1.1 Definition of a Distributed System

1. A distributed system is one in which components located at networked computers communicate and co-ordinate their actions only by passing messages.
2. A distributed system is collection of independent entities that co-operate to solve a problem that cannot be individually solved.
3. Tenenbaum's definition : A distributed system is a collection of independent computers that appears to its users a single coherent system.

- Each node of distributed computing system is equipped with a processor, a local memory and interfaces. Communication between any pair of nodes is realized only by message passing as no common memory is available.
- Usually, distributed system are asynchronous i.e., they do not use a common clock and do not impose any bounds on relative processor speeds or message transfer times.
- Differences between the various computers and the ways in which they communicate are mostly hidden from users.
- Fig. 1.1.1 shows distributed system organized as middleware.

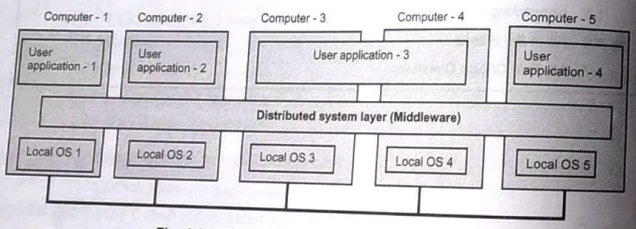

Fig. 1.1.1 Distributed system organized as middleware

- To support heterogeneous, computers and networks while offering a single system view, distributed system are often organized by means of a layer of software that is logically placed between a higher level consisting of users and applications and layer underneath consisting of OS.
- Middleware is software which lies between an operating system and the applications running on it. Distributed system is sometimes called as middleware.
- An example of a distributed system would be the world wide web where there are multiple components under the hood that help browsers display content but from a user's point of view, all they are doing is accessing the web via a browser.

- Users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction takes place.
- Each host executes components and operates a distribution middleware. Middleware enables the components to co-ordinate their activities. Users percieve the system as a single, integrated computing facility.
- A distributed system can consist of any number of possible configurations, such as mainframes, personal computers, workstations, minicomputers and so on.
- **Significant consequences of distributed systems :**
  1. **Concurrency :** The capacity of the system to handle shared resources can be increased by adding more resources to the network. p and q are concurrent if either p can happen before q or q can happen before p, thus having interleaving semantics.
  2. **No global clock :** The only communication is by sending messages through a network. Not possible to synchronize many computers on a network and guarantee synchronization over time, thus events are logically ordered. Not possible to have a process that can be aware of a single global state.
  3. **Independent failures :** The programs may not be able to detect whether the network has failed or has become unusually slow. Running processes may be unaware of other failure with context. Failed processes may go undetected. Both are due to processes running in isolation.

1.1.1 Need of Distributed System

- Share resource is main motivation of the distributed systems.
- The term "resource" is a rather abstract one, but it best characterizes the range of things that can usefully be shared in a networked computer system.
- Sharing of resource extends from hardware components such as disks and printers to software - defined entities such as files, databases and data objects of all kinds.
- It also includes the stream of video frames and audio connection that a mobile phone call represents.

1.1.2 Advantages of DS over Centralized Systems

1. Economics : A collection of microprocessors offer a better price/performance than mainframes. Low price/performance ratio : Cost effective way to increase computing power.
2. Speed : A distributed system may have more total computing power than a mainframe. large powerful computer

3. Inherent distribution : Some applications are inherently distributed e.g. a supermarket chain.
4. Reliability : If one machine crashes, the system as a whole can still survive. Higher availability and improved reliability.
5. Incremental growth : Computing power can be added in small increments. Modular expandability.

1.1.3 Disadvantages of Distributed System

1. Software : Difficult to develop software for distributed systems.
2. Network : Saturation, lossy transmissions.
3. Security : Easy access also applies to secret data.

1.1.4 Comparison between Centralised System and Distributed Systems

Sr. No.	Centralised system	Distributed systems
1.	Centralized systems have non-autonomous components.	Distributed systems have autonomous components.
2.	Centralized systems are often built using homogeneous technology.	Distributed systems may be built using heterogeneous technology.
3.	Multiple users share the resources of a centralized system at all times.	Distributed system components may be used exclusively and executed in concurrent processes.
4.	Centralized systems have a single point of control and of failure.	Distributed systems have multiple points of failure.

1.1.5 Comparison between Parallel Systems and Distributed Systems

Sr. No.	Parameters	Parallel systems	Distributed systems
1.	Memory	Tightly coupled shared memory UMA, NUMA	Distributed memory Message passing, RPC and/or used of distributed shared memory
2.	Control	Global clock control SIMD, MIMD	No global clock control. Synchronization algorithms needed
3.	Processor Interconnection	Bus, mesh, tree, mesh of tree and hypercube network	Ethernet(bus), token ring and SCI (ring), switching network
4.	Main focus	Performance scientific computing	Performance reliability/availability Information/resource sharing

1.2 Goals of a Distributed System

- The main goal of a distributed system is to connect users and resources in a transparent, open and scalable way.
  1. Making resources available
  2. Distribution transparency
  3. Openness
  4. Scalability

1.2.1 Making Resources Available

- Support user access to remote resources like printers, data files, web pages, CPU cycles etc. and share them in a controlled and efficient way.
- Connecting users and resources also makes it easier to collaborate and exchange information. For example : Internet for exchanging files, mail, documents, audio and video.
- Security is becoming increasingly important.
  i. Little protection against evesdropping or intrusion on communication.
  ii. Tracking communication to build up a preference profile of a specific user.
- A good reason to build a distributed system is to make them distributed resources available as they would belong to a single system. By making interaction possible between users and resources, distributed systems are enablers of sharing, information exchange, collaboration.

1.2.2 Transparency

- Software hides some of the details of the distribution of system resources. It makes the system more users friendly.
- A distributed system that appears to its users and applications to be a single computer system is said to be **transparent**.
- Users and applications should be able to access remote resources in the same way they access local resources.
- The distributed systems should be perceived as a single entity by the users or the application programmers rather than as a collection of autonomous systems, which are co-operating.
- The users should be unaware of where the services are located and also the transferring from a local machine to a remote one should also be transparent.
- To make certain aspects of distributed system invisible to the application programmer so that they need only be concerned with the design of their particular application.

- The implication of transparency is a major influence on the design of the system software.
- 8 forms of transparency are Access, Location, Concurrency, Replication, Failure, Mobility, Performance and Scaling.
- Network transparency is access and location transparency.

1. ***Access transparency*** : Using identical operations to access local and remote resources, e.g. Hyperlink in web page.
2. ***Location transparency*** : Resources to be accessed without knowledge of their location, e.g. URL.
3. ***Concurrency transparency*** : Several processed operate concurrently using shared resources without interference with between them.
4. ***Replication transparency*** : Multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers, e.g. Web cache.
5. ***Failure transparency*** : Users and applications to complete their tasks despite the failure of hardware and software components, e.g. email.
6. ***Mobility transparency*** : Movement of resources and clients within a system without affecting the operation of users and programs, e.g. mobile phone.
7. ***Performance transparency*** : Allows the system to be reconfigured to improve performance as loads vary.
8. ***Scaling transparency*** : Allows the system and applications to expand in scale without change to the system structure or the application algorithms.

Sr. No.	Transparency	Description
1.	Access	Hide differences in data representation and how a resource is accessed.
2.	Location	Hide where a resource is located.
3.	Migration	Hide that a resource may move to another location.
4.	Relocation	Hide that a resource may be moved to another location while in use.
5.	Replication	Hide that a resource may be shared by several competitive users.
6.	Concurrency	Hide that a resource may be shared by several competitive users.
7.	Failure	Hide the failure and recovery of a resource.
8.	Persistence	Hide whether a (software) resource is in memory or on disk.

