrashing may occur when data item in the same data block are being updated by multiple node at the same time.
   - Problem may occur with any block size; it is more likely with larger block size.
5. **False sharing**
   - Occur when two different processes access two unrelated variables that reside in the same data block
   - The larger is the block size the higher is the probability of false sharing



6. **Using page size as block 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.
     - It allows the access right control.
     - Page size do not impose undue communication overhead at the time of network page fault.
     - Page size is a suitable data entity unit with respect to memory contention.

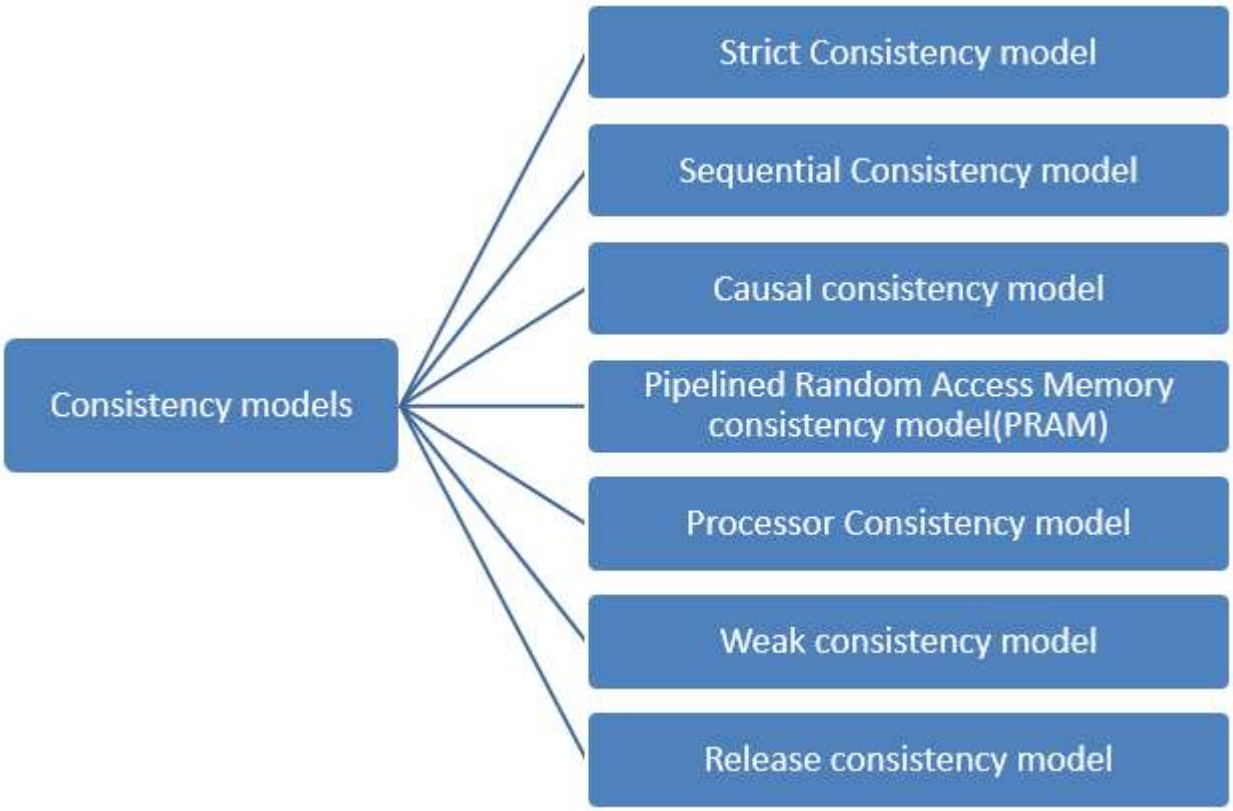
4) Explain Structure of shared memory space also write approach for structuring shared memory space.

- 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**
   - Most DSM systems do not structure their shared-memory space.
   - In these 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.
   - Therefore, it is simple and easy to design such a DSM system.
   - It also allows applications to impose whatever data structures they want on the shared memory.
2. **Structuring by data type**
   - The shared-memory space is structured either as a collection of objects (as in Clouds) or as a collection of variables in the source language (as in Munin).
   - The granularity in such DSM systems is an object or a variable.
   - Since the sizes of the objects and data types vary greatly, these DSM systems are variable grain size to match the size of the object/variable being accessed by the application.
   - The use of variable grain size complicates the design and implementation of these DSM systems.
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 tuple space, which is a collection of immutable tuples with typed data items in their fields.
   - A set of primitives that can be added to any base language are provided to place tuples in the tuple space and to read or extract them from tuple space.
   - To perform updates, old data items in the DSM are replaced by new data items.
   - Processes select tuples by specifying the number of their fields and their values or types.
   - Although this structure allows the location of data to be separated from its value, it requires programmers to use special access functions to interact with the shared-memory space.

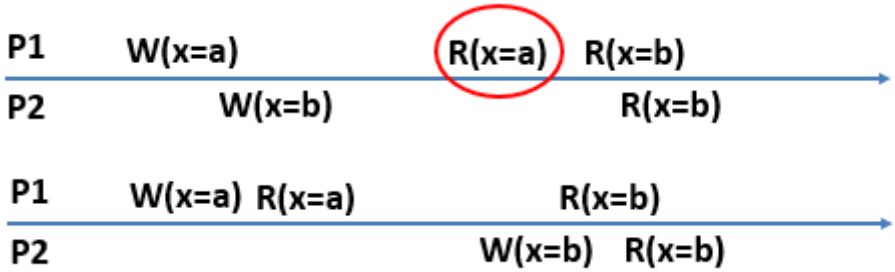
5) What are Consistency Models? Explain each Consistency Models in detail.

1. Consistency model defines set of rules that must obey the memory consistency.
2. A consistency model refers to the degree of consistency that has to be maintained for the shared memory data.
3. Consistency requirement vary from application to application.
4. If a system supports the stronger consistency model, then the weaker consistency model is automatically supported but the converse is not true.



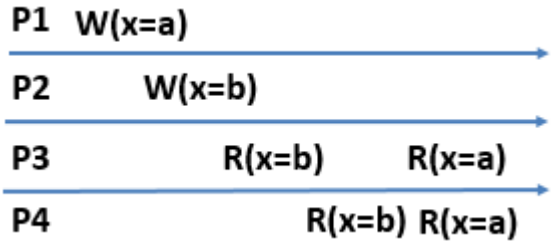
1. Strict Consistency Model

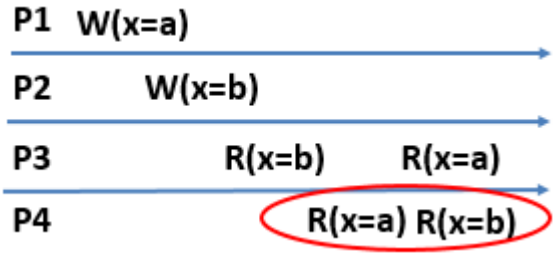
- Value returned by a read operation on a memory address is always same as the most recent write operation to that address.
- All writes instantaneously become visible to all processes.
- Implementation of this model for a DSM system is practically impossible.
- Practically impossible because absolute synchronization of clock of all the nodes of a distributed system is not possible.



2. Sequential Consistency Model

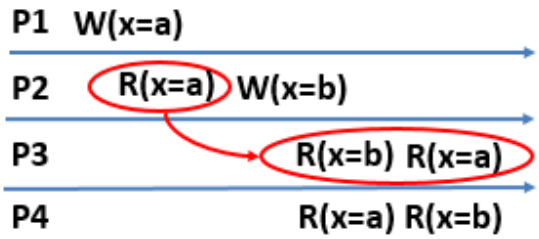
- A shared memory system is said to support the sequential consistency model if all processes see the same order.
- Exact order of access operations is interleaved does not matter.
- The consistency requirement of the sequential consistency model is weaker than that of the strict consistency model.
- It is Time independent process.
- 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.
- Note here that the only acceptable ordering for a strictly consistent memory is (r1, w1, r2).

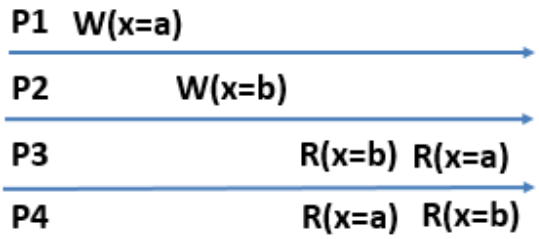




3. Causal Consistency Model

- All write operations that are potentially causally related are seen by all processes in the same(correct) order.
- Write operations that are not potentially causally related may be seen by different processes in different orders.
- A shared memory system is said to support the causal consistency model if all write operations that are potentially causally related are seen by all processes in the same (correct) order.
- 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.

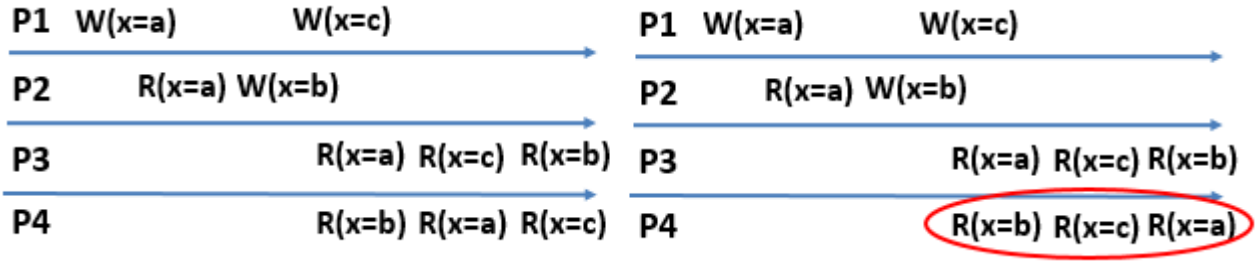




4. Pipelined Random Access Memory Consistency Model

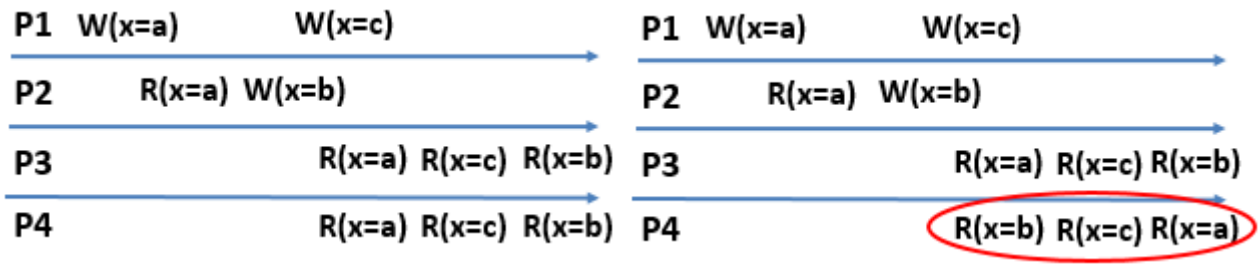
- Ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed.
- All write operations performed by a single process are in a pipeline.

- Write operations performed by different processes can be seen by different processes in different order.
- Simple and easy to implement and also has good performance.
- 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)].
- It leads to better performance because a process need not wait for a write operation performed by it to complete before starting the next one since all write operations of a single process are pipelined.



5. 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)].



6. Weak Consistency Model

- Changes in memory can be made after a set of changes has happened (Example: critical section).
- Isolated access to variable is usually rare, usually there will be several accesses and then none at all.
- Difficulty is the system would not know when to show the changes.
- Application programmers can take care of this through a synchronization variable.
- For supporting weak consistency, the following requirements must be met:
  - All accesses to synchronization variable must obey sequential consistency semantics.
  - All previous write operations must be completed everywhere before an access to a synchronization variable is allowed.
  - All previous accesses to synchronization variables must be completed before access to a nonsynchronization variable is allowed.

7. Release Consistency Model

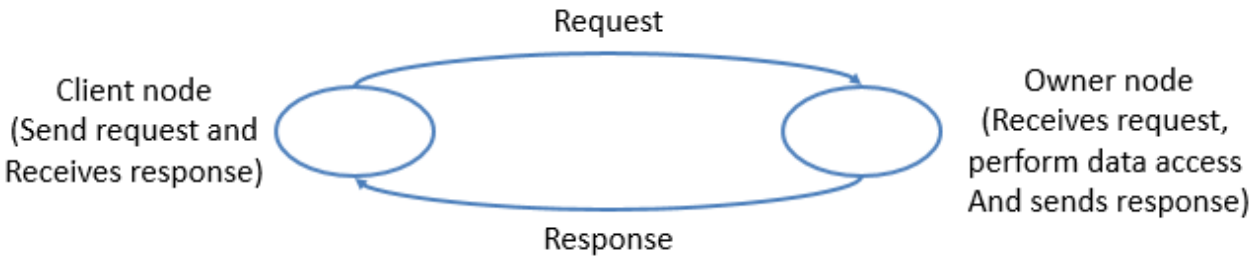
- All changes made to memory by process are propagated to other nodes.
- All changes made to the memory by other processes are propagated from other nodes to the process's node.
- It uses two synchronization variables
  - Acquire (used to tell the system it is entering CR)
  - Release (used to tell the system it has just exited CR)
- Acquire results in propagating changes made by other nodes to process's node.
- Release results in propagating changes made by the process to other nodes.
- Barrier defines the end of a phase of execution of a group of concurrently executing processes.
- Barrier can be implemented by using a centralized barrier server.

6) Write a note on Implementation of Sequential Consistency Model.

- Protocols for implementing the sequential consistency model in a DSM system depend to a great extent on whether the DSM system allows replication and migration of shared memory data blocks.
- Following are various migration and replication strategies:

1. Nonreplicated, Nonmigrating blocks (NRNMB)

- Each block of the shared memory has a single copy whose location is always fixed.
- All access request to a block from any node are sent to the owner node of the block, which has only copy of the block.
- Serializing data access creates a bottleneck.
- Parallelism is not possible.
- There is single copy of each block in the system.
- The location of the block never changes.

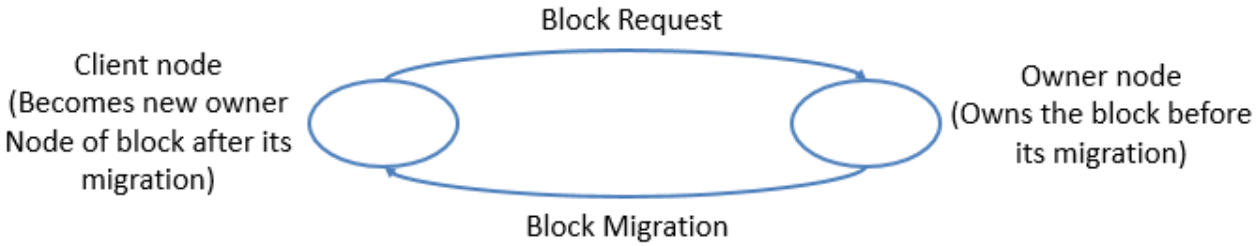


Data Locating in the NRNMB strategy:

- The NRNMB strategy has the following characteristics:
  - There is a single copy of each block in the entire system.
  - The location of block never changes.
- The best approach for locating a block in this case is to use a simple mapping function to map a block to a node.
- When a fault occurs, the fault handler of the faulting node uses the mapping function to get the location of the accessed block and forwards the access request to that node.

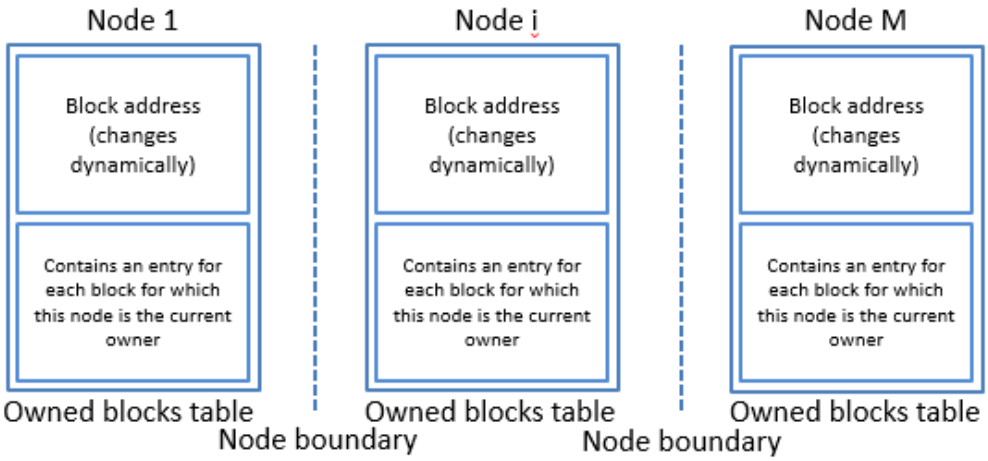
2. Nonreplicated, Migrating Blocks (NRMB)

- In this strategy each block of the shared memory has a single copy in the entire system.
- Each access to a block causes the block to migrate from its current node to the node from where it is accessed.
- The owner node of a block changes as soon as the block is migrated to anew node.
- When a block is migrated away it is removed from any local address space it has been mapped into.