1.2.3 Openness

- An open distributed system offers services according to standard rules that describe the syntax and semantics of those services. In other words, the interfaces to the system are clearly specified and freely available.
- The openness of distributed system is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs.
- Open systems are characterized by the fact that their key interfaces are published.
- It is based on a uniform communication mechanism and published interfaces for access to shared resources.
- It can be constructed from heterogeneous hardware and software.
- Openness is concerned with extensions and improvements of distributed systems. Detailed interfaces of components need to be published. New components have to be integrated with existing components.
- Differences in data representation of interface types on different processors (of different vendors) have to be resolved.
- One of the important features of distributed systems is openness and flexibility :
  i. Every service is equally accessible to every client (local or remote);
  ii. It is easy to implement, install and debug new services;
  iii. Users can write and install their own services.
- Interface Definition/Description Languages (IDL) used to describe the interfaces between software components, usually in a distributed system. Definitions are language and machine independent. It support communication between systems using different OS/programming languages; e.g. a C++ program running on windows communicates with a Java program running on UNIX. The communication is usually RPC-based.

Open Systems Support

- **Interoperability :** The ability of two different systems or applications to work together. A process that needs a service should be able to talk to any process that provides the service. Multiple implementations of the same service may be provided, as long as the interface is maintained.
- **Portability :** An application designed to run on one distributed system can run on another system which implements the same interface.
- **Extensibility** : Easy to add new components and features.

1.2.4 Scalability

- Scalability refers to the capability of a system to adapt to increased service load. Distributed operating system should be designed to easily cope with the growth of nodes and users in the system.
- The system should remain efficient even with a significant increase in the number of users and resources connected :
  1. Cost of adding resources should be reasonable;
  2. Performance loss with increased number of users and resources should be controlled;
  3. Software resources should not run out (number of bits allocated to addresses, number of entries in tables, etc.)
- Some guiding principles for designing scalable distributed systems are as follows :
  1. Avoid centralized entities.
  2. Avoid centralized algorithms.
  3. Perform most operations on client workstations.
- The design of scalable distributed systems presents the following challenges :
  1. Controlling the cost of resources or money.
  2. Controlling the performance loss.
  3. Preventing software resources from running out.
  4. Avoiding preformance bottlenecks.
- Controlling the cost of physical resources i.e. servers and users.
- Controlling the performance loss : DNS hierarchic structures scale better than linear structures and save time for access structured data.
- Preventing software resources running out : Internet 32 - bit addresses run out soon. 128 - bit one gives extra space in messages.

Avoiding performance bottlenecks : DNS name table was kept in a single master file partitioning between servers.

1.2.5 Pitfalls

- False assumptions made by first time developer :
  a) The network is reliable.
  b) The network is secure.
  c) The network is homogeneous.
  d) The topology does not change.
  e) Latency is zero.
  f) Bandwidth is infinite.
  g) Transport cost is zero.
  h) There is one administrator.

1.3 Types of Distributed Systems

- Following are the types of distributed systems.
  1. Distributed computing systems
  2. Distributed information systems
  3. Distributed pervasive systems

Types of distributed systems	Examples
Distributed computing systems	Clusters Grids Clouds
Distributed information systems	Transaction processing systems Enterprise application integration
Distributed pervasive systems	Home systems Health care systems Sensor networks

1.3.1 Distributed Computing Systems

- Distributed systems used for high-performance computing task.
  1. **Cluster computing**
- Hardware consists of a collection of similar workstations or PCs, closely connec by means of a high-speed local-area network; each node runs the same opera system. Computer cluster is a group of linked computers, working together clo thus in many respects forming a single computer.
- Clustering allows us to run applications on several parallel servers. The loa distributed across different servers and even if any of the servers fails application is still accessible via other cluster nodes. Fig. 1.3.1 shows cl computing system.
- In virtually all cases, cluster computing is used for parallel programming in a single program is run in parallel on multiple machines. Example of a c computer : Linux-based Beowulf clusters.

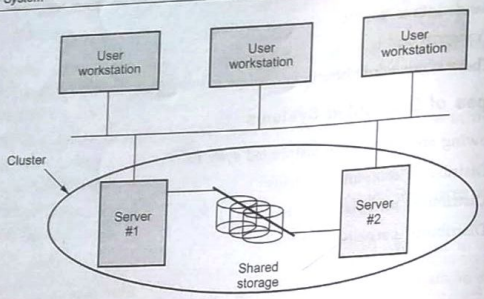

Fig. 1.3.1 Cluster computing

- Cluster architectures are quite flexible and as a result, it is possible to mix both shared and distributed storage when necessary. Such an architecture would strongly suit an enterprise with a corporate headquarters where large data warehouses are managed and with offices around the globe that operate autonomously on a day-to-day basis.
- Functions of master node :
  a) Master node handles the allocation of nodes to a particular parallel program.
  b) It also maintains a batch queue of submitted jobs.
  c) System interface is provided to the user.
  d) The master runs the middleware needed for the execution of programs and management of the cluster.

2. Grid computing

- Grid computing is a distributed computing system where a group of computers are connected to create and work as one large virtual computing power, storage, database, application and service.
- Fig. 1.3.2 shows the grid protocol architecture
- **Application layer :** This layer consists of the applications that operate

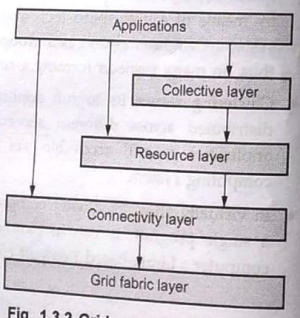

Fig. 1.3.2 Grid protocol architecture

within a virtual organization and which make use of the grid computing environment.

- **Connectivity :** Core communication and authentication protocols required for grid-specific network transactions.
- **Resource :** Secure negotiation, initiation, monitoring, control, accounting and payment of sharing operations on individual resources.
- **Collective :** Protocols and services of global nature to capture interactions across collections of resources.
- **Fabric layer :** Fabric layer provides interfaces to local resources at a specific site. Interfaces are tailored to allow sharing of resources within a virtual organization. It also provide functions for querying the state and capabilities of a resource, along with functions for actual resource management (e.g., locking resources).
- The grid computing middleware software will manage and execute all the activities related to identification, allocation, de-allocation and consolidation of all the computing resources to the end-users transparently, as in the case of a geographical distributed resources system.

1.3.2 Distributed Information Systems

- Web services are a form of distributed information systems.
- Typical examples of distributed computing and information systems are systems that automate the operations of commercial enterprises such as banking and financial transaction processing systems, warehousing systems and automated factories.
- The basic components of a transaction processing system can be found in single user systems. The evolution of these systems provides a convenient framework for introducing their various features. Decreased cost of hardware and communication make it possible to distribute components of transaction processing system. Client-server organization generally used.
- A transaction manager allows the application programmer to group the set of actions, requests, messages and computations into a single operation that is "all or nothing" it either happens or is automatically aborted by the system. The programmer is provided with COMMIT and ABORT verbs that declare the outcome of the transaction.
- Transactions provide the ACID property. ACID characteristic properties of transactions :
  1. *Atomic :* To the outside world, the transaction happens indivisibly.
  2. *Consistent :* The transaction does not violate system invariants.

3. *Isolated* : Concurrent transactions do not interfere with each other.
4. *Durable* : Once a transaction commits, the changes are permanent.

- Nested transaction extends the transaction model by allowing transactions to be composed of other transactions. The outermost transaction in a set of nested transactions is called the top-level transaction. Transactions other than the top-level transaction are called sub-transactions.
- If the top-level transaction commits, then all of the sub-transactions that have provisionally committed can commit too, provided that none of their ancestors has aborted.
- Early enterprise middleware systems handled distributed (or nested) transactions using a transaction processing monitor or TP monitor for integrating applications at the server or database level. Its main task was to allow an application to access multiple server/databases by offering it a transactional programming model.
- A coordinator need simply ensure that if one of the nested transactions aborts, that all other sub-transactions abort as well. Likewise, it should coordinate that all of them commit when each of them can. To this end, a nested transaction should wait to commit until it is told to do so by the coordinator .

TP monitor in distributed systems

- A TP Monitor is a subsystem that groups together sets of related database updates and submits them together to a relational database. The result is that the database server does not need to do all of the work of managing the consistency/correctness of the database; the TP Monitor makes sure that groups of updates take place together or not at all.
- Fig. 1.3.3 show TP monitor system.

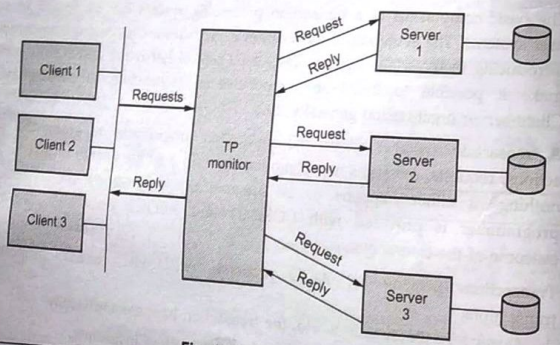

Fig. 1.3.3 TP monitor

Enterprise Application Integration

- The more applications became decoupled from the databases they were built upon, the more evident it became that facilities were needed to integrate applications independent from their databases.
- Application components should be able to communicate directly with each other and not merely by means of the request/reply behavior that was supported by transaction processing systems.
- Result : Middleware as a communication facilitator in enterprise application integration.
- Several types of communication middleware :
  1. **Remote Procedure Calls (RPC) :** An application component can send a request to another application component by doing a local procedure call, which results in the request being packaged as a message and sent to the callee. The result will be sent back and returned to the application as the result of the procedure call.
  2. **Remote Method Invocations (RMI) :** An RMI is the same as an RPC, except that it operates on objects instead of applications.

1.3.3 Distributed Pervasive Systems

- Networking has become a pervasive resource and devices can be connected at any time and any place. The modern Internet is collection of various computer networks.
- Computer network are of different types. Example of network includes a wide range of wireless communication technologies such as WiFi, WiMAX, Bluetooth and third-generation mobile phone networks.
- Programs running on the computers connected to it interact by passing messages, employing a common means of communication.
- The Internet is a collection of large number of computer networks of many different types. Internet communication mechanism is big technical achievement and it is possible by using passing of messages.
- The Internet is a very large distributed system. The web is not equal to the Internet. The implementation of the Internet and services that it supports has entailed the development of practical solutions to many distributed system issues.
- Internet service providers are companies that provide modem links and other types of connection to individual users and small organizations, enabling them to access services anywhere. It also provides local services such as email and web hosting.

- A device must be continually aware of the fact that its environment may change at any time. Many devices in pervasive system will be used in different ways by different users. Devices generally join the system in order to access information and information should then be easy to read, store, manage and share.
- Pervasive Systems are all around us and ideally should be able to adapt to the lack of human administrative control :
  1. Automatically connect to a different network ;
  2. Discover services and react accordingly ;
  3. Automatic self configuration

Electronic Health Care Systems

- New devices are being developed to monitor the well-being of individuals and to automatically contact physicians when needed. Major goal is to prevent people from being hospitalized.
- Personal health care systems equipped with various sensors organized in a Body-Area Network (BAN). Such a network should at worst only minimally hinder a person.
- A central hub is part of the BAN and collects data as needed. Data is then offloaded to a larger storage device. The BAN is continuously hooked up to an external network through a wireless connection, to which it sends monitored data.

Sensor Network

- A sensor network consists of tens to hundreds or thousands of relatively small nodes, each equipped with a sensing device. Most sensor networks use wireless communication and the nodes are often battery powered.
- Their limited resources, restricted communication capabilities, and constrained power consumption demand that efficiency be high on the list of design criteria.
- The relation with distributed systems can be made clear by considering sensor networks as distributed databases. To organize a sensor network as a distributed database, there are essentially two extremes :
  1. Sensors do not cooperate but simply send their data to a centralized database located at the operator's site.
  2. Forward queries to relevant sensors and to let each compute an answer, requiring the operator to sensibly aggregate the returned answers.

Disadvantages : Limited resources including power, restricted communication capabilities.

1.4 Basics of Operating System

- **OS definition** : *Operating system is a program that controls the execution of application programs. It is an interface between applications and hardware.*
- An operating system is software that manages the computer hardware. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.
- A computer is a set of resources. This resource provides various functions to the user. Functions like data movement, storing of data and program, operation on data are control by an operating system.
- Program is a passive entity. Program becomes process when executable file loaded into memory. A process may be independent of other processes in the system.
- Process is an active entity that requires a set of resources, including a processor, program counter, registers to perform its function. Multiple processes may be associated with one program.
- Process means a program in execution. Process execution must progress in sequential order. It also called task. Task is a single instance of an executabl program.
- Each process has its own address space. Address space is divided into regions Text region, Data region and Stack region.
- A process is used as a fundamental unit for resource allocation in operati system. A process normally has its own private memory area in which it runs.
- A process consists of an executing program, its current values, state informati and the resources used by the operating system to manage its execution.

Java processes :

- There are three types of Java program : Applications, applets and servlets, all written as a class.
- A Java application program has a main method and is run as independent(standalone) process. An applet does not have a main method a run using a browser or the applet viewer.
- A servlet does not have a main method and is run in the context of a web se
- A Java program is compiled into bytecode, a universal object code. When ru bytecode is interpreted by the Java Virtual Machine (JVM).

- **Java programs are of three types :**
  - **a) Applications :** Program whose byte code can be run on any system which has a Java virtual machine. An application may be standalone (monolithic) or distributed.
  - **b) Applets :** A program whose byte code is downloaded from a remote machine and is run in the browser's Java virtual machine.
  - **c) Servlets :** A program whose byte code resides on a remote machine and is run at the request of an HTTP client (a browser).

1.4.1 Difference between Process and Program

Sr. No.	Process	Program
1.	Process is active entity.	Program is passive entity.
2.	Process is a sequence of instruction executions.	Program contains the instructions.
3.	Process exists in a limited span of time.	A program exists at single place and continues to exist.
4.	Process is a dynamic entity.	Program is a static entity.

1.4.2 Process State Diagram

- Each process has an execution state which indicates what process is currently doing. The process descriptor is the basic data structure used to represent the specific state for each process.
- Fig. 1.4.1 shows a process state diagram. A state diagram is composed of a set of states and transitions between states.

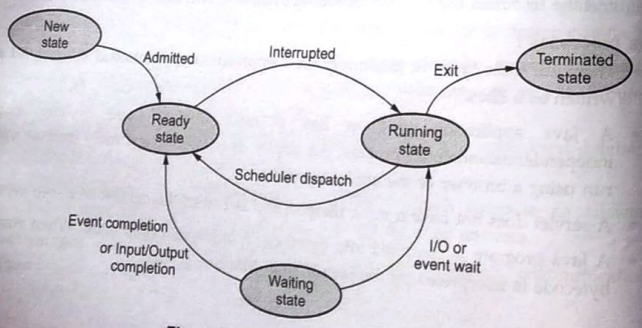

Fig. 1.4.1 Process state diagram

- State diagram is used by process manager to determine the type of service to provide to the process.
- The process states are as follows :
  1. New 2. Ready 3. Running 4. Waiting 5. Terminated.
- **New :** Operating system creates new process by using fork( ) system call. These process are newly created process and resources are not allocated.
- **Ready :** The process is competing for the CPU. Process reaches to the head of the list (queue).
- **Running :** The process that is currently being executed. Operating system allocates all the hardware and software resources to the process for execution.
- **Waiting :** A process is waiting until some event occurs such as the completion of an input-output operation.
- **Terminated :** A process completes its operations and releases all resources.
- Operating system maintains a ready list of ready process and a blocked list of blocked processes. The ready list is maintained in priority order and blocked list is typically unordered.
- The act of assigning a processor to the first process on the ready list is called **dispatching** and is performed by a system entity called the **dispatcher.**
- When process is in new state, the program remains in the secondary storage. Here process itself is not in the main memory and space is not allocated.
- The process exists a system because of two reasons :
  1. Process is terminated when it completes its operation.
  2. Process aborts due to an unrecoverable error.
- To prevent any one process from continuously using the system, the operating system sets a hardware interrupting clock to allow a process to run for a specific time interval or time quantum.
- The process may request a resource when it is in the running state. In most of the operating system, if a running process requests an immediately available resources the process is allowed to continue in the running state.
- From the blocked state, the process can move to the ready state only by being allocated the requested resource.
- User only initiate process state transition is **blocked** and remaining all other state transitions is initiated by the operating system.