Data Locating in the NRMB strategy:

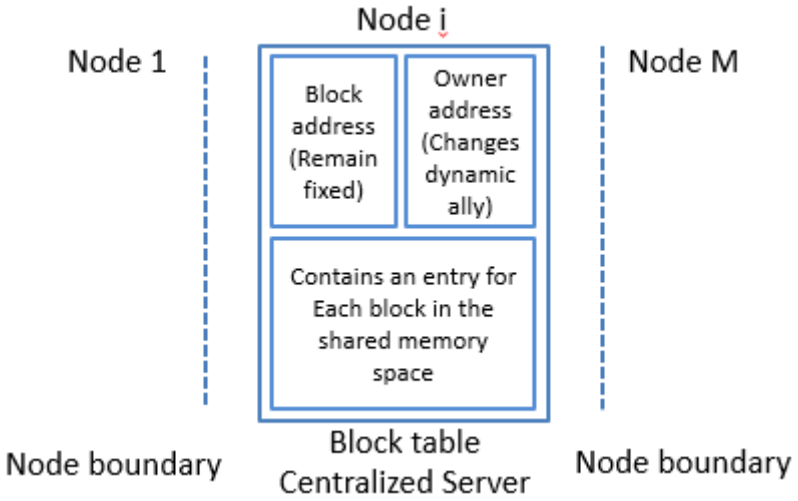
1. Broadcasting



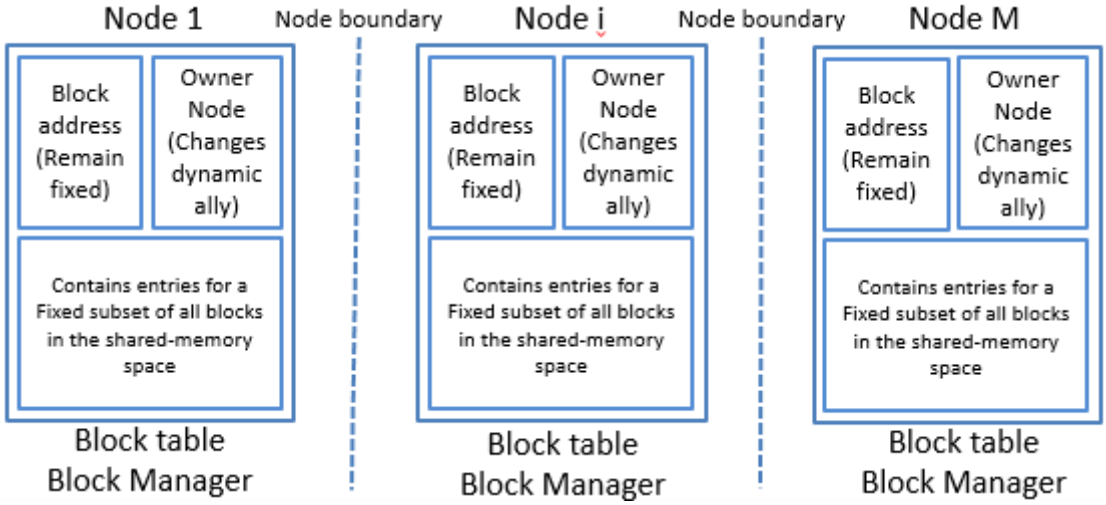
- Each node maintains an owned blocks table that contains an entry for each block for which the node is the current owner.
- When a fault occurs, the fault handler of the faulting node broadcasts a read/write request on the network.
- The node currently having the requested block then responds to the broadcast request by sending the block to requesting table.
- A major disadvantage of broadcasting algorithm is that it does not scale well.

2. Centralized Algorithm

- A centralized server maintains a block table that contains the location information for all block in the shared memory space.
- Faulting node sends a request for the accessed block to the centralized server.
- The centralized server extracts the location information from the block table and forwards the request to that node.



3. Fixed distributed server algorithm

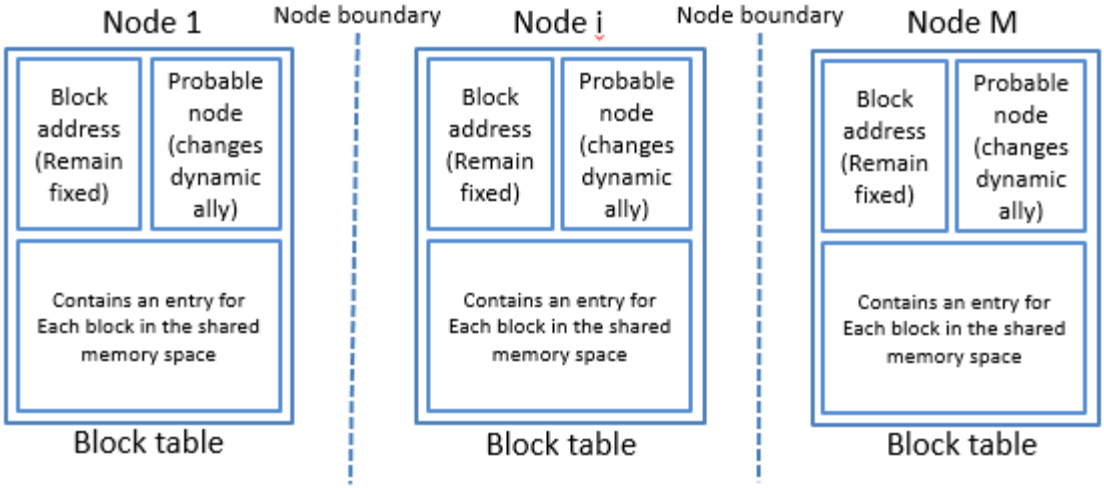


- It is a direct extension of the centralized server scheme.
- It overcomes the problems of the centralized server scheme by distributing the role of the centralized server.

- Each block manager is given a predetermined subset of data blocks to manage.
- The mapping function is used by the fault handler to find the node whose block manager is managing the currently accessed block.

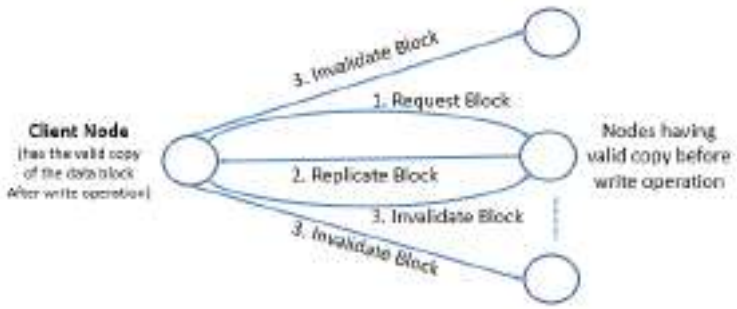
4. Dynamic distributed server algorithm

- Does not use any block manager and attempts to keep track of the Ownership information of all block in each node.
- Each node has a block table that contains the ownership information for all block.
- A field gives the node a hint on the location of the owner of a block and hence is called the probable owner.



3. Replicated, Migrating Blocks (RMB)

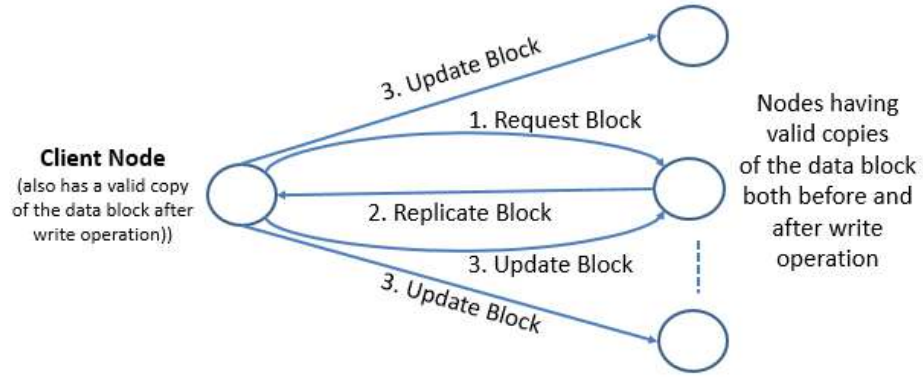
- To increase parallelism, all DSM systems replicate blocks.
- With replicated blocks, read operations can be carried out in parallel with multiples nodes.
- Write operations increase the cost because its replicas must be invalidated or update to maintain consistency.
- Two basic protocols that may be used for ensuring sequential consistency in this case are:
  1. **Write-invalidate**
     - All copies of a piece of data except one are invalidated before write operation can be performed on it.
     - When a write fault occurs, its fault handler copies the accessed block to its client node and invalidates all other copies.
     - The write operation is performed on client node.
     - The client node holds the modified version of block and is replicated to other nodes.



2. Write-update

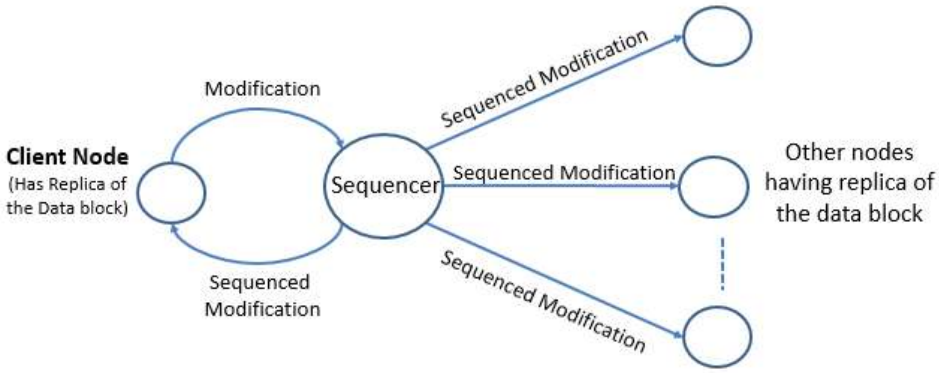
Method :1

- A write operation is carried out by updating all copies of the data on which the write is perform.
- When write fault occurs the fault handler copies the accessed block from one of the block's current node to its own node.
- The write operation completes only after all the copies of the block have been successfully updated.



Method :2

- Use a global sequencer to sequence the write operations of all nodes.
- Modification of each write operation is first sent to the global sequencer.
- The sequencer assigns the next sequence number to the modification.
- Multicasts the modification with this sequence number to all the nodes.



Data Locating in the RMB strategy:

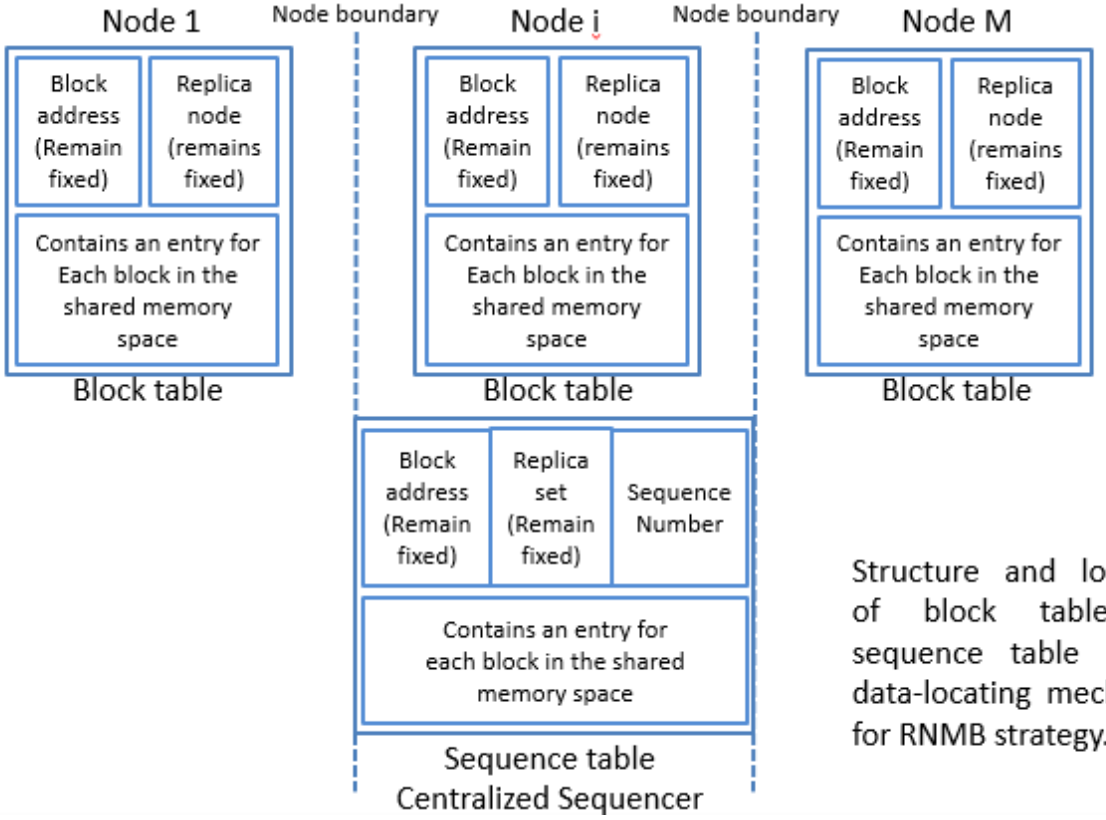
- Data locating issues are involved in the write-invalidate protocol used with the RMB strategy:
  - Locating the owner of a block, the most recent node has to write access to it.
  - Keeping track of the node that are currently have a valid copy of the block.
- Following algorithms may be used to address these two issues:
  - Broadcasting
  - Centralized-server algorithm
  - Fixed distributed-server algorithm
  - Dynamic distributed-server algorithm

4. Replicated, Nonmigrating Blocks (RNMB)

- A shared memory block may be replicated at multiple node of the system but the location of each replica is fixed.
- All replicas of a block are kept consistent by updating them all in case of a write access.
- Sequential consistency is ensured by using a global sequencer to sequence the write operation of all nodes.

Data Locating in the RNMB strategy:

- The replica location of a block never changes.
- All replicas of a data block are kept consistent.
- Only a read request can be directly send to one of the node having a replica.
- A write operation on a block is sent to sequencer.
- The sequencer assigns the next sequence number to the requested modification.
- It then multicast the modification with this sequence number to all the nodes listed in the replica set field.



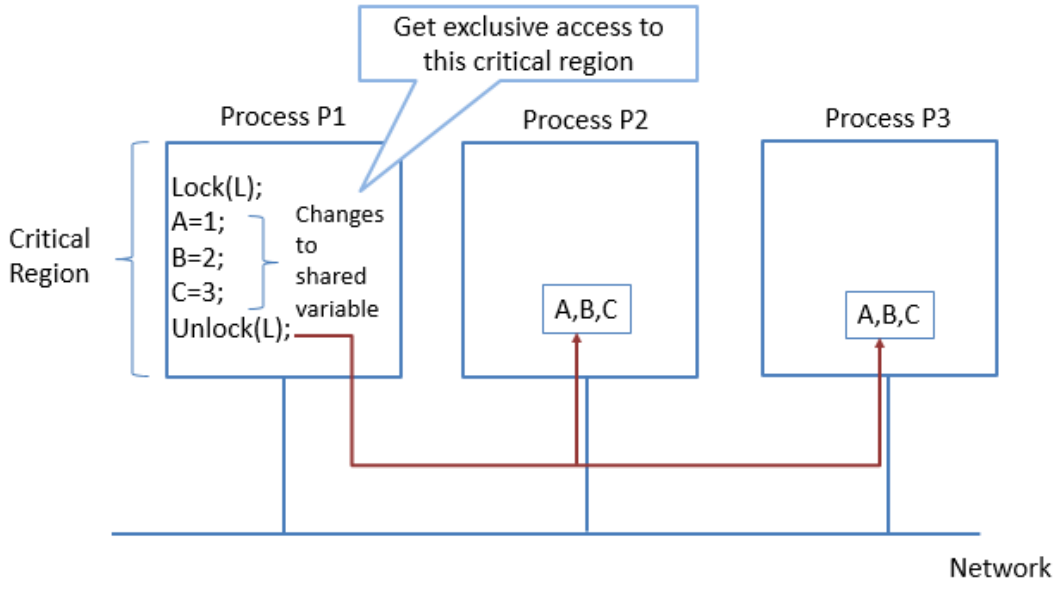
Structure and locations of block table and sequence table in the data-locating mechanism for RNMB strategy.

7) Write a note on Munin: A release consistent DSM system.

- **Structure of Shared-Memory Space:**
  - The shared memory space of Munin is structured as a collection of shared variables.
  - The shared variables are declared with the keywords shared so the compiler can have recognized them.
  - Each shared variable, by default, is placed on a separate page that is the unit of data transfer across the network.

- **Implementation of Release Consistency**
  - Release consistency application must be modeled around critical sections.
  - Therefore, a DSM system that supports release consistency must have mechanisms and programming language constructs for critical sections.
  - Munin provides two such synchronization mechanism.
    - Locking mechanism
    - Barrier mechanism

1. **Locking mechanism**
   - The locking mechanism uses lock synchronization variables with acquire lock and released lock primitives for accessing these variables.
   - Acquire lock primitive with a lock variable as its parameter is executed by a process to enter a critical section.
   - Release lock primitive with the same lock variable as its parameter is executed by the process to exit from the critical section.
2. **Barrier mechanism**
   - The barrier mechanism uses barrier synchronization variables with a Wait at Barrier primitive for accessing these variables.
   - Barriers are implemented by using the centralized barrier server mechanism.



- **Annotation for shared variables**
- The standard annotations and consistency protocol for variables of each type are as follows:
  - Read-only
  - Migratory
  - Write-shared
  - Producer-consumer
  - Result
  - Reduction
  - Conventional

8) Explain Block replacement strategy.

- DSM system allows shared memory block to be dynamically migrated/replicated.
- The following issues must be addressed when the available space for caching shared data fills up at a node:

1. Which block should be replaced to make space for a newly required block?

- **Usage based verses non-usage based**

USAGE BASED ALGORITHM	NON- USAGE BASED ALGORITHM
Usage based algorithms keep track of the history of usage of a cache line (or page) and use this information to make replacement decisions.	Non usage based algorithms do not take the record of use of cache lines into account when doing replacement.
Example: Least recently used (LRU) algorithm.	Example: First in first out (FIFO) algorithm.

- **Fixed space verses variable space**

FIXED SPACE ALGORITHM	VARIABLE SPACE ALGORITHM
Fixed space algorithms assume that the cache size is fixed.	Variable space algorithm assume that the cache size can be changed dynamically.
Replacement in fixed space algorithms simply involves the selection of a specific cache line.	In variable space algorithm, fetch does not imply a replacement, and a swap-out can take place without a corresponding fetch.