Sr. No.	State transition	Remarks
1.	Ready to running	Process is dispatched.
2.	Running to ready	Process time slice expires.
3.	Running to blocked	When a process blocks.
4.	Blocked to ready	When the event for which it has been waiting occurs.
5.	Ready to exit	This is possible when parent process may terminate child process at any time.
6.	Running to exit	When currently running process completes its operation then OS terminates the process.

1.4.3 Concurrent Processing

- Concurrent processing can create the same effect with one processor by switching between threads of processes at different times to allow all of the processes to execute seemingly simultaneously.
- In concurrent processing, the processor executes each thread for a specific time frame. If the execution of a thread is not completed during the allocated time slot, the thread is paused and the processor moves to the next thread. A thread that has been paused will continue to be executed the next time it is assigned processor time.
- Contemporary OS : Multiple processes appear to execute concurrently on a machine via timesharing resources.
- Fig. 1.4.2 shows timesharing of a resource.

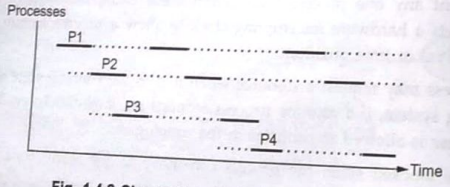

Fig. 1.4.2 Shows timesharing of a resource

1.4.4 Concurrent Processing within a Process

- It is often useful for a process to have parallel threads of execution, each of which timeshare the system resources in much the same way as concurrent processes.
- Fig. 1.4.3 shows parent process may spawn child processes.

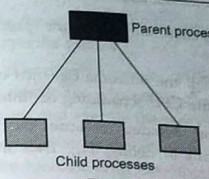

Fig. 1.4.3

- Fig. 1.4.3 shows a process may spawn child threads.
- The Java virtual machine allows an application to have multiple threads of execution running concurrently. When a Java virtual machine starts up, there is usually a single thread. The Java virtual machine continues to execute threads until either of the following occurs :
  a) The exit method of class runtime has been called and the security manager has permitted the exit operation to take place.
  b) All threads have terminated either by returning from the call to the run method or by throwing an exception that propagates beyond the run method.

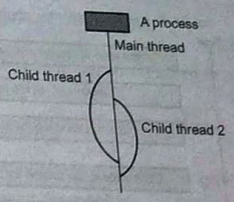

Fig. 1.4.4

- Thread is a dispatchable unit of work. It consists of thread ID, program counter, stack and register set. Thread is also called a Light Weight Process (LWP). Because they take fewer resources then a process. A thread is easy to create and destroy.

Thread - safe programming :

- When two threads independently access and update the same data object, such as a counter, as part of their code, the updating needs to be synchronized.
- Because the threads are executed concurrently, it is possible for one of the updates to be overwritten by the other due to the sequencing of the two sets of machine instructions executed in behalf of the two threads.
- To protect against the possibility, a synchronized method can be used to provide mutual exclusion.

1.4.5 Race Condition

- Race condition occurs when two or more operations occur in an undefine manner. When two or more processes are reading or writing some shared da

and the final result depends on who runs precisely when, are called race condition.

- There is a "race condition" if the outcome depends on the order of the execution. The outcome depends on the CPU scheduling or "interleaving" of the threads.
- Race condition should be avoided because they can cause fine errors in applications and are difficult to debug. Fig. 1.4.5 shows race condition between threads.

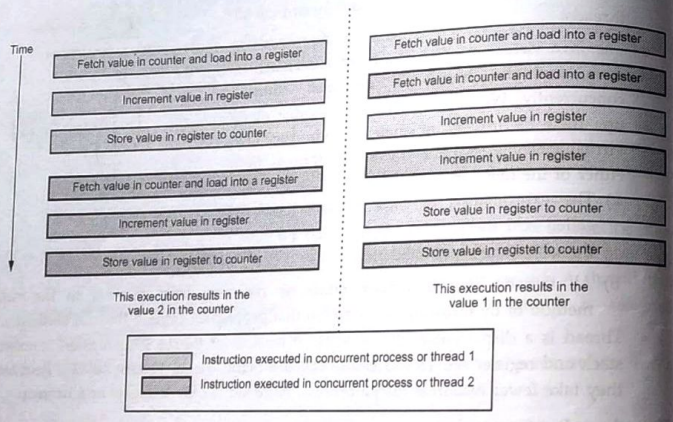

Fig. 1.4.5 Race condition between threads

Monitor :

- A monitor is a concurrency construct that encapsulates data and functionality for allocating and releasing shared resources (such as network connections, memory buffers, printers and so on).
- Monitor is an object that contains both data and procedures needed to perform allocation of a shared resource. It is a programming language construct that support both data access synchronization and control synchronization.
- Monitor is implemented in programming languages like Pascal, Java and C++. Java makes extensive use of monitors to implement mutual exclusion.
- Monitor is an abstract data type for which only one process may be executing a procedure at any given time. Monitor is a collection of procedure, variables and

data structure. Data inside the monitor may be either global to all routines within the monitor or local to a specific routine.

- To accomplish resource allocation or release, a thread calls a monitor entry. If there is no other thread executing code within the monitor, the calling thread is allowed to enter the monitor and execute the monitor entry's code.
- But if a thread is already inside of the monitor, the monitor makes the calling thread wait outside of the monitor until the other thread leaves the monitor. The monitor then allows the waiting thread to enter.
- Because synchronization is guaranteed, problems such as data being lost or scrambled are avoided.
- The JVM specification goes on to state that monitor behaviour can be explained in terms of locks.
- A lock is a token that a thread must acquire before a monitor allows that thread to execute inside of a monitor entry.
- That token is automatically released when the thread exits the monitor, to give another thread an opportunity to get the token and enter the monitor.
- Java associates locks with objects : Each object is assigned its own lock and each lock is assigned to one object.
- A thread acquires an object's lock prior to entering the lock-controlled monitor entry, which Java represents at the source code level as either a synchronized method or a synchronized statement.

1.5 Networking

- Computer network is designed around the concept of layered protocols or functions. For exchange of data between computers, terminals or other data processing devices, there is data path between two computers, either directly or via a communication network.
- Distributed system is a system whose components are located on different networked computers, which then communicate and coordinate their actions by passing messages to one another.

Network standards and protocols :

- On public networks such as the Internet, it is necessary for a common set of rules to be specified for the exchange of data. Such rules, called protocols, specify such matters as the formatting and semantics of data, flow control, error correction.

- Protocol is a well-known set of rules and formats used for communication between processes to perform a given task. It is implemented by a pair of software modules located in the sending and receiving computers.
- Protocol software modules are arranged in a hierarchy of layers. A complete set of protocol layers is referred to as a protocol suite or protocol stack.
- Software can share data over the network using network software which supports a common set of protocols.
- In the context of communications, a protocol is a set of rules that must be observed by the participants. In communications involving computers, protocols must be formally defined and precisely implemented.
- For each protocol, there must be rules that specify the followings :
  a) How is the data exchanged encoded ?
  b) How are events (sending, receiving) synchronized so that the participants can send and receive in a coordinated order ?
- The specification of a protocol does not dictate how the rules are to be implemented.

The network architecture

- Network hardware transfers electronic signals, which represent a bit stream, between two devices.
- Modern day network applications require an Application Programming Interface (API) which masks the underlying complexities of data transmission.
- A layered network architecture allows the functionalities needed to mask the complexities to be provided incrementally, layer by layer. Actual implementation of the functionalities may not be clearly divided by layer.
- Most of all networks are organized as a series of layers, each one built upon the one below it. Because of layer, it reduces the design complexity.
- In layer protocols, a layer is a service provider and may consists of several service functions. Function is a sub system of a layer. Each subsystem may also be made up of entities. An entity is a specialized module of a layer or subsystem.
- Name of the layer, total number of layers, function and content of each layer differ from network to network.

1.5.1 OSI Architecture

- The ISO was one of the first organizations to formally define a common way to connect computers. Their architecture, called the Open System Interconnection (OSI).

- The International organization for standardization developed the Open System Interconnection (OSI) reference model. OSI model is the most widely used model for networking.
- OSI model is a seven-layer standard. The OSI model does not specify the communication standard or protocols to be used to perform networking tasks.
- Fig. 1.5.1 shows OSI seven-layer network architecture.

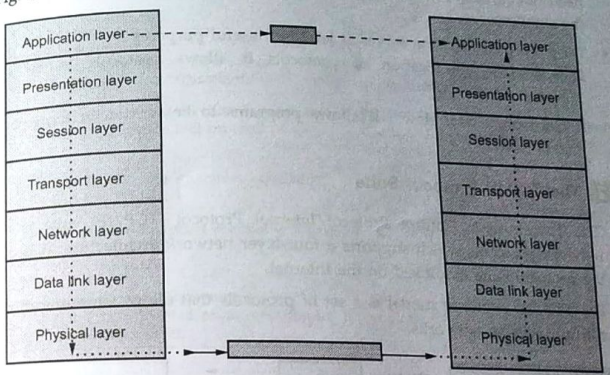

Fig. 1.5.1 OSI seven-layer network architecture

- Physical layer co-ordinates the functions required to transmit a bit stream over a communication channel. It deals with electrical and mechanical specifications of interface and transmission media.
- The data link layer is responsible for transmitting frames from one node to the next. It transforms the physical layer to a reliable link making it an error free link to upper layer.
- The network layer is responsible for the delivery of packets from the source to destination.
- The transport layer is responsible for delivery of message from one process to another.
- The session layer is network dialog controller i.e. it establishes and synchronizes the interaction between communication system.
- The presentation layer deals with syntax and semantics of the information being exchanged.

- Application layer is responsible for accessing the network by user. It provides the interfaces and other supporting services such as e-mail, remote file access, file transfer, sharing database, message handling (X.400), directory services (X.500).

Network architecture

- The division of the layers is conceptual : The implementation of the functionalities need not be clearly divided as such in the hardware and software that implements the architecture.
- The conceptual division serves at least two useful purposes :
  1. Systematic specification of protocols it allows protocols to be specified systematically.
  2. Conceptual data flow : It allows programs to be written in terms of logical data flow.

1.5.2 The TCP/IP Protocol Suite

- The Transmission Control Protocol/Internet Protocol (TCP/IP) suite is a set of network protocols which supports a four-layer network architecture. It is currently the protocol suite employed on the Internet.
- The TCP/IP reference model is a set of protocols that allow communication across multiple diverse networks.

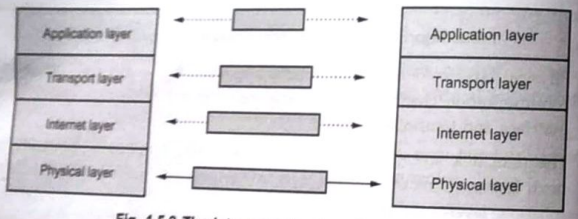

Fig. 1.5.2 The Internet network architecture

- The Internet layer implements the Internet protocol, which provides the functionalities for allowing data to be transmitted between any two hosts on the Internet.
- The transport layer delivers the transmitted data to a specific process running on an Internet host.
- The application layer supports the programming interface used for building a program.

- Physical layer : It is responsible for accepting and transmitting IP datagrams. This layer may consist of a device driver in the operating system and the corresponding network interface card in the machine.

1.5.3 Comparison of the OSI and TCP/IP Protocol Suite

Sr. No.	OSI	TCP/IP
1.	7 layers	4 layers
2.	Model was first defined before implementation takes place.	Model defined after protocol were implemented.
3.	OSI model based on three concept i.e. service, interface and protocol.	TCP/IP model did not originally clearly distinguish between service, interface and protocol.
4.	OSI model gives guarantee of reliable delivery of packet.	Transport layer does not always guarantee the reliable delivery of packet.
5.	OSI does not support Internet working.	TCP/IP support.
6.	Strict layering.	Loosely layered.
7.	Support connectionless and connection-oriented communication in the network layer.	Support only connection-oriented communication in the transport layer.

1.5.4 Network Resources

- Network resources are resources available to the participants of a distributed computing community. It includes hardware such as computers and equipment and software such as processes, email mailboxes, files, web documents.
- An important class of network resources is network services such as the world wide web and file transfer (FTP), which are provided by specific processes running on computers.
- In identification of network resources, one of the key challenges in distributed computing is the unique identification of resources available on the network, such as email mailboxes and web documents.
- Resource sharing is main motivation of the distributed system. The term "resource" is a rather abstract one, but it best characterizes the range of things that can usefully be shared in a networked computer system.
- Resources may be the software resources or hardware resources. Printers, disks, CDROM and data are the example of software and hardware resources. Sharing of

resource extends from hardware components such as disks and printers to software - defined entities such as files, databases and data objects of all kinds.

- It also includes the stream of video frames and audio connection that a mobile phone call represents. A resource manager is a software module that manages a set of resources of a particular type.

1.5.5 Addressing an Internet Host

- Internet topology is the structure by which hosts, routers or Autonomous Systems (AS) are connected to each other. Fig. 1.5.3 shows internet topology.

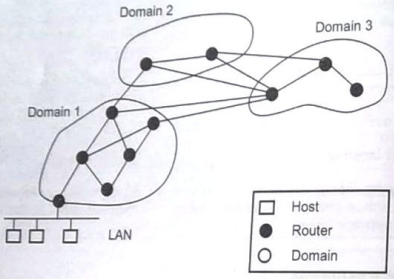

Fig. 1.5.3 Internet topology

- AS-level Internet topology is at the highest granularity of the Internet; other levels of Internet topology partially depend on AS-level topology. Second, the AS-level topology is relatively easy to obtain; other levels of topology are sometimes regarded as private information and are harder to get. Third, AS-level topology is not directly engineered by humans; instead, it is the aggregate result of technological and economical forces and, therefore, its origin and evolution attract considerable interest from investigators.
- The Internet consists of a hierarchy of networks, interconnected via a network backbone. Each network has a unique network address.
- Computers or hosts, are connected to a network. Each host has a unique ID within its network. Each process running on a host is associated with zero or more ports. A port is a logical entity for data transmission.

1.5.6 The Internet Addressing Scheme IPv4

- The IP address size is 32-bit. The 32-bit numeric identifier contains a unique network identifier within the Internet, allocated by the Internet Network Information Center (NIC). A unique host identifier within that network, assigned by its manager.
- The version of IP currently using is IPv4. New version is IPv6 that designed to overcome addressing limitation of IPv4.
- IP address written as a sequence of four decimal numbers separated by dots. It has equivalent symbolic domain name represented in a hierarchy. Fig. 1.5.4 shows IP address.