2. Where to place a Replaced Block

- The two commonly used approaches for storing a useful block at the time of its replacement are as follows:

- **Using secondary storage**
  - The block is simply transferred on to a local disk.
  - The advantage of this method is that it does not waste any memory space and if the node wants to access the same block again; it can get the block locally without a need for network access.

- **Using memory space of other nodes**
  - Other method for storing a useful block is to keep track of free memory space at all nodes in the system and to simply transfer the replaced block to the memory or a node with available space.
  - This method requires each node to maintain a table of free memory space in all other nodes.
  - This table may be updated by having each node piggyback its memory status information during normal traffic.

9) Define thrashing in DSM. Explain methods for solving thrashing problem in DSM.

- Thrashing is said to occur when the system spends a large amount of time transferring shared data blocks from one node to another.
- Thrashing may occur in following situation:
  - When interleaved data accesses made by processes on two or more nodes.
  - When blocks with read only permissions are repeatedly invalidated soon after they are replicated.
  - Thrashing degrades system performance considerably.
- Methods for solving Thrashing problems:
  - Providing application controlled locks
    - Locking data to prevent other node from accessing for a short period of time can reduce Thrashing.
  - Nailing a block to a node for a minimum amount of time
    - Disallow a block to be taken away from a node until a minimum amount of time elapses after its allocation to that node.
    - Drawback: It is very difficult to choose the appropriate value for the time.
  - Tailoring the coherence algorithm to the shared-data usage patterns
    - Thrashing can also be minimized by using different coherence protocols for shared data having different characteristics.

10) Write Advantages of DSM System.

- **Simpler Abstraction**
  - Programming loosely coupled distributed memory machines using message passing models is tedious and error phone.
  - RPC was introduced to provide a procedure call interface.
  - Even in RPC, since the procedure call is performed in an address space different from that of the caller's address space, it is difficult for the caller to pass context related data or complex data structures; that is, parameters must be passed by value.
  - The shared memory programming paradigm shields the application programmers from many such low level concerns.
  - Thus primary advantage of DSM is the simpler abstraction it provides to the application programmers of loosely coupled distributed memory machines.
- **Better Portability of Distributed System**
  - Distributed application programs written for a shared memory multiprocessor system can be executed on a distributed shared memory system without change.

- Therefore, it is easier to port an existing distributed application program to a distributed memory system with DSM facility than to a distributed men system without this facility.

- **Better Performance of Some Applications**
  - **Locality of Data**
    - The communication model of DSM is to make the data more accessible by moving it around.
    - DSM algorithms normally move data between nodes in large blocks.
    - Thus in those applications that exhibit a reasonable degree of locality in their data accesses, communication overhead is reduced over multiple memory accesses.
    - This ultimately results in reduced overall communication cost for such applications.
  - **On Demand Data Movement**
    - The computation model of DSM also facilitates on demand movement of data as they are being accessed.
    - There are several distributed applications that execute in phase, where each computation phase is preceded by a data exchange phase.
    - The time needed for the data exchange phase is often dictated by the throughput of existing communication bottlenecks.
  - **Larger Memory Space**
    - With DSM facility, the total memory size is the sum of the memory sizes of all the nodes in the system.
    - Thus, paging and swapping activities, which involve disk access, are greatly reduced.
- **Flexible Communication Environment**
  - The shared memory paradigm of DSM provides a more flexible communication environment than message passing approach in which the sender process need not specify the identity of the receiver processes of the data.
  - It simply places the data in the shared memory and the receivers access it directly from the shared memory.
  - Therefore, the coexistence of the sender and receiver processes is also not necessary in the shared memory paradigm.

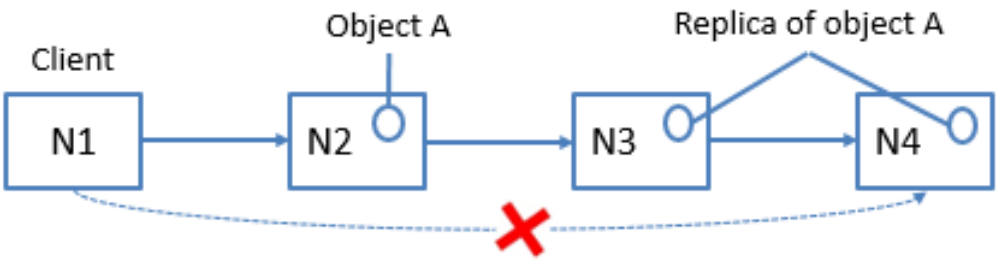
1) What is Naming in Distributed system? Write Desirable features of good naming system.

- A distributed system comprises various objects like nodes, processes, files, I/O devices, etc.
- The naming system plays a vital part in the design of a distributed system.
- The naming system comprises two important mechanisms:
- **The naming mechanism**
  - The naming mechanism incorporated in the distributed OS enables character names to be assigned to these objects so that they can be subsequently referred to.
- **The locating mechanism**
  - The locating mechanism is responsible for mapping the object's name to the object's location in the distributed system.
- The most important function of the naming system is to provide location transparency in a distributed system.
- It also facilitates transparent migration, replication of objects, and object sharing.
- An object can also have multiple names.
- The naming system also maps these multiple names to the same object.

Desirable features of good naming system

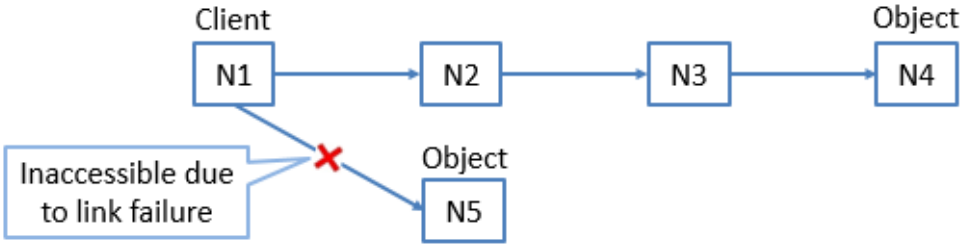
1. **Location transparency**
   - This feature implies that the name of the object must not indicate the physical location of the object, directly or indirectly.
   - The name should not depend on physical connectivity, topology of the system, or even the current location of the object.
2. **Location Independency**
   - Any distributed system allows object migration, movement, and relocation of objects dynamically among the distributed nodes.
   - Location independency means that the name of the object need not be changed when the object's location is changed.
   - The user must be able to access the object irrespective of the node from where it is being accessed.
3. **Scalability**
   - The naming system should be capable of adapting to the dynamically-changing scale, leading to dynamic changes in namespace.
   - Any changes in system scale should not require any change in naming or location mechanisms.
4. **Uniform Naming Convention**
   - Most of distributed systems use some standard naming conventions for naming different types of objects.
   - File names are named differently from user names and process names.
   - Ideally a good naming system must use the same naming mechanism for all types of objects, so as to reduce the complexity of designing.

5. **Meaningful names**
   - Meaningful names indicate what the object actually contains.
   - A good naming system must support at least two levels of object identifiers; one which is convenient for users and the other which is relevant for machines.
6. **Allow multiple user-defined names for the same object**
   - Users must be allowed to use their own names for accessing the object.
   - A naming system must have the flexibility of assigning multiple user-defined names to the same object.
   - It should be possible for the user to either change or delete the user-defined name for the object without affecting the names given by other users to the same object.
7. **Group Naming**
   - Broadcast facility is used to communicate with groups of nodes, which may not comprise the entire distributed system.
   - Good naming system must allow many different objects to be identified by the same name.
8. **Performance**
   - One important aspect of performance of a naming system is the time required to map an object's name to its attributes.
   - This depends on the number of messages transmitted to and fro during the mapping operation.
   - The naming system should use a small number of message exchanges.
9. **Fault tolerance**
   - The naming system must be capable of tolerating faults such as the failure of a node or network congestion.
   - The naming system should continue functioning, may be poorly, in case of such failures, but should still be operational.
10. **Replication transparency**
    - Replicas are generally created in a distributed system to improve system performance and reliability.
    - A good naming system should support multiple copies of the same object in a user transparent manner.
11. **Locating the nearest replica**
    - When a naming system supports replicas, the object location mechanism should locate the nearest replica of the requested object.



12. Locating all replicas

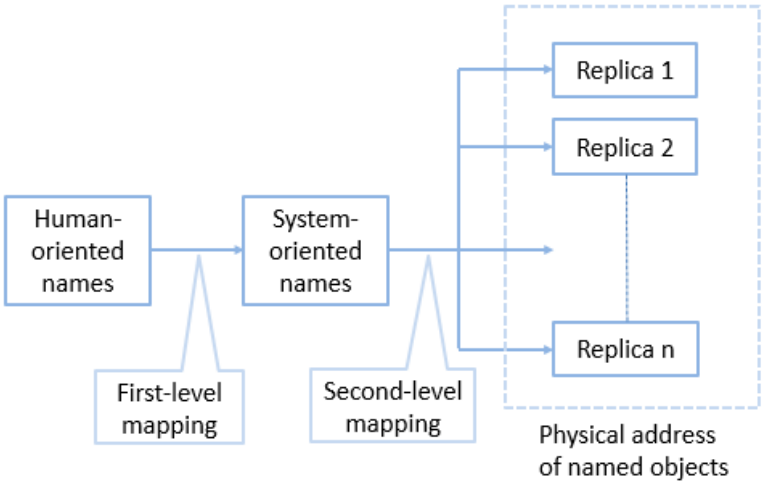
- From a reliability and consistency point of view, the object-locating mechanism must be able to locate all replicas.



2) Write Fundamental Terminologies & Concepts of Naming in Distributed System.

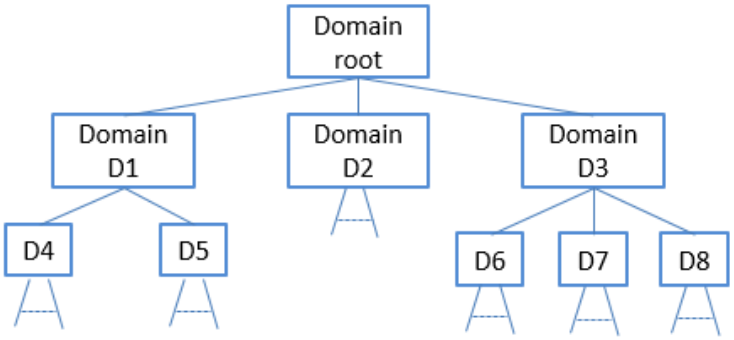
- **Name**
- Name is a string composed of a set of symbols chosen from a finite alphabet.
- It comprises of characters, numerals, and punctuation symbols from the ASCII character set.
- Name is also called an identifier because it is used to denote or identify an object.
- Examples of names are: TEST, $asd123, node-1!, 234wer, etc.

- **Human-oriented names**
  - Human-oriented name is a set of characters that is meaningful to users.
  - For example, /user/sinha/project-1/file-1 is a human-oriented name.
  - Human-oriented names are defined by the users.
  - Human-oriented names are independent of the physical location of the object.
  - Human-oriented names are also called high-level names.
- **System-oriented names**
  - System-oriented names are bit patterns of fixed size that can be easily manipulated and stored by machines.
  - System-oriented names are automatically generated by the system.
  - These names can be easily manipulated and stored by machines.
  - These names are also called unique identifiers or low-level names.



Mapping names in a namespace

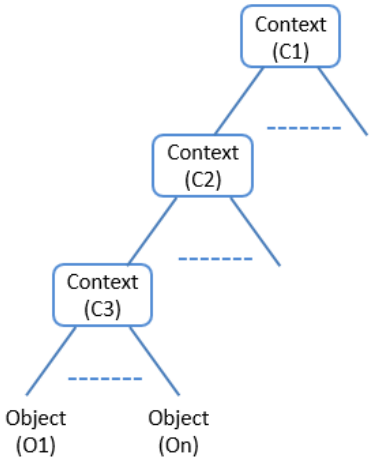
- **Name space**
- Namespace is defined as the set of names within a distributed system that complies with the naming convention.
- **Flat namespace**
  - Flat namespace is a simple name space with names as character strings.
  - Flat names do not have any structure and hence they are not suitable for large systems.
- **Partitioned namespace**
  - Partitioned namespaces are desirable in situations involving a large set of objects.
  - The partitioning is done syntactically and the name structure represents physical association.
- **Name server**
- Name server is used to maintain information about named objects and enable users to access that information.
- It also binds the object's name to its property (such as location).
- Each name server will store information about a subset of objects in the distributed system.
- The authoritative name servers store the information about the object and the naming service maintains these names.



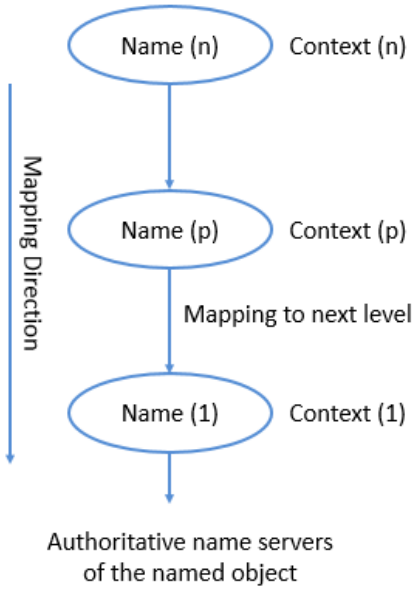
- The root node stores information only about the location of the name servers that are coming out from root such as D1, D2 and D3.
- Domain D1 node stores information about domains D4 and D5.
- Domain D1 node stores information about domains D6, D7 and D8.

- **Name agent**
- Name agents act as an interface between the name servers and the clients.
- The major function of a name agent is to maintain information about all the existing name servers in the system.
- **Private Name agent**
  - Works for a single client.
  - Structured as a set of subroutines that are linked to the client program.
- **Shared Name Agent**
  - Works for several clients.
  - Structured as a part of an operating system kernel, with system calls to invoke name service operations.

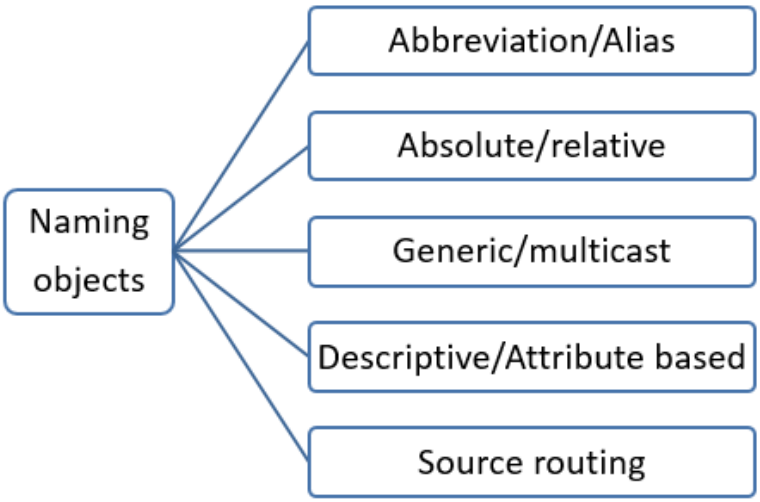
- **Context**
- Names are always associated with some context.
- A context can be thought of as the environment in which a name is valid.
- Contexts helps in partitioning the naming information database to distribute among multiple name servers.
- Example: The qualified name for object O1 that is associated with context C3 will be C1/C2/C3/O1.



- **Name resolution**
- Name resolution is the process of mapping an object's name to the object's properties, such as its location.
- The name resolution process works as follows:
  - A client contacts the name agent.
  - The name agent contacts the name server to locate the object.
  - If the object is not located, then known name server contacts other name servers.
  - The name resolution process involves traversing the context chain till the specific authoritative name server is found.



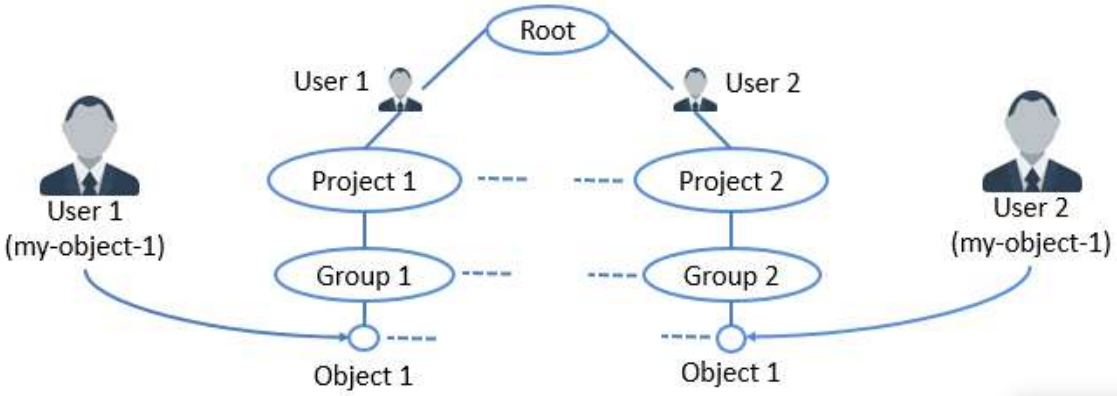
- **Naming Objects**
- There are multiple names used to name the same object and conversely the same name may mean different objects.



- **Abbreviations/Aliases**
  - In a partitioned namespace, a qualified name may be a combination of several names and hence of long length.
  - To avoid inconvenience to the user, the naming convention may allow users to access these objects with user-defined short forms for qualified names called **aliases or abbreviations.**
  - These names form the private context of the user and hence a separate mapping is maintained on this context basis.
  - For example, two Users, User-l and User-2, may use the same abbreviation my-object-1 to identify their objects having the qualified names.