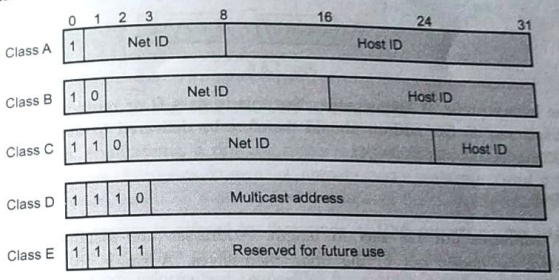

Fig. 1.5.4 IP address

- IP address has five classes :
  a. Class A : Reserved for very large networks (224 hosts on each).
  b. Class B : Allocated for organization networks contain more than 255 hosts.
  c. Class C : Allocated to all other networks (less than 255 hosts on each).
  d. Class D : Reserved for multicasting but this is not supported by all routers.
  e. Class E : Unallocated addresses reserved for future requirements.
- **Example :** Suppose the dotted-decimal notation for a particular Internet address is 129.65.24.50. The 32-bit binary expansion of the notation is as follows :

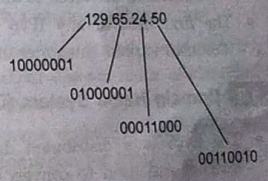

Fig. 1.5.5

- Since the leading bit sequence is 10, the address is a class B address. Within the class, the network portion is identified by

the remaining bits in the first two bytes, that is, 00000101000001 and the host portion is the values in the last two bytes or 0001100000110010.

- For convenience, the binary prefix for class identification is often included as part of the network portion of the address, so that we would say that this particular address is at network 129.65 and then at host address 24.50 on that network.
- Example : Given the address 224.0.0.1, one can expand it as follows :

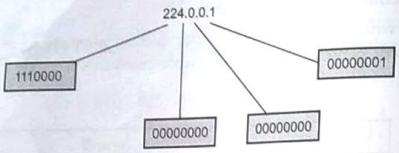

Fig. 1.5.6

- The binary prefix of 1110 signifies that this is class D or multicast, address. Data packets sent to this address should therefore be delivered to the multicast group 00000000000000000000000000000001.

1.5.7 IPv6

- IPv6 addresses are 128 bits in length. Addresses are assigned to individual interface on nodes, not to the node themselves. A single interface may have multiple unique unicast addresses. The first field of any IPv6 address is the variable length format prefix, which identifies various categories of addresses.
- There are three types of addresses :
  1. Unicast : An identifier for a single interface. A packet sent to a unicast address is delivered to the interface identified by that address.
  2. Anycast : An identifier for a set of interfaces. A packet sent to an anycast address is delivered to one of the interfaces identified by the address.
  3. Multicast : An identifier for a set of interfaces. A packet sent to a multicast address is delivered to all interfaces identified by that address.
- The first field of any IPv6 address is the variable-length format prefix, which identifies various categories of address.

1.5.8 Domain Name System (DNS)

- The DNS is a distributed database that resides on multiple machines on the Internet and used to convert between names and address and to provide e-mail routing information.

- For user friendliness, each Internet address is mapped to a symbolic name, using the DNS, in the format of :
  <computer-name>.<subdomainhierarchy>.<organization>.<sectorname>
  {.<country code>}
  e.g., www.technicalpublications.org
- Fig. 1.5.7 shows the DNS in the internet

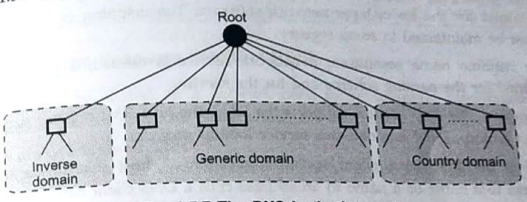

Fig. 1.5.7 The DNS in the Internet

- For network applications, a domain name must be mapped to its corresponding Internet address. Processes known as domain name system servers provide the mapping service, based on a distributed database of the mapping scheme.
- The mapping service is offered by thousands of DNS servers on the Internet, each responsible for a portion of the name space, called a zone. The servers that have access to the DNS information (zone file) for a zone is said to have authority for that zone.

The Domain Name System (DNS) is a hierarchical, distributed naming system designed to cope with the problem of explosive growth :

1. It is *hierarchical* because the name space is partitioned into *subdomains*.
2. It is distributed because management of the name space is delegated to local sites. Local sites have complete control (and responsibility) for their part of the name space. DNS queries are handled by servers called *name servers*.
3. It does more than just map machine names to Internet addresses. For example, it allows a site to associate multiple machines with a single, mailbox name.

In the DNS, the name space is structured as a tree, with *domain names* referring to nodes in the tree. The tree has a *root* and a *fully-qualified* domain name is identified by the *components* of the path from the domain name to the root.

- In zone, a server is responsible and have some authority. The server makes database called zone file and keeps all the information for every node under that domain.

- Domain and zone are same if server accepts responsibility for a domain and does not divide the domain into subdomain.
- Domain and zone are different, if a server divides its domain into subdomains and delegates part of its authority to other server.

Name lookup and resolution

- If a domain name is used to address a host, its corresponding IP address must be obtained for the lower-layer network software. The mapping or name resolution, must be maintained in some registry.
- For runtime name resolution, a network service is needed; a protocol must be defined for the naming scheme and for the service.
- Example : The DNS service supports the DNS; the Java RMI registry supports RMI object lookup; JNDI is a network service lookup protocol.
- DNS is designed as a client server application. A host that needs to map an address to a name or a name to an address calls a DNS client named a resolver.
- Name resolving must also include the type of answer desired. The DNS partitions the entire set of names by class (for mapping to multiple protocol suites).
- Naming items is required since one cannot distinguish the names of subdomains from the names of individual objects or their types.

Well known ports :

- Each Internet host has 2^{16} (65,535) logical ports. Each port is identified by a number between 1 and 65535 and can be allocated to a particular process.
- Port numbers between 1 and 1023 are reserved for processes which provide well-known services such as finger, FTP, HTTP and email.
- Port numbers from 0 to 65535 is used in Internet. It is 16 bits integer so the range is 0 to 65535. The client program defines itself with a port number, chosen randomly by the transport player software running on the client host. This is the *ephemeral port number.*
- Server also define a port number but not randomly. Internet has decided to use universal port numbers for servers, these are called *well known port numbers.* The port number ranging from 0 to 1023 are called well known port numbers and are restricted, which means that they are reserved for use by well known application protocols such as HTTP.
- The Internet Assigned Number Authority (IANA) has divided the port number into three ranges. They are

a) Well known ports b) Registered ports c) Dynamic ports.

Sr. No.	Port type	Range	Remark
1.	Well known port	0 to 1023	Assigned and controlled by IANA.
2.	Registered port	1024 to 49151	Not assigned and controlled by IANA. Only registered to prevent duplication.
3.	Dynamic	49152 to 65535	Neither controlled nor registered. Used by any process. These are ephemeral ports.

1.5.9 The Uniform Resource Identifier (URI)

- Resources to be shared on a network need to be uniquely identifiable. On the Internet, a URI is a character string which allows a resource to be located.
- There are two types of URIs :
  a) URL (Uniform Resource Locator) points to a specific resource at a specific location.
  b) URN (Uniform Resource Name) points to a specific resource at a nonspecific location.

Uniform Resource Locator

- The Uniform Resource Locator (URL) is a standard for specifying any kind of information on the Internet.
- URL has three parts
  1. The protocol.
  2. DNS name of the machine where the page is located.
  3. File name containing the page.
- URL is represented as Fig. 1.5.8.
- The protocol is the client-server program used to retrieve the document.
- Host is the computer on which the information is located.

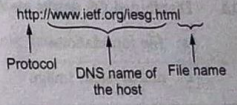

Fig. 1.5.8

- The URL can optionally contain the port number of the server.
- File name gives where the information is located.

1.6 Fill in the Blanks

Q.1 A _____ is a network link with a high transmission capacity, employing satellite connections, fibre optic cables and other high-bandwidth circuits.

2.1 Architectural Styles

- Architectural model is an abstract view of a distributed system. Models are constructed to simplify reasoning about the system.
- A model of a distributed system is expressed in terms of **components, placement of components and interactions among components.**
- An Architectural models describe a system in terms of the computational and communication tasks performed by its computational elements; the computational elements being individual computers or aggregates of them supported by appropriate network interconnections.
- An Architectural model defines the way in which the components of systems interact with one another and the way in which they are mapped onto an underlying network of computers.
- An architectural model of a distributed system first simplifies and abstracts the functions of the individual components of a DS and then it considers :
  1. The placement of the components across a network of computers
  2. The interrelationships between the components.
- Software architecture is a logical organization of distributed systems into software components. **Software component** is a modular unit with well defined, required and provided interfaces that is replaceable within its environment.
- Important styles of architecture for distributed systems
  a) Layered architectures
  b) Object - based architectures
  c) Data - centered architectures
  d) Event - based architectures

2.1.1 Layered Architectures

- Layered architecture style is simple.
- In this method, any complex system is divided into number of layers. Each layer performed it's given task and it also provides services to below and above layers. A given layer therefore offers a software abstraction, with higher layers being unaware of implementation details.
- Components are organized in a layered fashion where a component at layer L_i is allowed to call components at the underlying layer L_{i-1}, but not the other way around.
- Fig. 2.1.1 shows layered architecture.

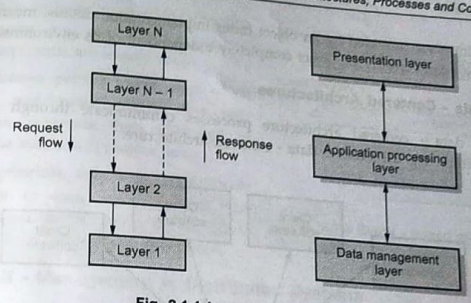

Fig. 2.1.1 Layered architecture

- Layered architecture is widely adopted by the networking community.
- Layer : A group of related functional components.
- Service : Functionality provided to the next layer.
- The lowest - level hardware and software layers are often referred to as a platform for distributed systems and applications. These low - level layers provide services to the layers above them, which are implemented independently in each computer.
- These low - level layers bring the system's programming interface up to a level that facilitates communication and coordination between processes.

2.1.2 Object-Based Architectures

- This architectural style is based on an arrangement of loosely coupled objects, it is less structured. Each object corresponds a component. Components are connected through a (remote) procedure call mechanism.
- Fig 2.1.2 shows Object based architectural style.
- Object - based architectures are attractive because they provide a natural way to encapsulate data and the operations that can be performed on that data in a single entity.

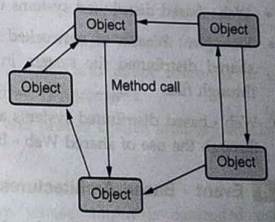

Fig. 2.1.2 Object based architectural style

- The interface provided by an object hides implementation details, meaning that first we can consider an object completely independent of its environment.

2.1.3 Data - Centered Architectures

- In a data - centered architecture processes communicate through a commo repository. Fig. 2.1.3 shows data - centered architecture.

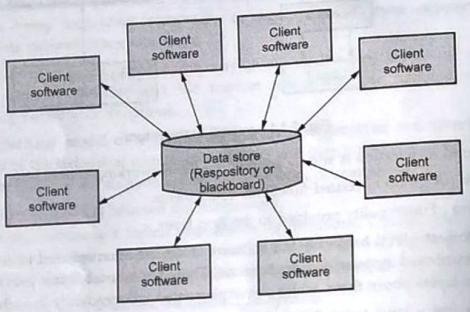

Fig. 2.1.3 Data-centered architecture

- For instance many networked applications use a shared distributed file system i which communication takes place through files.
- Web - based distributed systems use shared web - based data services.
- Example : Wealth of networked applications has been developed that rely on shared distributed file system in which virtually all communication takes pla through files.
- Web - based distributed systems are largely data - centric : processes communica through the use of shared Web - based data services.

2.1.4 Event - Based Architectures

- Fig. 2.1.4 shows event - based architecture.
- Components communicate by using events that can carry data. Process communicate through the propagation of events.

- For instance, publish/subscribe systems are event-based systems. Components are loosely coupled.
- Processes publish events after which the middleware ensures that only those processes that subscribed to those events will receive them.
- In principle, they need not explicitly refer to each other. This is also referred to as being decoupled in space, or **referentially decoupled**.

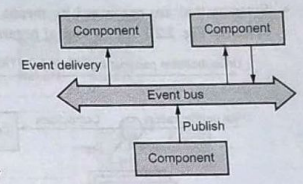

Fig. 2.1.4 Event - based architecture

2.2 Self - Management in Distributed Systems

- Self - management exists on all levels of the system. At the lowest level, self - management means that the system should be able to automatically handle frequent addition or removal of nodes, frequent failure of nodes, load balancing between nodes, and threats from adversaries.
- Autonomic computing, proposed by Paul Horn of IBM in 2001, shared the vision of making all computing systems manage themselves automatically.
- It refers to self - managing characteristics of distributed computing resources, which recognize and understand changes in the system, take appropriate corrective actions completely automatically, with close to zero human intervention.
- The Autonomic Computing Initiative divides self - management into four functional areas :
  a) Self - configuration : Automatically configure components to adapt them to different environments.
  b) Self - healing : Automatically discover, diagnose, and correct faults.
  c) Self - optimization : Automatically monitor and adapt resources to ensure optimal functioning regarding the defined requirements.
  d) Self - protection : Anticipate, identify and protect against arbitrary attacks.

2.2.1 Feedback Control Model

- A feedback control system is a system whose output is controlled using its measurement as a feedback signal. This feedback signal is compared with a reference signal to generate an error signal which is filtered by a controller to produce the system's control input.

- Systems that are organized by means of loops are referred to as feedback Contro systems. Fig. 2.2.1 shows logical organization of a feedback control system.

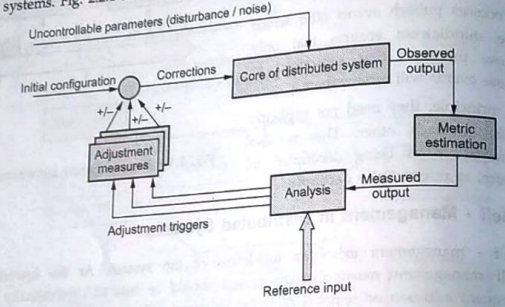

Fig. 2.2.1 Feedback control system

- The main objectives of feedback control is to ensure that variables of interest in a process or a system, thought of as the output signals, either
  a) track reference trajectories (called tracking or servo), or
  b) are maintained close to their setpoints (called regulation).
- Feedback Systems are very useful and widely used in amplifier circuits, oscillators process control systems as well as other types of electronic systems.
- Elements of feedback control system are as follows :
  1) System itself needs to be monitored. Feedback control loop generally contains a logical metric estimation component.
  2) Analyses the measurement and compares these values with references value.
  3) Various mechanisms to directly influence the behaviour of the system.

2.2.2 Systems Monitoring with Astrolabe

- Astrolabe gathers, disseminates and aggregates information about zones. A zone is recursively defined to be either a host or a set of non-overlapping zones.
- Zones are said to be non-overlapping if they do not have any hosts in common. Thus, the structure of Astrolabe's zones can be viewed as a tree. The leaves of this tree represent the host.
- Fig. 2.2.2 shows zone hierarchy of Astrolabe.