/user-1/project-1/group-1/object-1

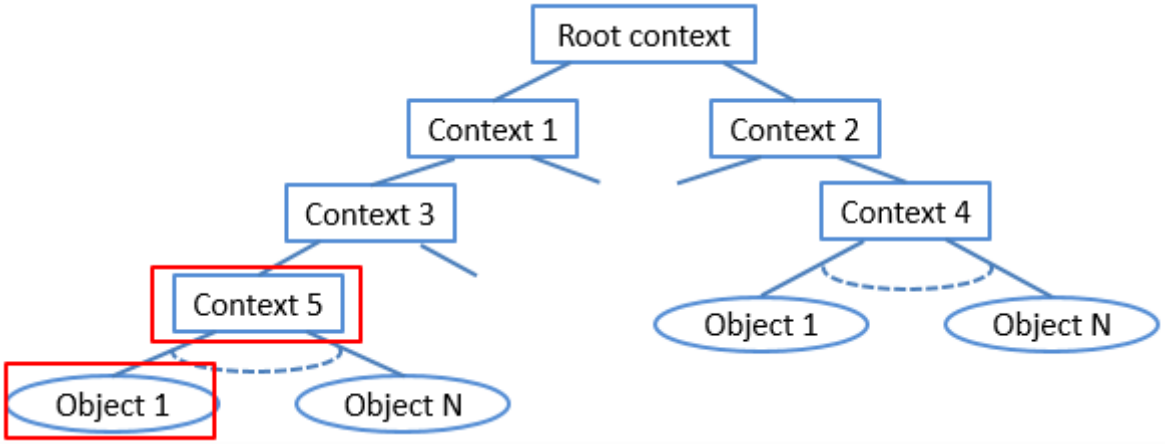
/user-2/project-2/group-2/object-1



- **Absolute and relative names**
  - An absolute name is specified from the root to the specified object.
  - A relative name is specified from the current context to the specified object.
  - The user's context is context5 and the object to be located is object-1.
  - Absolute name of the object is:

    Root context/context1/context3/context5/object-1
  - Relative name of this same object in context 5 is:

    context5/object-1



- **Generic and multicast names**
  - **Generic Names**
  - Name is mapped to any one object of the set to which it is bound.
  - This method is used in the scenario when a user wants a request to be serviced by any of the servers capable of servicing the request.
  - **Multicast names**
  - Name is mapped to all object of the set to which it is bound.
  - This naming method is useful for broadcasting or multicasting.

- **Descriptive/Attribute-based names**
  - A naming system supports descriptive or attribute-based names and allows an object to be named with a set of attributes.
  - An attribute has a type (indicating format and meaning) and a value.
  - Examples of attributes are:
    - User: Darshan
    - Creation date: 01/01/2017
    - File type: Source
    - Language: C
    - Name: Test1
  - To identify an object, it is not necessary to specify all the attributes.
  - Multicast naming facility can be easily provided with this naming method by constructing an attribute for a list of names.

- **Source-routing names**
  - When the structure of a name space has the same form as the underlying network of a distributed system, the name space defines source-routing names.
  - A typical example is the UNIX-to-UNIX Copy (UUCP) namespace that defines the names of the form:

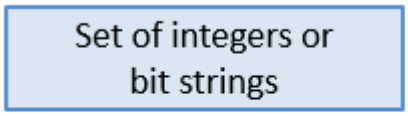
    host-1!host-2!host-3!darshan
  - This style is called as source routing namespace because the route through the network is specified at the source computer.

3) Explain Features and Types of System oriented Names.

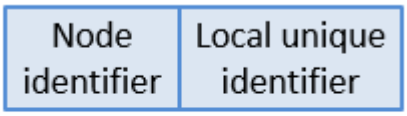
- **Characteristic features of System-oriented names:**
  - Names are integers or bit strings.
  - These names are also referred as unique identifiers, since they are guaranteed to be unique in both time and space.
  - These names are automatically generated by the system.
  - Names are of the same size irrespective of the type or location of the object identified by the names.
  - It is easy to perform operations like hashing, sorting, etc. on them.
  - These names are hard to crack and hence prove useful in case of security-related situations.

Types of System Oriented Names

- **Unstructured names**
  - Unstructured names have a single set of integers or a bit string which uniquely identifies the object.
  - It does not provide any other information about the object.
- **Structured names**
  - Structured names have additional component which provide information about the properties of the object.
  - It provides information about the properties of the object identified by its name.

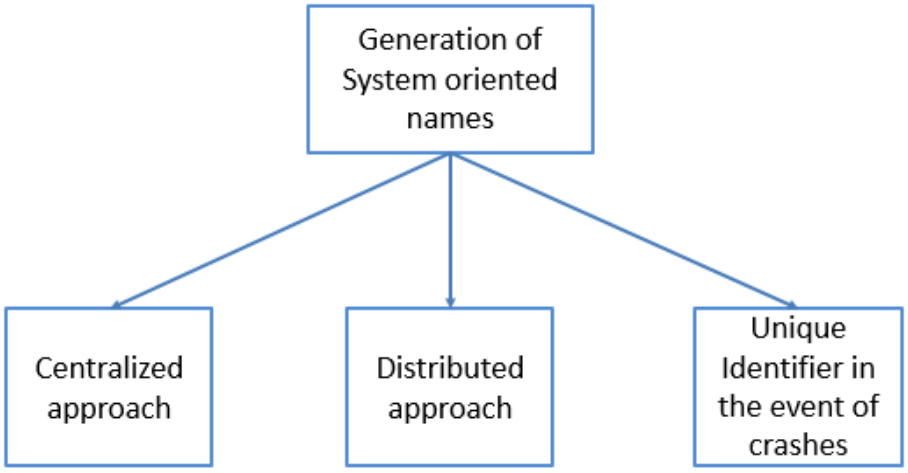


Unstructured name

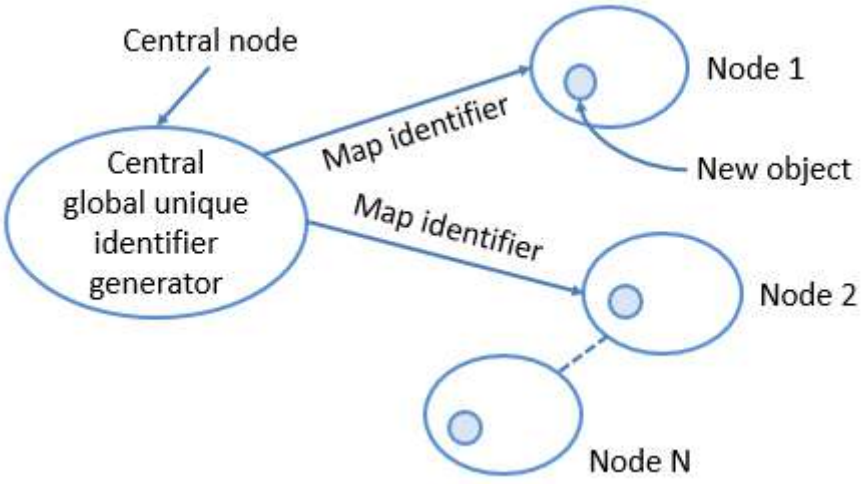


Structured name

4) Explain Approaches to generate system oriented names.



- **Centralized approach**
  - The centralized approach is basically used for generating unstructured names.
  - This system incorporates a centralized global unique identifier generator that generates a standard global unique identifier for each object in the system.
  - Then, any method is used to partition this global namespace among local domains of each node.
  - Each node may either bind or map the global identifier to the locally created object.

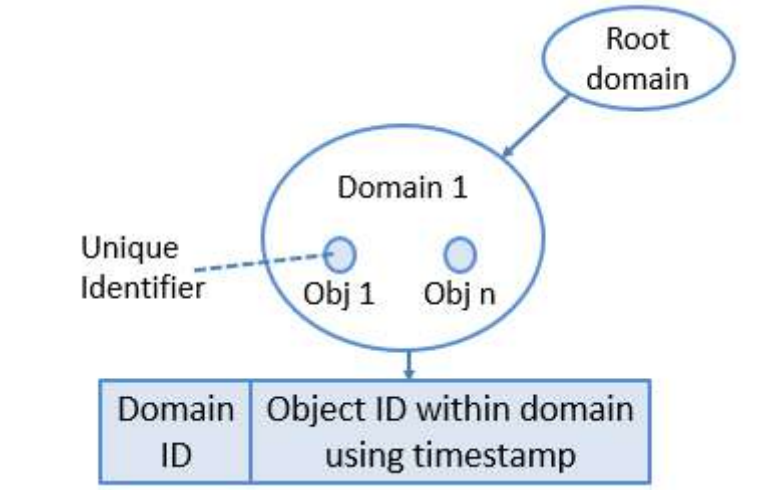


- **Distributed approach**
  - It forces the naming system to use structured object identifiers.
  - The hierarchical concatenation strategy is used to create global unique identifiers.
  - First, each domain is given a unique identifier.
  - The global unique identifier for an object is created by concatenating the unique identifier of the domain with an identifier used within the domain.

  Technique 1
  - Treat each node of the system as a domain of the name space.

- Treat a reading from the real-time clock (called timestamp) of a node as a unique identifier within the node.
- Global unique identifiers take the form of the pair of

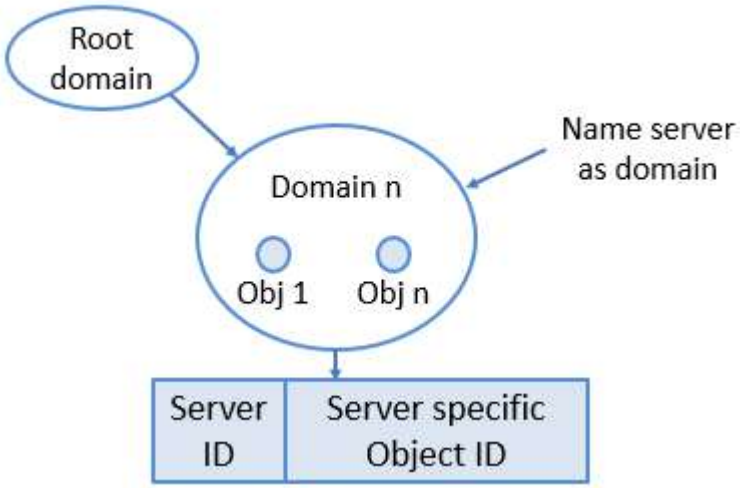
(Domain ID, Identifier with in domain using timestamp)



Technique 2

- Treat each server as a domain and then to allow a server to generate object identifiers for the objects it serves in a server-specific manner.
- Global unique identifiers take the form of the pair of

(server ID, server specific unique identifier)



- **Generating Unique Identifiers in the event of crashes**
  - A crash may lead to loss of the state information of a unique identifier generator.
  - On recovery, the unique identifier generator may not function correctly, which may result in generate of non-unique identifiers.
  - This problem can be resolved by following methods:

Using a clock that operates across failures

- Clock is used at the location of the unique identifier generator.

- It will not recycle during the period within which the needed identifiers must be unique.
- To implement this method, one may require rather long identifiers, depending on the granularity of the clock interval needed.

Using multiple levels of storage

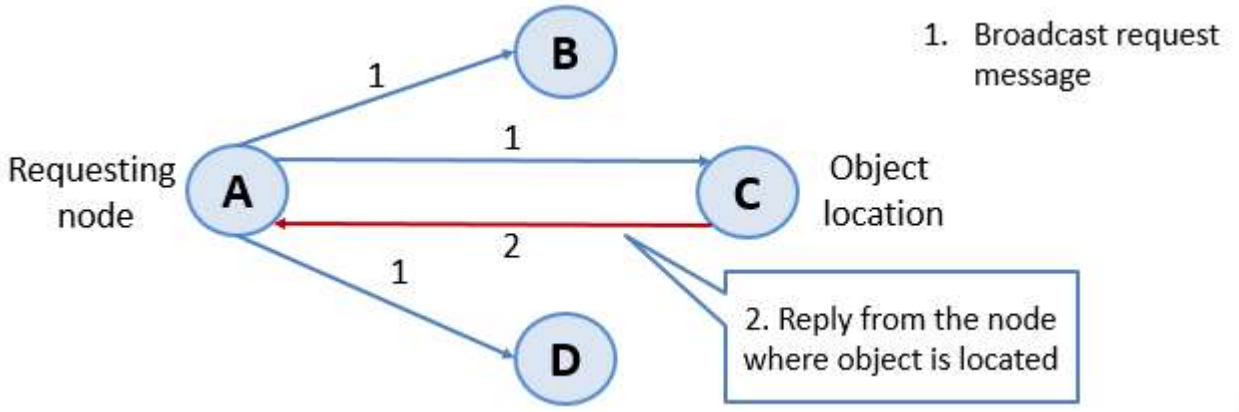
- The unique identifiers are structured in a hierarchical fashion with one field per level.
- A counter maintained at each level contains the highest value of the field assigned.
- Current value is cached in the main memory.
- In case of a crash at one level, the next higher level is incremented while the low-level counter is reset.
- Two levels with main memory and stable storage are sufficient for creating a safe and efficient unique identifier generator.

5) What is Object-locating mechanism? Explain types of object-locating mechanisms.

- Object locating is the process of mapping an object's system-oriented unique identifier from the replica locations of the object.
- Different types of object-locating mechanisms are as follows:

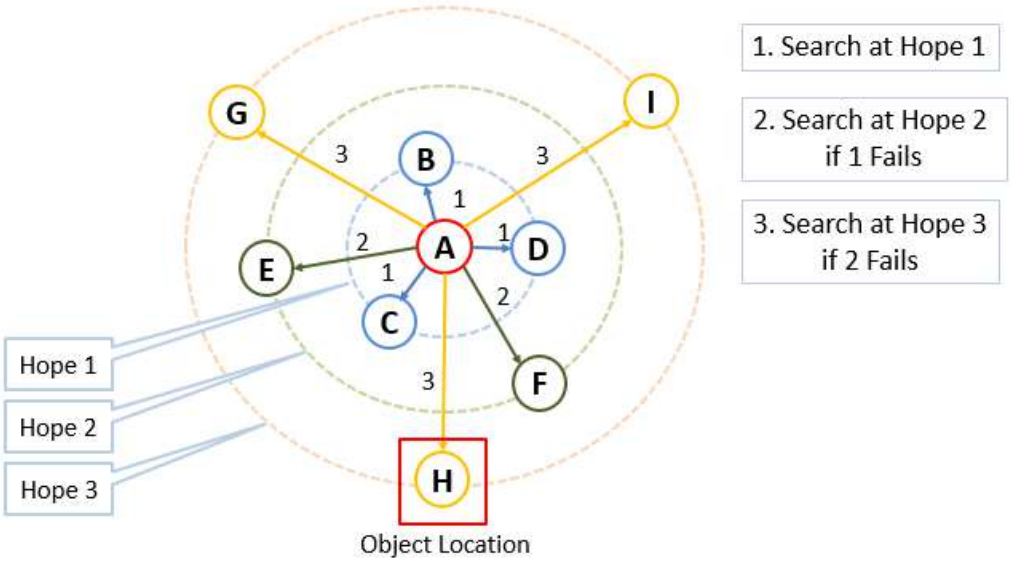
1. Broadcasting

- The node with an object request broadcasts the request to all nodes in the system.
- The node currently having the requested object replies to the requesting node.
- This method is suitable for small systems and those systems having a low frequency of object-locating requests.

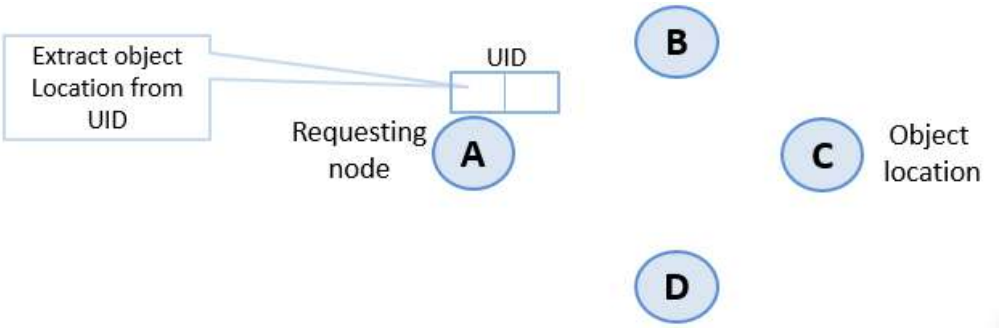


2. Expanding ring broadcast

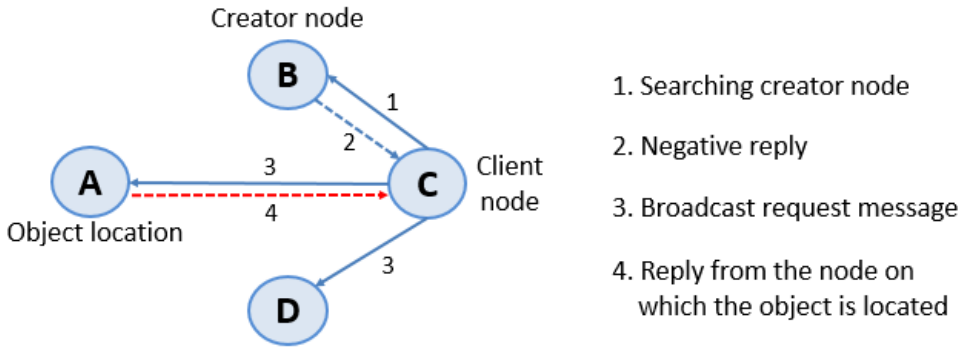
- Expanding Ring Broadcast (ERB) is a modification of the pure broadcast method.
- It is used for Wide Area Networks (WAN).
- In this method, increasingly distant LANs are searched till the object is found or till every LAN searched is unsuccessful.
- This increase is in terms of a hop, which corresponds to a gateway between two processors.
- A ring is a set of LANs at a specific distance from the requesting processor.



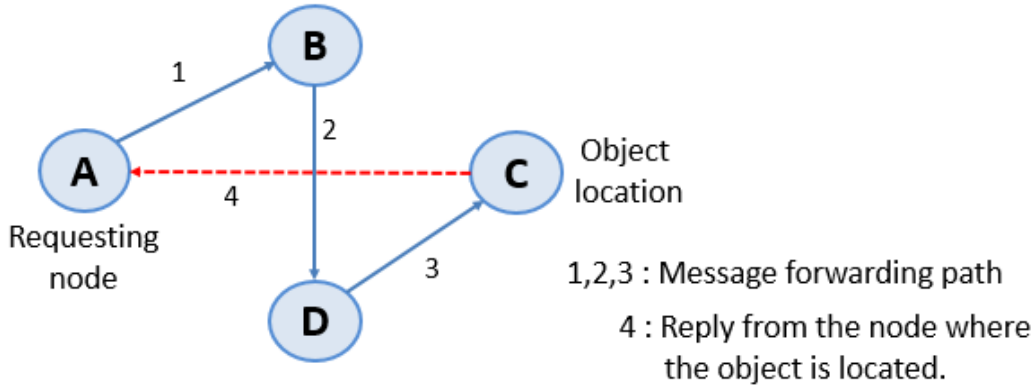
3. **Encoding location of object within its UID**
   - The encoding mechanism of object location uses structured identifiers whose one field indicates the location of the object.
   - From the UID, the requesting node extracts the object location.
   - The object will be fixed to one node throughout its lifetime.
   - This method of object location is limited to those distributed systems that do not support object migration and replication.



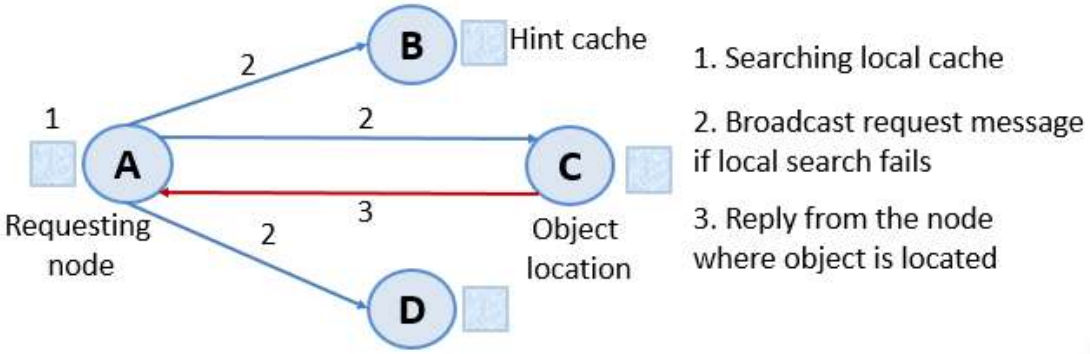
4. **Searching creator node first and then Broadcasting**
   - This method is based on the principle of locality which states that an object is most likely to remain on the node where it was created.



5. **Using forward location pointers**
   - A forward location pointer is stored at a node when the object is migrated to a new location.
   - The system contacts the creator node of the object and follows the forward chain of pointers to the node where the object is currently located.



6. **Using hint cache and broadcasting**
   - This method maintains a cache on each node that contains a UID and the last known location of the recently accessed remote objects.
   - The requesting node first searches the local hint cache for the UID.
   - If the UID search is successful, the location information is extracted from the cache; else
   - The request is sent to the node specified in the extracted location information.
   - In the worst case, if the object is not on that node, the request is returned.



6) Explain Issues in designing human oriented names.

- **Selecting a proper scheme for global object naming**
  - Combining an object's local name with its host name.
  - Interlinking isolated namespaces into a single namespace.
  - Sharing remote namespaces on explicit request.
  - Providing a single global namespace of object naming.
- **Selecting a proper scheme for partitioning a name space into contexts**
  - Context is used for partitioning a name space into smaller components.
  - Contexts represent indivisible units for storage and replication of information regarding named objects.
  - A name space is partitioned into contexts by using clustering conditions.

- **Selecting a proper scheme for implementing context bindings**
  - The context of a namespace is distributed among multiple name servers that manage the namespace.
  - A name server hence stores only a small subset of all the contexts of the namespace.
  - When a sure requests a name resolution, a name server looks into its local context for an authority attribute of an object.
  - If an authority attribute is not found in the local server, the name server searches for additional configuration data called context bindings.
  - A context binding associates the context within which it is stored to another context and the name server that stores that context.
- **Selecting a proper scheme for name resolution**
  - Name resolution is the process of mapping an object's name to the authoritative name servers of the object.
  - This process is a traversal of the resolution chain of contexts till the authority attribute of the named object is reached.
  - Name resolution depends on the policy used for distributing the contexts of namespaces.
  - The commonly used name resolution mechanisms are.
    - Centralized Approach
    - Fully replicated approach
    - Distribution based on the physical structure of namespace
    - Structure-free distribution of contexts

7) Explain Characteristics of Name Service Activities.

Characteristics of Name Service Activities

- **High degree of locality of name lookup**
  - This is similar to the property of locality of reference observed in program execution, file access, database access, etc.
  - A high degree of locality exists in using pathnames for accessing objects.
  - The hit ratio can be increased by providing a reasonable size name cache that caches recently used naming information.
- **Slow update of name information database**
  - Users are mostly confined to the namespace which is only a small subset of the slowly changing name information database.
  - Also, the naming data has a high read-to-modify ratio.
  - This means that the cost of maintaining the consistency of cached data is relatively low.
- **On-use consistency of cached information is possible**
  - In case there is access to obsolete naming data, it will not work and hence it can be attended to at the time of use.
  - So name consistency can be maintained by detecting and discarding stale cache entries on use.
  - Thus, there is no need to invalidate all related cache entries when a naming update occurs and stale data does not cause mapping name to a wrong object.

8) Explain Issues in name cache design.

- There are various issues involved in name cache design:
- **Types of Name caches**
  - A name cache stores the name resolution information about recently accessed objects'
  - names.
  - Name caches can be categorized depending on the type of information stored in each entry.
  - **Directory cache:**
    - In this type of cache, each entry consists of a directory page and it is mostly used in the iterative method of name resolution.
    - All recently used directory pages that are brought to the client node during name resolution are cached for some time.
  - **Prefix cache**
    - This type of cache is basically used in systems with zone-based context distribution mechanism.
    - Each entry consists of a name prefix and the zone identifier corresponding to that zone.
    - This type of cache is not suitable in those systems that use structure-free context distribution mechanisms.
  - **Full-name cache**
    - Each entry in this type of cache consists of an object's full path name and the identifier and the location of its authoritative name server.
    - Hence the requests for accessing an object that is available in the local cache can be sent directly to the authoritative name server.
    - The full name cache is suitable for all naming mechanisms.
- **Approaches to name cache implementation**
  - A name cache can be implemented using two approaches:
    - a cache-per-process
    - a single cache for all processes
  - In the first approach, each cache is maintained per process and in the process' address space.
  - Since it is small in size, information can be accessed faster and the OS memory area does not remain occupied with this information.
  - This name cache is deleted once the process is terminated and for every new process, a new name cache is created.
  - In case the process is short-lived, the hit ratio will be low due to start-up misses that occur when a new name cache is created.
  - It is also possible that the same naming information may be duplicated in many caches on the same node.
  - Start-up misses are reduced in the V-system by enabling the child process to inherit the initial contents of the parent process.
  - The other approach is to maintain a single name cache at each node for all the processes at that node.

- These caches are hence larger in size as compared to per-process name cache and are located in the OS's memory.
- The cache information is long-lived, since it uses standard replacement policies when there is a need to evacuate the cache, leading to a higher hit ratio.
- This method also avoids the problem of duplicating the naming information.

- **Multi Cache Consistency**
  - Whenever a naming data is updated, the relative name cache entries become stale and they need to be either updated or invalidated.
  - Two commonly used methods to maintain multi-cache consistency are immediate invalidate and on-use update.
  - **Immediate invalidate**
    - As the name suggests the naming information is immediately invalidated when the naming update occurs.
    - It is done in two ways: broadcast method and multicast method.
    - In the first approach, the invalidation message is broadcast to all nodes in the system.
    - Each node examines its own name caches and invalidates cache entry if it has that data.
    - This method increases network traffic overhead.
    - In the second approach, a storage node keeps a list of nodes against each data that corresponds to the nodes on which the data is cached.
    - A storage node receives an update request; it checks its list and the corresponding nodes are notified to invalidate their cache entries.
  - **On-use update**
    - It is a commonly used method for maintaining naming consistency.
    - When a client uses a stale entry from a cache, it is informed by the naming system that the data being used is stale.
    - Once this message is received, either using broadcast or multicast, the updated data is sent to refresh the stale cache entry.

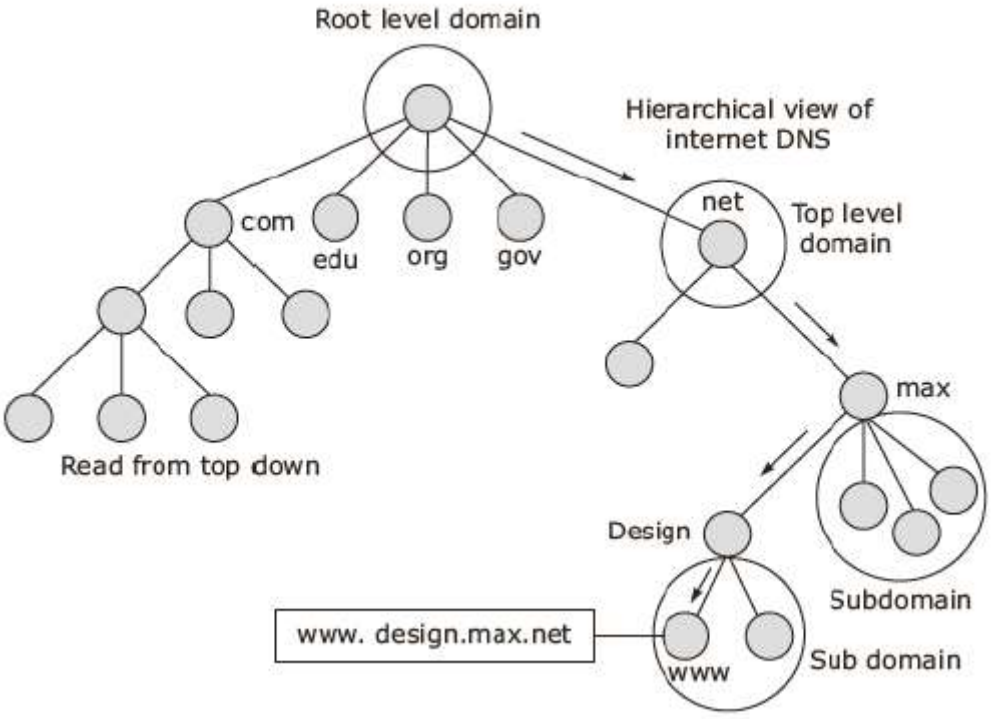
9) Write a note on Naming and security in Distributed System.

- Various naming-related access control mechanisms are as follows:
- **Object names as protection keys**
  - An object's name is used as a protection key for the object.
  - A user who knows the name of an object can access the object.
  - The users are not allowed to name the objects that they are not authorized to access.
  - It is not possible for a user to access another user's objects.
- **Capabilities**
  - Capability is a special type of identifier that contains additional information for protection in one or more of the permission modes.
  - Capabilities are protected objects that are maintained by the OS and are indirectly accessed by the users.

- **Associating protection with name resolution path**
  - The name resolution path method uses access control list (ACL) based protection that controls access based on user identity.
  - Trusted identifier must be a password, address, or any other identifier form that cannot be forged.

10) Write a note on Domain Name Service (DNS).

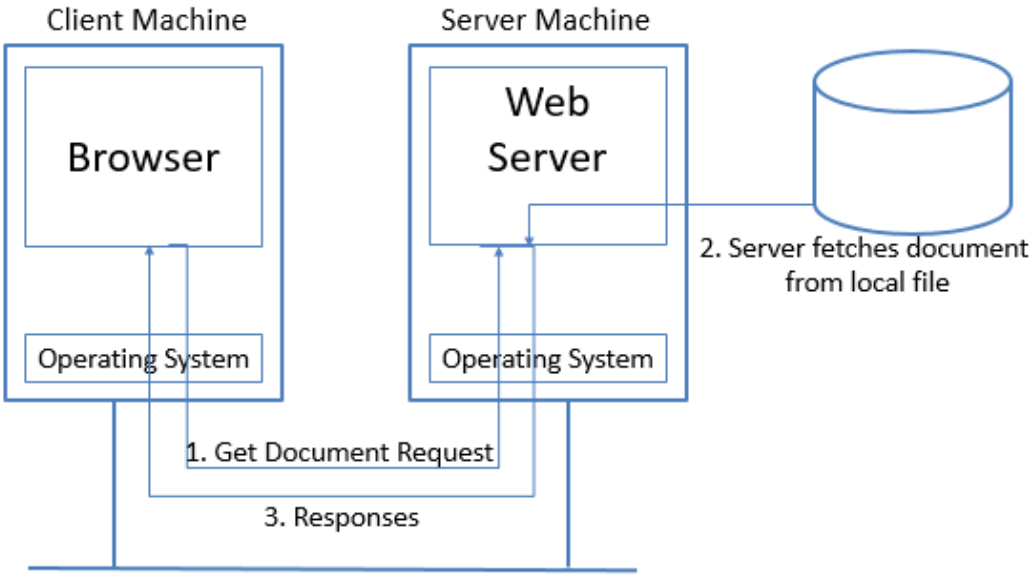
- Domain Name Service (DNS) is widely used to access the Internet.
- It is an Internet directory service that provides a way to map the user-friendly name of a computer or service to its numeric address.
- The DNS defines and describes how domain names are translated into IP addresses, and how it controls email delivery.
- A client computer queries a DNS server, asking for the IP address of a computer configured to use host-a.example.microsoft.com as its DNS domain name.
- The DNS server answers the query based on its local database and replies with an answer containing the requested information, which contains the IP address information for host a.example.microsoft.com.
- The DNS system consists of three components:
  - DNS data called resource records
  - Servers called name servers
  - Internet protocols for fetching data from the servers
- The billions of resource records on the Internet are split into millions of files called zones.
- Zones are kept on authoritative servers distributed all over the Internet that answer queries based on the resource records stored in the zones they have copies of.



- To standardize DNS naming across the Internet, different organizations were assigned authoritative controls of top-level domains. The original seven top-level domains are:
  - com (commercial organizations)
  - edu (educational organizations)
  - gov (government organizations)
  - mil (military organizations)
  - net (networking organizations)
  - org (noncommercial organizations)
  - int (International organizations)
- A domain name usually consists of two or more parts and is conventionally written separated by dots, such as example.com.
- The rightmost label conveys the top-level domain (for example, the address www.example.com has the top-level domain com).
- Each label to the left specifies a subdivision or the sub-domain of the domain above it.
- The DNS is maintained by a distributed database system that uses the client-server model. The nodes of this database are the name servers.
- Each domain or sub-domain has one or more authoritative DNS servers that publish information about that domain and the name servers of any domains subordinate to it.
- The top of the hierarchy is served by the root name servers: the servers to query when lookingup (resolving) a top-level domain name (TLD).
- For example, a DNS recursor consults three name servers to resolve the address www.wikipedia.org.
- Most name servers are authoritative for some zones and perform a caching function for all other DNS information.
- Large name servers are often authoritative for tens of thousands of zones, but most name servers are authoritative for just a few zones.

1) Draw and explain Architecture of Traditional Web Based system.

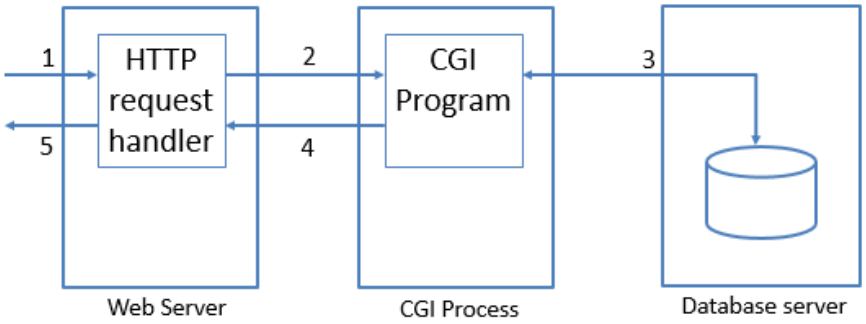
- Many Web-based systems are organized as simple client-server architecture.
- The simplest way to refer to a document is by means of a reference called a Uniform Resource Locator (URL).
- It specifies where a document is located, often by embedding the Domain name server (DNS) of its associated server.
- A URL specifies the application-level protocol for transferring the document across the network.
- A client interacts with Web servers through a special application known as a browser.
- A browser is responsible for properly displaying a document.
- A browser accepts input from a user mostly by letting the user select a reference to another document, which it then subsequently fetches and displays.
- The communication between a browser and Web server is standardized: they both adhere to the Hyper Text Transfer Protocol (HTTP).



- **Web documents**
  - Fundamental to the Web is that all information comes in the form of a document.
  - Most documents can be divided into two parts:
    - A main part that at the acts as a template.
    - The second part consists of many different bits and pieces that jointly constitute the document that is displayed in a browser.
  - The main part is generally written in a markup language (HTML, XML).
  - Each embedded document has an associated MIME (Multipurpose Internet Mail Exchange) type.
  - It was developed to provide information on the content of a message body that was sent as part of electronic mail.

2) Draw and explain Multitier architecture of Web Based system.

- Common Gateway Interface (CGI) defines a standard way by which a Web server can execute a program taking user data as input.
- User data come from an HTML forms.
- It specifies the program that is to be executed at the server side, along with parameter values that are filled in by the user.
- Once the form has been completed, the program's name and collected parameter values are sent to the server.
- When the server gets a request it starts the program named in the request and passes its parameter values.
- Program simply does its work and returns the results in the form of a document that is sent back to the user's browser to be displayed.
- As server-side processing of Web documents increasingly requires more flexibility, it should come as no surprise that many Web sites are now organized as a three-tiered architecture consisting of a Web server, an application server and a database.
- The Web server is the traditional Web server application server that runs all kinds of programs and that may or may not access the third tier, consisting of a database.

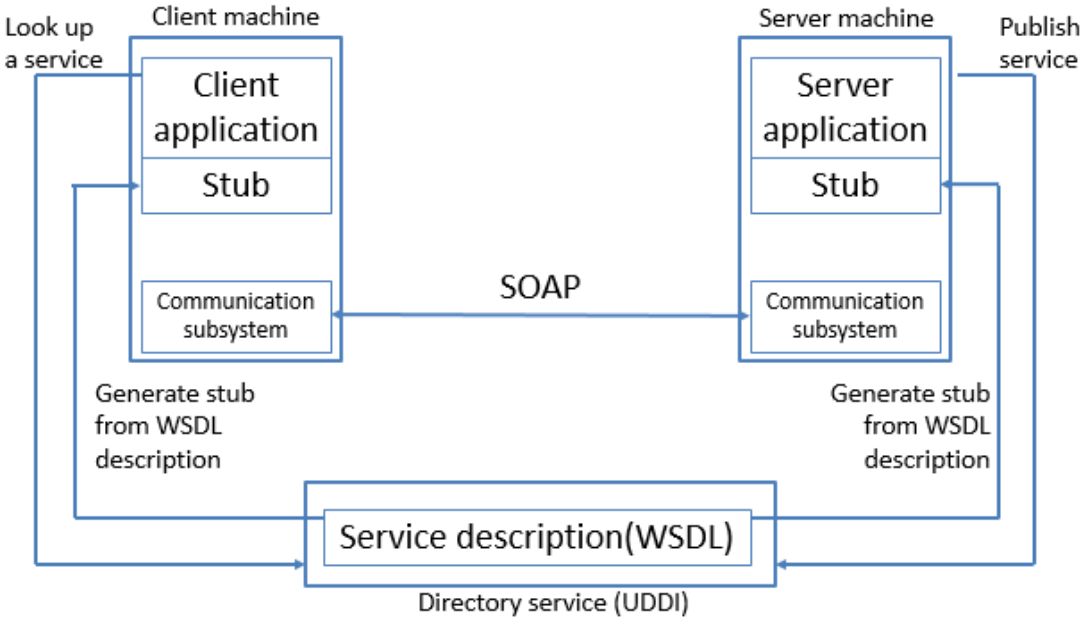


1. Get Request
2. Start process to fetch document
3. Database Interaction
4. HTML Document creator
5. Return Result

3) What is Web services. Explain Architecture of Web Services.

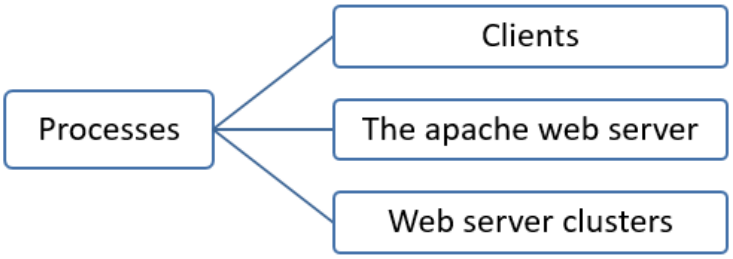
- The basic idea is that some client application can call upon the services as provided by a server application.
- Standardization takes place with respect to how those services are described such that they can be looked up by a client application.
- In addition, we need to ensure that service call proceeds along the rules set by the server application.
- An important component in the Web services architecture is formed by a directory service storing service descriptions.
- Universal Description, Discovery and Integration standard (UDDI) prescribes the layout of a database containing service descriptions that will allow Web service clients to browse for relevant services.

- Services are described by means of the Web Services Definition Language (WSDL).
- A WSDL description contains the precise definitions of the interfaces provided by a service, that is, procedure specification, data types, the (logical) location of services, etc.
- A core element of a Web service is the specification of how communication takes place.
- The Simple Object Access Protocol (SOAP) is used, which is essentially a framework in which much of the communication between two processes can be standardized.



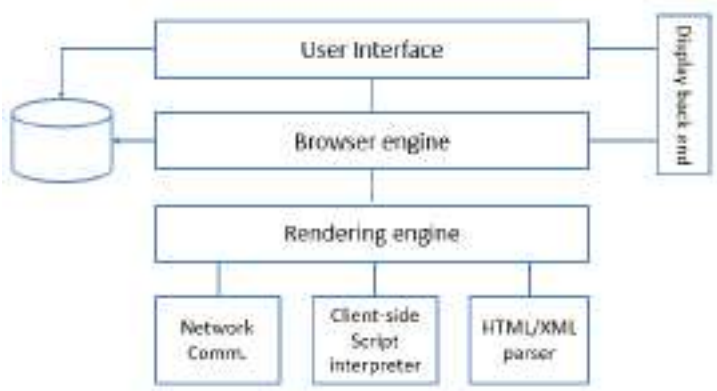
4) Explain the Processes used in Web-based systems and their internal organization.

- The most important processes used in Web-based systems and their internal organization.

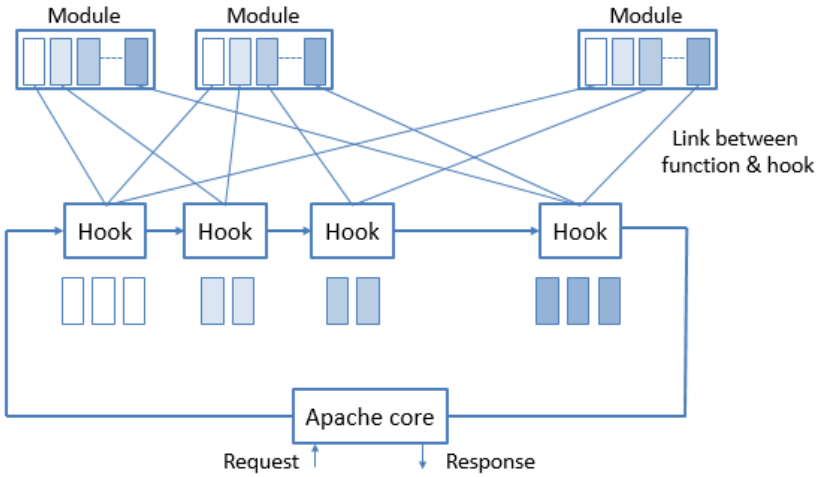


- **Clients**
  - The most important Web client is a piece of software called a Web browser.
  - It enables a user to navigate through Web pages by fetching those pages from servers and subsequently displaying them on the user's screen.
  - The core of a browser is formed by the browser engine and the rendering engine.
  - The rendering engine contains all the code for properly displaying documents.
  - This rendering requires parsing HTML or XML, but may also require script interpreter (for JavaScript).
  - In most case, there is only an interpreter for JavaScript included, but in theory other interpreters may be included as well.

- The browser engine provides the mechanisms for an end user to go over a document, select parts of it, activate hyperlinks etc.

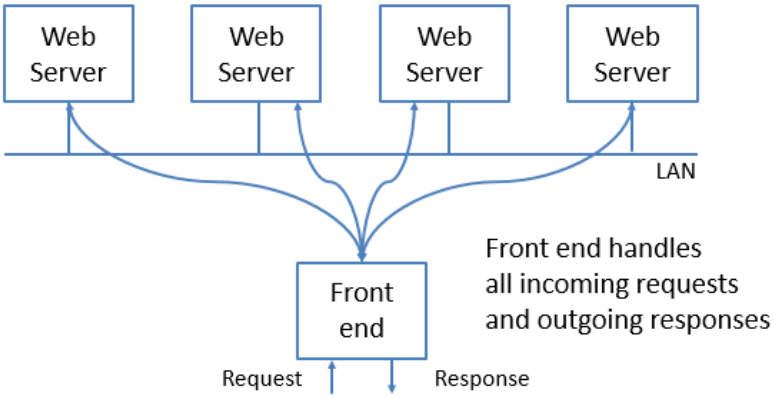


- **Apache web server**
  - Apache Portable Runtime (APR), is a library that provides a platform-independent interface for file handling, networking, locking and threading.
  - Fundamental to this organization is the concept of a hook, which is nothing but a placeholder for a specific group of functions.
  - The Apache core assumes that requests are processed in a number of phases, each phase consisting of a few hooks.
  - Each hook thus represents a group of similar actions that need to be executed as part of processing a request.
  - For example, there is a hook to translate a URL to a local file name.
  - Such a translation will almost certainly need to be done when processing a request.
  - Likewise, there is a hook for writing information to a log, a hook for checking a client's identification, a hook for checking access rights, and a hook for checking which MIME type the request is related to.



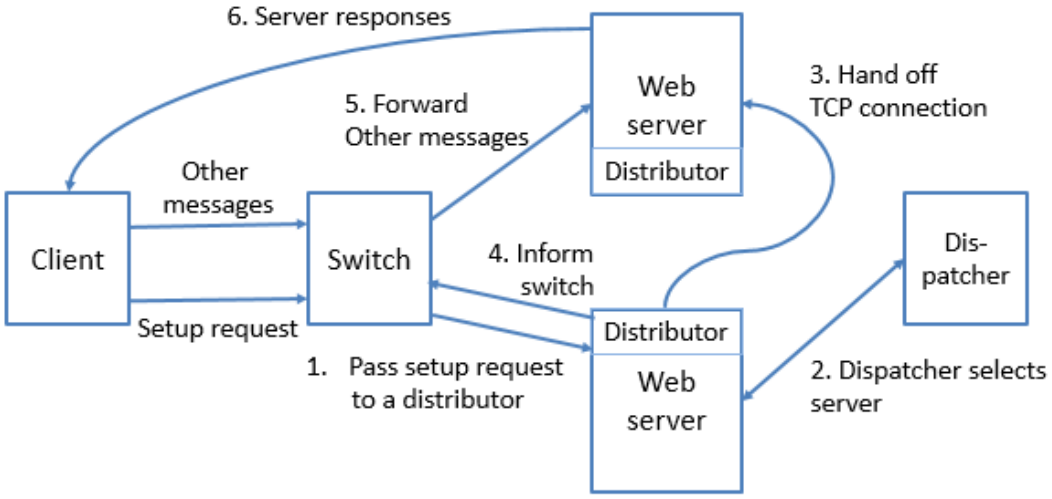
- **Web server clusters**
  - An important problem related to the client-server nature of the Web is that a Web server can easily become overloaded.
  - A practical solution employed in many designs is to simply replicate a server on a cluster of servers.

- This principle is an example of horizontal distribution.
- The design of the front end becomes a serious performance bottleneck.
- Whenever a client issues an HTTP request, it sets up a TCP connection to the server.



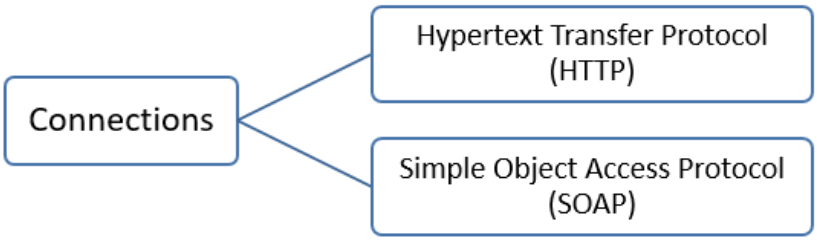
The principle of using a server cluster in combination with a front end to implement a Web service.

- A better approach is to deploy content-aware request distribution by which,
  - The front end first inspects an incoming HTTP request
  - Then decides which server it should forward that request
- In combination with TCP handoff, the front end has two tasks.
  - When a request initially comes in, it must decide which server will handle the rest of the communication with the client.
  - The front end should forward the client's TCP messages associated with the handed off TCP connection.
- The dispatcher is responsible for deciding to which server a TCP connection should be handed off.
- The switch is used to forward TCP messages to a distributor.



A scalable content-aware cluster of web servers

5) Explain Types of Connections used in Web-based systems.

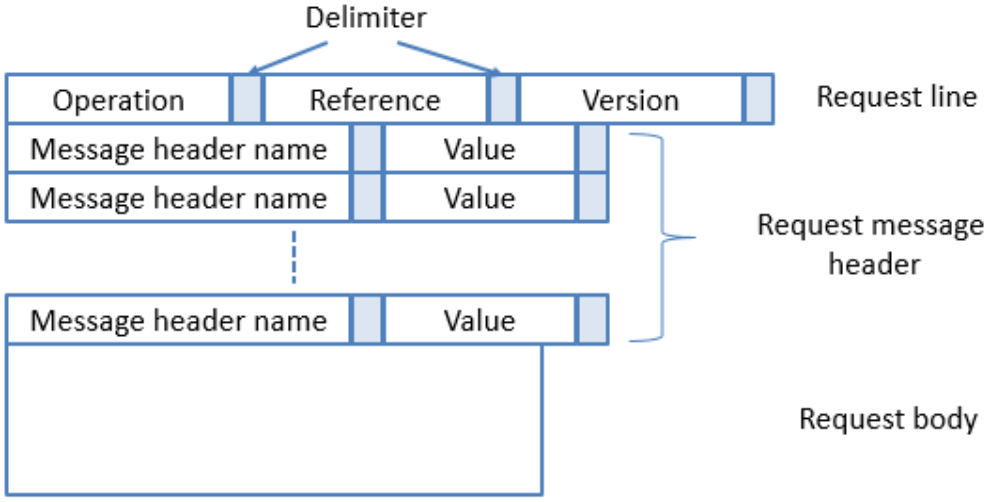


- **Hypertext Transfer Protocol (HTTP)**
- All communication in the Web between clients and servers is based on the Hypertext Transfer Protocol (HTTP).
- HTTP is a relatively simple client-server protocol: a client sends a request message to a server and waits for a response message.
- It does not have any concept of open connection and does not require a server to maintain information on its clients.
  - **HTTP connections**
    - HTTP is based on TCP.
    - Whenever a client issues a request to a server,
    - It first sets up a TCP connection to the server
    - Then sends its request message on that connection.
    - The same connection is used for receiving the response.
    - By using TCP as its underlying protocol, HTTP need not be concerned about lost request and responses.
  - **HTTP methods**
    - A client can request each of these operations to be carried out at the server by sending a request message containing the operation desired to the server.
    - Most commonly used request methods are as follows:

OPERATION	DESCRIPTION
Head	Request to return the header of a document
Get	Request to return a document to the client
Put	Request to store a document
Post	Provide data that are to be added to a document
Delete	Request to delete document

  - **HTTP messages**
    - All communication between a client and server takes place through messages.
    - HTTP recognizes only request and response messages.
    - The request line is mandatory and identifies the operation that the client wants the server to carry out along with a reference to the document associated with that request.
    - A separate field is used to identify the version of HTTP the client is expecting.

- A response message starts with a status line containing a version number and also a three-digit status code.



- **Simple object access protocol (SOAP)**
  - The Simple object access protocol (SOAP) forms the standard for communication with Web services.
  - A SOAP message generally consists of two parts, which are jointly put inside what is called a SOAP envelope.
  - The body contains the actual message.
  - Header is optional, containing information relevant for nodes along the path from sender to receiver.
  - Everything in the envelope is expressed in XML.

6) Explain Naming system used in Distributed Web-based systems

- Web uses a single naming system to refer a documents.
- The names used are called Uniform resource identifiers (URIs).
- Uniform resource locator (URL) is a URI that identifies a document by including information on how and where to access the document.
- A URL is used as a globally unique, location-independent and persistent reference to a document.
- How to access a document is generally reflected by the name of the scheme that is part of the URL such as http, ftp or telnet.
- URL also contains the name of the document to be looked up by that server.
- If the server is referred to by its DNS name, that name will need to be resolved to the server's IP address.
- Using the port number contained in the URL, the client can then contact the server using the protocol named by the scheme, and pass it the document's name that forms the last part of the URL.

- General structure of URLs are as follows.

Scheme	Host name	Pathname
http ://	www.darshan.ac.in	/home/comp/faculty

1. Using only DNS name

Scheme	Host name	Port	Pathname
http ://	www.darshan.ac.in :	80	/home/comp/faculty

2. Combining DNS name with port number

Scheme	Host name	Port	Pathname
http ://	130.37.24.11 :	80	/home/comp/faculty

3. Combining an IP address with port number

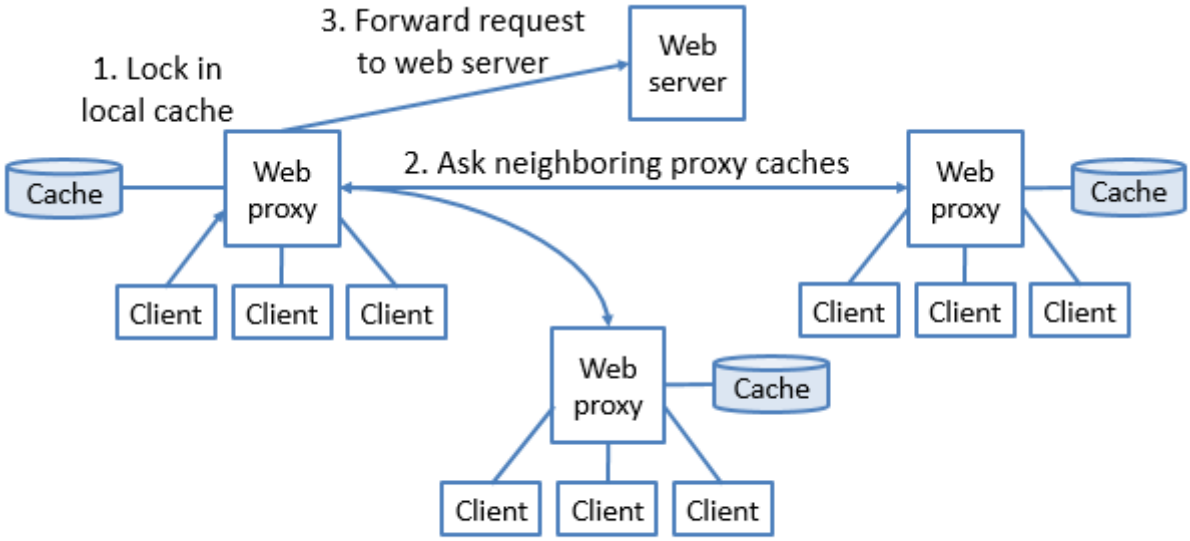
7) Explain Synchronization in Distributed Web-based systems.

- Synchronization has not been much of an issue for most traditional Web- based systems for two reasons.
  - The servers never exchange information with other servers means that there is nothing much to synchronize.
  - The web can be considered as being a read-mostly system.
- Distributed authoring of Web documents is handled through a separate protocol called WebDAV (Web Distributed Authoring and Versioning)
- To synchronize concurrent access to a shared document, WebDAV supports a simple locking mechanism.
- There are two types of write locks.
  1. An exclusive write lock can be assigned to a single client, and will prevent any other client from modifying the shared document while it is locked.
  2. A shared write lock, which allows multiple clients to simultaneously update the document.
- Assigning a lock is done by passing a lock token to the requesting client.
- The server registers which client currently has the lock token.
- Whenever the client wants to modify the document, it sends an HTTP post request to the server along with the lock token.

8) Explain concept of Web proxy caching in Distributed web based system.

- Web proxy accepts requests from local clients and passes these to Web servers.
- When a response comes in, the result is passed to the client.
- The advantage of this approach is that the proxy can cache the result and return that result to another client, if necessary.
- In other words, a Web proxy can implement a shared cache.
- In addition to caching at browsers and proxies, it is also possible to place caches that cover a region, or even a country, thus leading to hierarchical caches.

- Such schemes are mainly used to reduce network traffic.
- But has the disadvantage of incurring a higher latency compared to using non-hierarchical schemes.
- As an alternative to building hierarchical caches, one can also organize caches for cooperative deployment as shown in Figure.
- In cooperative caching or distributed caching, whenever a cache miss occurs at a Web proxy, the proxy first checks a number of neighbouring proxies to see if one of them contains the requested document.
- If such a check fails, the proxy forwards the request to the Web server responsible for the document.
- This scheme is primarily deployed with Web caches belonging to the same organization or institution that are collocated in the same LAN.



The Principle of cooperative caching

9) Explain concept of Replication in Web Hosting system.

- There are essentially three different kinds of aspects related to replication in web hosting systems.

1. **Metric estimation**
   - **Latency metrics:** By which the time is measured for an action to take place for example: Fetching a document.
   - **Spatial metrics:** It mainly consists of measuring the distance between nodes in terms of the number of network-level routing hops or hops between autonomous systems.
   - **Network usage metrics:** It computes consumed bandwidth in terms of the number of bytes to transfer.
   - **Consistency metrics:** It tell us to what extent a replica is deviating from its master copy.
   - **Financial metrics:** It is closely related to the actual infrastructure of the Internet.
2. **Adaptation triggering**
   - Important question that needs to be addressed is when and how adaptations are to be triggered.

- A simple model is to periodically estimate metrics and subsequently take measures as needed.
- Special processes located at the servers collect information and periodically check for changes.

3. **Adjustment Measures**
   - There are essentially only three measures that can be taken to change the behavior of a Web hosting service:
     1. Changing the placement of replicas
     2. Changing consistency enforcement
     3. Deciding on how and when to redirect client requests

- **Replication of web applications**
  - It is complicated to improve performance of Web applications through caching and replication.
  - To improve performance, we can apply full replication of the data stored at the origin server.
  - This scheme works well whenever the update ratio is low and when queries require an extensive database search.
  - Replicating for performance will fail when update ratio is high.
  - Alternative is partial replication in which only a subset of the data is stored at the edge server.
  - The problem with partial replication is that it may be very difficult to manually decide which data is needed at the edge server.

1) Write a note on Security in Distributed System. Explain Goal of Security.

- Security in distributed systems can be divided into two parts.
- One part concerns the communication between users or processes, possibly residing on different machines.
- The principal mechanism for ensuring secure communication is
  - Secure channels
  - Authentication
  - Message integrity
  - Confidentiality
- The other part concerns authorization, which deals with ensuring that a process gets only those access rights to the resources in a distributed system it is entitled to.
- **Goals of security**
  - **Secrecy:** Information within the system must be accessible only to authorized users.
  - **Privacy:** Information given to the users must be used only for the purpose for which was given.
  - **Authenticity:** The user must be able to verify that the data obtained is from expected sender only.
  - **Integrity:** Information must be protected from unauthorized access.

2) Define Threat. Explain types of Threat.

- Threat is a possible danger that might exploit a vulnerability to breach security and thus cause possible harm.
- **Type of threats**
  - **Interception:** Unauthorized user gaining access to a service or data. E.g. eavesdropping, illegal copying.
  - **Interruption:** Services or data becoming unavailable, unusable, destroyed. E.g. intentional file corruption, denial of service attacks.
  - **Modification:** Unauthorized changing of data or service so that it no longer adheres to its original specification.
  - **Fabrication:** Additional data or activity is generated that would normally not exist. E.g. adding entry to password file or database, breaking into a system by replaying previously sent messages.

3) Define Attacks. Explain types of Attacks.

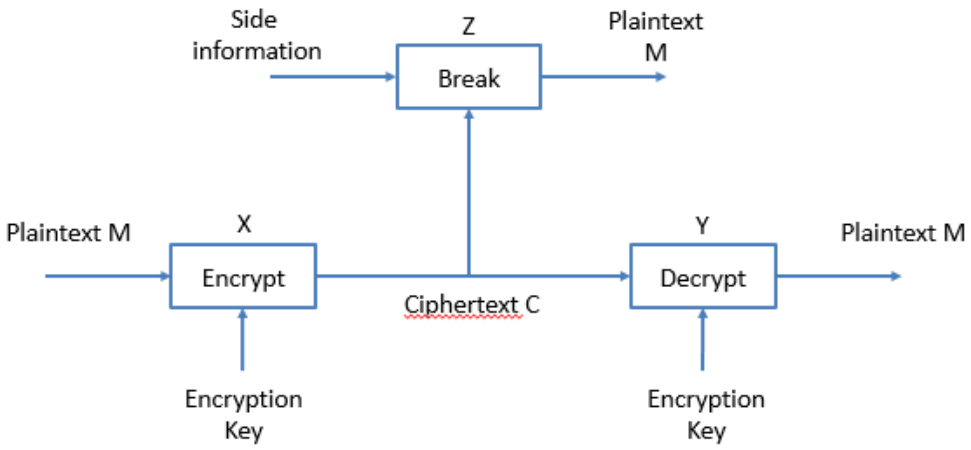
- Attack is any attempt to destroy, expose, alter, disable, steal or gain unauthorized access to or make unauthorized use of an asset.
- **Type of attacks**
  - **Passive attack**
    - Intruder access unauthorized information from a computer system but cannot cause harm to the system.
    - **Browsing:** Intruders here attempt to read stored files, traverse message packet on the network, access other process memory, etc.
    - **Inferencing:** The intruder records and analyzes past activities and access methods and uses this information to draw inferences.

- **Masquerading:** An intruder Masquerades as an authorized user or a program to gain access to unauthorized data or resource.

- **Active attack**
  - **Virus:** is a small computer program that needs to be executed by either running it or having it loaded from boot sector of a disk.
  - **Worm:** is a small piece of software the uses computer network and security holes to replicate itself.
  - **Logic bomb:** is a piece of code intentionally inserted into a software system that will set off a malicious function when specified conditions are met.
  - **Integrity attack:** An intruder can change the message while it is traveling in the communication channel and the receiver may interpret it as original message.
  - **Authenticity attack:** An intruder can illegally connect to computer network, impersonate and insert bogus message with valid address in the system. These will then be delivered as genuine message.
  - **Denial attack:** An intruder might partly or completely block communication path between two processes.
  - **Delay attack:** An intruder can delay the message delivery that can make it useless to receive if it is received late.
  - **Replay attack:** An intruder retransmit an old message that is accepted as new message by the receiver.

4) What is Cryptography? Explain Symmetric and Asymmetric Cryptosystems.

- 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:** The process of transforming intelligible information (plaintext) into unintelligible form (cipher text).
  - **Decryption:** The process of transforming the information from cipher text to plaintext.
- The encryption algorithm has the following form: C = E(P,Ke)

  Where,

  P = plaintext to be encrypted

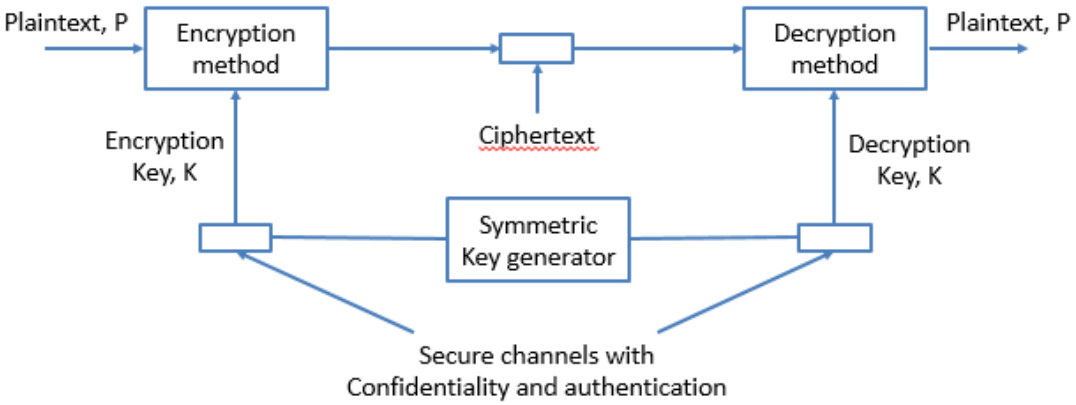
Ke = encryption key
C = resulting cipher text

- The decryption algorithm is performed by the same matching function which has the following form:

P = D(C,Kd)

Where,
C = cipher text to be decrypted
Kd = decryption key
P = resulting plaintext

- **Symmetric cryptosystem**
  - A symmetric cryptosystem uses the same key for both encryption and decryption.
  - It is necessary that the key should be easily alterable if required and is always kept secret.
  - This implies that the key is known only to authorized users.
  - Symmetric cryptosystems are also called as shared key or private key cryptosystems since both sender and receiver share the same key.
  - P=Dk(Ek(P))

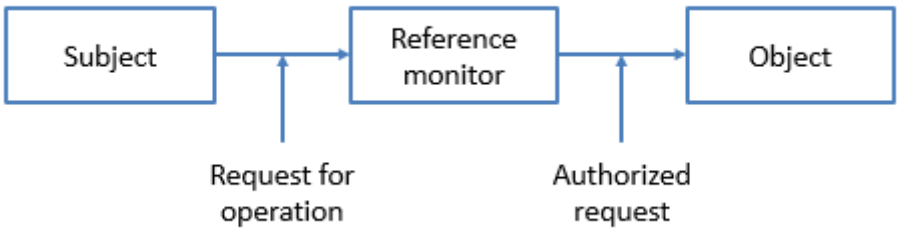


Secret key Distribution

- **Asymmetric cryptography**
  - In asymmetric cryptosystem, the keys used for encryption and decryption are different but they form a unique pair.
  - There are separate keys for encryption (Ke) and decryption (Kd).
  - P = DKd(EKe(P))
  - One key in the asymmetric cryptosystem is kept private while the other one is made public.
  - Hence these types of cryptosystems are referred to as public key systems.

5) Write a note on Access control in Distributed system.

- 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.

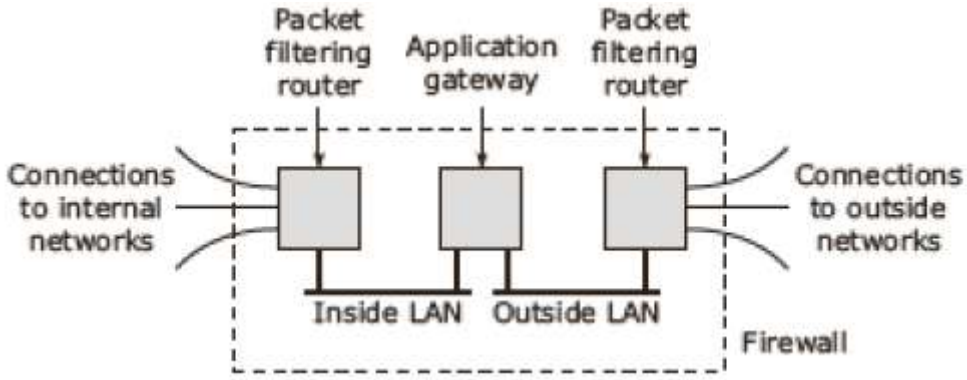


- The object may be an abstract entity like a process, file, database, semaphore, tree data structure, or a physical entity like CPU, memory segment, printer, tape drive, network site, etc.
- Each object has a unique name that differentiates it from others in the system and it is referenced by this unique name.
- The object is associated with a type to determine the type of operations that can be performed on it.
- Subjects are processes that act on behalf of users or they could be objects that need services of other objects to carry out their tasks.
- A subject in other words is an active entity whose access to objects needs to be controlled.
- 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.
- The complete set of access rights defines which subjects can perform what operations on which objects.

6) Explain Concept of Firewalls in Distributed System.

- 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.
- All outgoing and incoming packets are routed through a special computer and inspected before they are passed.
- 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.
- The firewall itself should be thoroughly protected against any security threat and should never fail.
- 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 also includes or works with a proxy server that makes network requests on behalf of workstation users.
- 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.
- It makes the decision of whether or not to pass a network packet based on the source and destination addresses in the packet.
- The other type of firewall is the application-level gateway.
- This type of firewall inspects the contents of the incoming and outgoing packet.
- For example, a mail gateway can discard incoming and outgoing messages exceeding a certain size.



7) Explain Concept of Key Management in Distributed System.

- In different cryptosystems keys were readily available.
- In the case of authentication, each party shares a key with the KDC.
- The main problem is how to distribute the keys securely.
- The second issue is to I revoke keys that are compromised or invalidated.
- **Key establishment**
  - First, let us consider how to set up a session's key.
  - When Anant wants to have a secure channel of communication with Balwant.
  - He uses Balwant's public key to initiate communication.
  - If Balwant accepts, he generates a session's key and sends it to Anant after encrypting it with Anant's public key.
  - Similarly, a session key can be generated and distributed if both Anant and Balwant share a secret key.
  - It means that they both have the means to establish a secure channel.
  - A secure channel is necessary to share the secret key with the help of KDC.
- **Issues in Key Distribution**

  Key distribution in symmetric cryptosystem
  - If two users on different nodes want to communicate using a symmetric cryptosystem, they must first share the encryption / decryption key that needs to be transmitted over the physical insecure medium.
  - Hence, the key must be encrypted before transmission.

- Usually, a small number of keys are distributed earlier using a server process managed by a key distribution centre (KDC). This is a trusted entity and shared by all communicating users.
- The various implementation approaches are:
- **Centralized approach**
  - In this approach, a single KDC maintains a table of secret keys for each user.
  - A user A makes a request to the KDC (in plaintext with its userid) indicating that it wants a secure communicating channel with user B.
  - The KDC extracts the key value corresponding to the userid and creates a secret key for secure communication between user A and B and sends it to user /I.
  - On receiving the message, user A decrypts the key after confirming that it is matching with the original request.
  - Now the user uses this key and sends a message to user B who decrypts with its private key and retrieves the secret key.
  - Now both users A and B use the secret key for message transmission.
  - The centralized approach is simple and easy to implement.
  - The drawbacks are poor reliability and performance bottleneck due to a single KDC.
- **Fully Distributed**
  - The KDC resides at each node in the distributed system and the secret keys are distributed well in advance.
  - Thus, all KDCs can communicate with each other, the KDC has a table of secret keys with private keys of all other KDCs For a system with n nodes, each KDC has n - 1 keys and a total of n(n - 1)/2 key pairs exist in the system.
  - To establish a secure logical communication channel, user A makes a request to the local KDC (plaintext form).
  - The KDC extracts the keys corresponding to the users and creates a secret key for secure communication and sends it to user A and B.
  - On receiving the message, user A verifies that the message matches with the original and keeps the key for future use.
  - Similarly, user B decrypts the message with its private key and extracts the secret key.
  - Now user B initiates the authentication procedure, authenticates user A and then, proceeds with secure communication.
- **Partially Distributed**
  - Here the nodes are partitioned into regions and each region has a KDC.
  - The prior distribution of secret keys allows each KDC to communicate securely with each user of its own region and with KDCs of other regions.
  - Each KDC has a table of secret keys that contains private keys of all users of its own region and of all other KDCs.
  - The distribution of a key for establishing a secure communication channel depends on the location of the two users: whether they belong to the same region or to two different regions.
  - In the former case, the key distribution is similar to the centralized approach.

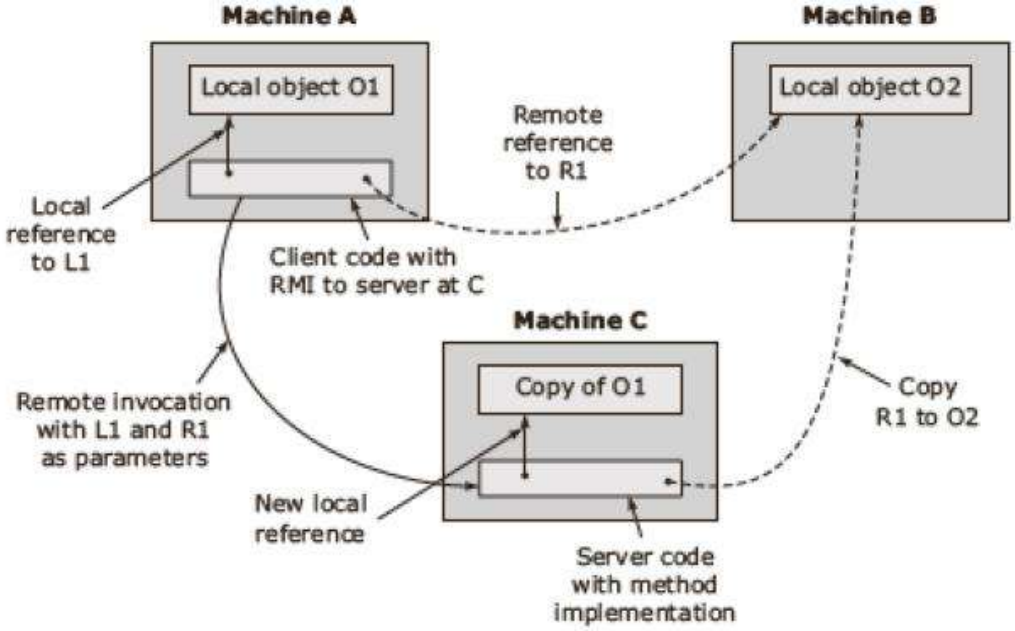
- If the users belong to different regions, the key distribution procedure is exactly similar to the fully distributed approach.

Key Distribution in Asymmetric system

- In an asymmetric cryptosystem, only public keys are distributed which are not kept secret and hence can be transmitted over an insecure communication channel.
- In an asymmetric crypto system, the key distribution procedure involves an authentication procedure to prevent an intruder from generating a pair of keys.
- A common method is to use a public- key manager (PKM) that maintains a directory of public keys of all users in the system.
- The public key is known to all users, while the secret key is known only to the PKM.
- Suppose user A wants to communicate with user B. then it sends a request message (cipher text) to the PKM to establish a secure logical communication channel with user B.
- PKM decrypts the message, extracts the public key corresponding to the user IDs and sends it (cipher text form) to user A.
- On receiving the message, user A decrypts the message and confirms that the message is a reply for its request.
- Next, it sends a message to user B who decrypts the message and understands that it is a request for communication.
- To authenticate the correct public key, user B sends a request to the PKM, gets the public key and authenticates user A.
- This allows regular communication to start.
- Authenticated distribution of public keys takes place through public-key certificates.
- These certificates consist of public keys and the identities to which the keys are associated.
- A certification authority signs the <public key, identified> pair and places it on the certificate.
- A private key (Kca -) of the certification authority is used to sign the certificate. Its corresponding public key (Kca +) is known.
- If a client wishes to verify that the public key found in the certificate belongs to the identified entity, then it uses the public key of the certification authority to verify the certificate's signature.
- If the signature on the certificate matches the <public key, identified pair>, then it accepts that the public key belongs to that entity.

1) Case study of Java RMI.

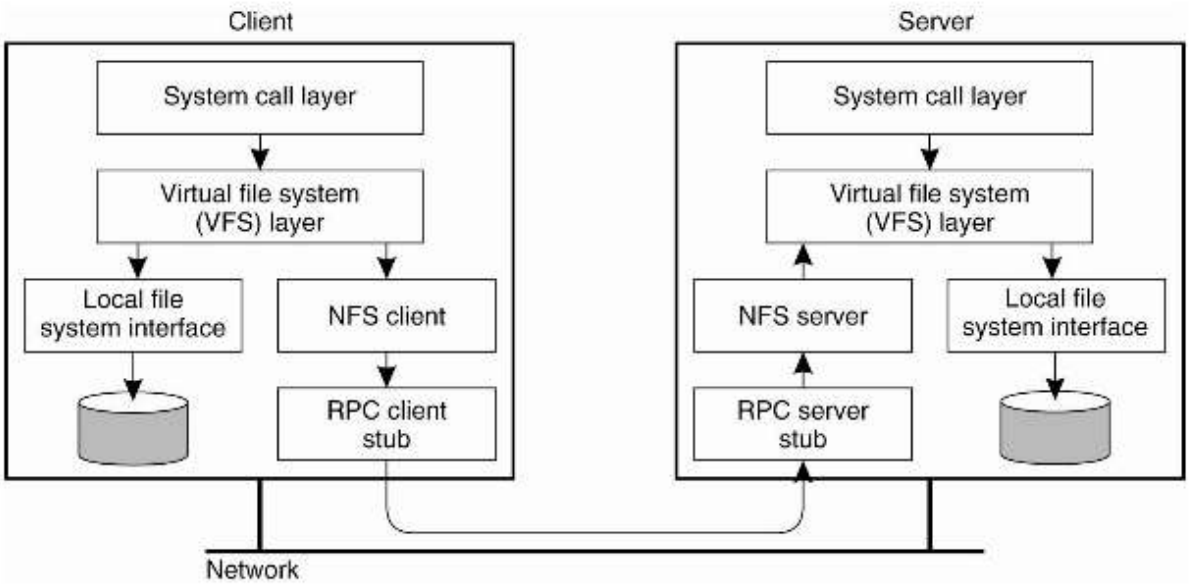
- Java hides the difference between local and remote method invocation from the user.
- After marshalling the type of object, it is passed as a parameter to RMI in Java.



- The reference to remote object consists of the network address and endpoint of server.
- It also contains the local ID for the actual object in the server address space.
- Each object in Java is an instance of a class, which contains the implementation of one or more interfaces.
- A java remote object is built from two classes (i) server class and (ii) client class.
- **Server class:**
  - This class contains the implementation of server side code, i.e objects which run on the server.
  - It also consists of a description of the object state and implementation of methods which operate on that state.
  - The skeleton on the server side stub is generated from the interface specification of the object.
- **Client Class:**
  - This class contains the implementation of client side code and proxy.
  - It is generated from the object interface specification.
  - The proxy basically converts each method call into message that is sent to the server side implementation of the remote object.
- Communication is set up to the server and is cut down when a call is complete.
- The proxy state contains the server's network address, end to the server and local ID of the object.
- Hence, a proxy consists of sufficient information to invoke the methods of a remote object.
- Proxies are serializable in Java, marshalled and sent as series of bytes to another process where it is unmarshalled, and can invoke methods in remote objects.

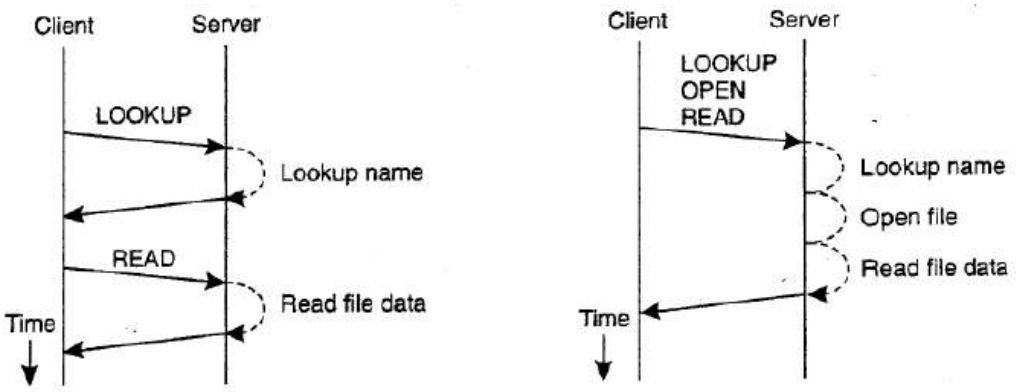
2) Case study of SUN Network File System.

- The basic idea behind NFS is that each file server provides a standardized view of its local file system.
- A client accesses the file system using the system calls provided by its local operating system.
- However, the local UNIX file system interface is replaced by an interface to the Virtual File System (VFS).
- With NFS, operations on the VFS interface are either passed to a local file system, or passed to a separate component known as the NFS client, which takes care of handling access to files stored at a remote server.
- In NFS, all client-server communication is done through RPCs.
- The NFS client implements the NFS file system operations as RPCs to the server.
- On the server side the NFS server is responsible for handling incoming client requests.
- The RPC stub unmarshals requests and the NFS server converts them to regular VFS file operations that are subsequently passed to the VFS layer.



- Again, the VFS is responsible for implementing a local file system in which the actual files are stored.
- The file system model offered by NFS is almost the same as the one offered by UNIX-based systems.
- Files are treated as uninterpreted sequences of bytes.
- They are hierarchically organized into a naming graph in which nodes represent directories and files.
- NFS also supports hard links as well as symbolic links, like any UNIX file system.
- To access a file, a client must first look up its name in a naming service and obtain the associated file handle.
- Each file has a number of attributes whose values can be looked up and changed.

- **RPC's in NFS**
  - Every NFS operation can be implemented as a single remote procedure call to a file server.
  - In order to read data from a file for the first time, a client normally first had to look up the file handle using the lookup operation, after which it could issue a read request.
  - This approach required two successive RPCs.



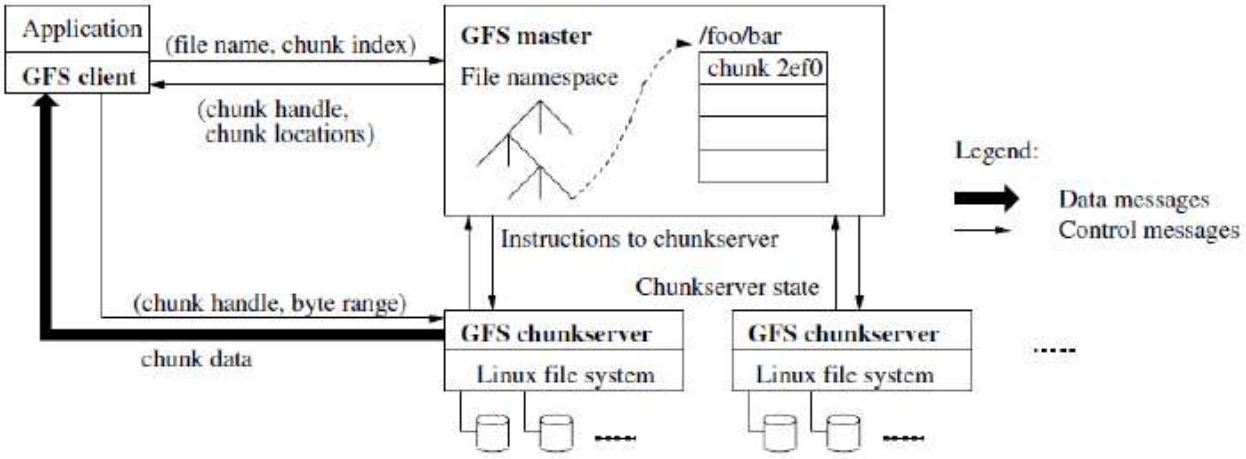
  - The drawback became apparent when considering the use of NFS in a wide-area system.
  - To overcome such problems, NFSv4 supports compound procedures by which several RPCs can be grouped into a single request.
  - In the case of version 4, it is also necessary to open the file before reading can take place.
  - After the file handle has been looked up, it is passed to the open operation, after which the server continues with the read operation.
  - The overall effect in this example is that only two messages need to be exchanged between the client and server.

3) Case study on Google File System.

- The Google File System is a scalable distributed file system for large distributed data-intensive applications.
- It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.
- **Assumptions/Features**
  - The system is built from many inexpensive commodity components that often fail.
  - It constantly monitors itself and detect, tolerate, and recover promptly from component failures on a routine basis.
  - The system stores a modest number of large files.
  - We expect a few million files, each typically 100 MB or larger in size.
  - The workloads primarily consist of two kinds of reads: large streaming reads and small random reads.
    - In large streaming reads, individual operations typically read hundreds of KBs, more commonly 1 MB or more.
    - A small random read typically reads a few KBs at some arbitrary offset.
  - The workloads also have many large, sequential writes that append data to files.

- The system must efficiently implement well-defined semantics for multiple clients that concurrently append to the same file.
- Atomicity with minimal synchronization overhead is essential.
- High sustained bandwidth is more important than low latency.

- **Architecture**
  - A GFS cluster consists of a single master and multiple chunk servers and is accessed by multiple clients.

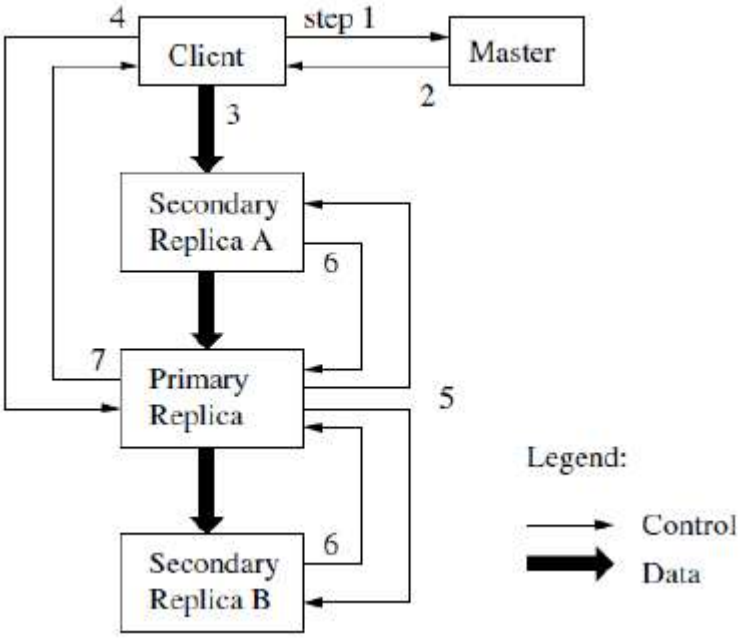


  - Each of these is typically a commodity Linux machine running a user-level server process.
  - Files are divided into fixed-size chunks.
  - Each chunk is identified by an immutable and globally unique 64-bit chunk handle assigned by the master at the time of chunk creation.
  - Chunk servers store chunks on local disks as Linux files and read or write chunk data specified by a chunk handle and byte range.
  - For reliability, each chunk is replicated on multiple chunk servers.
  - By default, we store three replicas, though users can designate different replication levels for different regions of the file namespace.
  - The master maintains all file system metadata.
  - This includes the namespace, access control information, the mapping from files to chunks, and the current locations of chunks.
  - It also controls system-wide activities such as chunk lease management, garbage collection of orphaned chunks, and chunk migration between chunk servers.
  - The master periodically communicates with each chunk server in Heartbeat messages to give it instructions and collect its state.
  - Clients interact with the master for metadata operations, but all data-bearing communication goes directly to the chunk servers.
  - The interactions for a simple read with reference to figure is as follows:
    - First, using the fixed chunk size, the client translates the file name and byte offset specified by the application into a chunk index within the file.
    - Then, it sends the master a request containing the file name and chunk index.

- The master replies with the corresponding chunk handle and locations of the replicas.
- The client caches this information using the file name and chunk index as the key.
- The client then sends a request to one of the replicas, most likely the closest one.
- The request specifies the chunk handle and a byte range within that chunk.
- Further reads of the same chunk require no more client-master interaction until the cached information expires or the file is reopened.
- In fact, the client typically asks for multiple chunks in the same request and the master can also include the information for chunks immediately following those requested.

- **Chunk Size**
  - Chunk size is one of the key design parameters. We have chosen 64 MB, which is much larger than typical file system block sizes.
  - A large chunk size offers several important advantages.
  - It reduces clients' need to interact with the master because reads and writes on the same chunk require only one initial request to the master for chunk location information.
  - Since on a large chunk, a client is more likely to perform many operations on a given chunk, it can reduce network overhead by keeping a persistent TCP connection to the chunk server over an extended period of time.
  - It reduces the size of the meta-data stored on the master.
  - A large chunk size offers several disadvantages
  - A small file consists of a small number of chunks, perhaps just one.
  - The chunk servers storing those chunks may become hot spots if many clients are accessing the same file.
- **Metadata**
  - The master stores three major types of metadata: the file and chunk namespaces, the mapping from files to chunks, and the locations of each chunk's replicas.
  - All metadata is kept in the master's memory.
  - The first two types (namespaces and file-to-chunk mapping) are also kept persistent by logging mutations to an operation log stored on the master's local disk and replicated on remote machines.
  - The master does not store chunk location information persistently.
  - Instead, it asks each chunk server about its chunks at master startup and whenever a chunk server joins the cluster.
- **Leases and Mutation Order**
  - Mutation is an operation that changes the contents or metadata of a chunk such as a write or an append operation.
  - Each mutation is performed at all the chunk's replicas.
  - We use leases to maintain a consistent mutation order across replicas.
  - The master grants a chunk lease to one of the replicas, which we call the primary.
  - The primary picks a serial order for all mutations to the chunk.

- All replicas follow this order when applying mutations.
- Thus, the global mutation order is defined first by the lease grant order chosen by the master, and within a lease by the serial numbers assigned by the primary.



- **Creation, Re-replication, Rebalancing**
  - Chunk replicas are created for three reasons: chunk creation, re-replication, and rebalancing.
  - When the master creates a chunk, it chooses where to place the initially empty replicas. It considers several factors.
    1. Place new replicas on chunk servers with below-average disk space utilization. Over time this will equalize disk utilization across chunk servers.
    2. Limit the number of "recent" creations on each chunk server.
    3. Spread replicas of a chunk across racks.
  - The master re-replicates a chunk as soon as the number of available replicas falls below a user specified goal.
  - The master picks the highest priority chunk and "clones" it by instructing some chunk server to copy the chunk data directly from an existing valid replica.
  - The new replica is placed with goals similar to those for creation: equalizing disk space utilization, limiting active clone operations on any single chunk server, and spreading replicas across racks.
  - The master rebalances replicas periodically: it examines the current replica distribution and moves replicas for better disks pace and load balancing