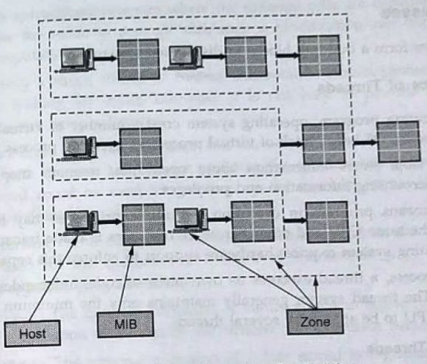

Fig. 2.2.2 Zone hierarchy of astrolabe

- Each zone has a local zone identifier, a string name unique within the parent zone. A zone is globally identified by its zone name, which is a string consisting of its path of zone identifiers from the root, separated by slashes.
- Each host runs an Astrolabe agent. The zone hierarchy is implicitly specified when the system administrator initializes these agents with their names.
- Each host maintains a set of attributes for collecting local information. Each zone has a set of aggregation functions that calculate the attributes for the zone's MIB. An aggregation function for a zone is an SQL program, which takes a list of the MIBs of the zone's child zones and produces a summary of their attributes.
- Leaf zones form an exception. Each leaf zone has a set of virtual child zones. The virtual child zones are local to the corresponding agent. The attributes of these virtual zones are writable, rather than being generated by aggregation functions. Each leaf zone has at least one virtual child zone called "system", but the agent allows new virtual child zones to be created.
- Aggregation functions are programmable. The code of these functions is embedded in so-called Aggregation Function Certificates (AFCs), which are signed and timestamped certificates that are installed as attributes inside MIBs.

2.3 Processes

- Processes form a building blocks in distributed systems.

2.3.1 Basics of Threads

- For executing program, operating system creates number of virtual processors for each process. OS keeps track of virtual processors by using process table.
- Process table stores information about open files, memory map, CPU register values, accounting information and privileges.
- Process means program in execution. Multiple processes may be concurrently sharing the same CPU and other hardware resources is made transparent. Usually, the operating system requires hardware support to enforce this separation.
- Like a process, a thread executes its own piece of code, independently from other threads. The thread system generally maintains only the minimum information to allow a CPU to be shared by several threads.

Advantages of Threads

a) Allow for parallel computation.

b) Switching between threads is faster than switching between process.

c) Creating and destroying threads is cheaper than creating and destroying a process.

d) Easier to structure many applications as a collection of cooperating threads.

e) Higher performance compared to multiple processes since switching between threads takes less time.

Thread Usage in Non-Distributed Systems

- The major drawback of all IPC mechanisms is that communication often requires extensive context switching, shown at three different points.
- Fig. 2.3.1 shows Context switching as the result of IPC.

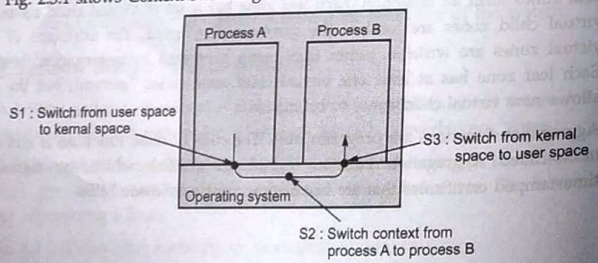

Fig. 2.3.1

- Example : A spreadsheet program where the different cells are dependent, when a user changes the value in a single cell, such a modification can trigger a large series of computations :
- If there is only a single thread of control, computation cannot proceed while the program is waiting for input. Likewise, it is not easy to provide input while dependencies are being calculated.
- The easy solution is to have at least two threads of control : one for handling interaction with the user and one for updating the spreadsheet. In the mean time, a third thread could be used for backing up the spreadsheet to disk while the other two are doing their work.
- Processor context : The minimal collection of values stored in the registers of a processor used for the execution of a series of instructions (e.g., stack pointer, addressing registers, program counter).
- Thread context : The minimal collection of values stored in registers and memory, used for the execution of a series of instructions (i.e., processor context, state).
- Process context : The minimal collection of values stored in registers and memory, used for the execution of a thread (i.e., thread context, but now also at least MMU register values).
- Threads use the same address space. No support from OS/HW to protect threads using each other's memory. Thread context switching may be faster than process context switching.

Advantages of User - level thread library

1. It is cheap to create and destroy threads.
2. Switching thread context can often be done in just a few instructions.

- Major drawback of user - level threads is that invocation of a blocking system call will immediately block the entire process to which the thread belongs.

2.3.2 Threads in Distributed Systems

- Allowing multiple threads of control in a process introduces concurrency. Each thread shares and operates within the common process address space but each has its own local processor state maintained in a Thread Control Block (TCB) associated with the process.
- When using a multithreaded client, connections may be set up to different replicas, allowing data to be transferred in parallel. It is used to express communication in the form of multiple logical connections at the same time.

- An important property of threads is that they can provide a convenient means allowing blocking system calls without blocking the entire process in which thread is running.
- This property makes threads particularly attractive to use in distributed systems it makes it much easier to express communication in the form of maintaining multiple logical connections at the same time.
- A main contribution of threads in distributed systems is that they allow clien and servers to be constructed such that communication and local processing o overlap, resulting in a high level of performance.
- Attractive to use in distributed systems are Multithreaded client a Multithreaded server.

Multithreaded Clients

- Multithreaded clients can be used to hide delays/latencies in netwo communications, by initiating communication and immediately proceeding wi something else.
- Example : web browsers such as IE are multi-threaded.
- A web browser can start up several threads : Once the main HTML file has be fetched, separate threads can be activated to take care of fetching the other par Each thread sets up a separate connection to the server and pulls in the data. O for downloading the HTML source of the page, one each for images on the pag one each for animations/applets etc.
- Replicated web servers along with multi - threaded clients can result in shor download times.

Multithreaded Servers

- There are several ways to organize servers. A multithreaded server is an examp of a concurrent server.
- In the case of an iterative server, the server itself handles the request and, necessary, returns a response to the requesting client.
- A concurrent server does not handle the request itself, but passes it to a separa thread or another process, after which it immediately waits for the next incomin request.
- One Thread (dispatcher) reads incoming requests for a file operation. The reques are sent by clients to a well - known end point for this server. After examining th request, the server chooses an idle worker thread and hands it the request.
- Fig. 2.3.2 shows multithreaded server organized in a dispatcher/worker model.
- The worker proceeds by performing a blocking read on the local file syste which may cause the thread to be suspended until the data are fetched from disk
- If the thread is suspended, another thread is selected to be executed. For exampl the dispatcher may be selected to acquire more work. Alternatively, anothe worker thread can be selected that is now ready to run.

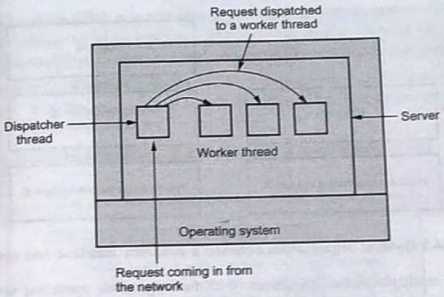

Fig. 2.3.2 Multithreaded server organized in a dispatcher/worker model

- Three ways to construct server are as follows :
  a) Thread : Blocking system calls make programming easier and parallelism improves performance.
  b) Single threaded process : The single - threaded server retains the ease and simplicity of blocking system calls, but gives up performance.
  c) Finite state machine : The finite - state machine approach achieves high performance through parallelism, and uses nonblocking calls, thus is hard to program.

2.4 Virtualization

- Virtualization is the creation of a virtual version of something, such as a hardware platform, resources, operating system, a storage device or network resources.
- Virtualization is becoming increasingly important :
  a) Hardware changes faster than software
  b) Ease of portability and code migration
  c) Isolation of failing or attacked components
- Fig. 2.4.1 shows general organization between a program, interface and system.
- Virtualization is a frame work or methodology of dividing the resources of computer into multiple execution environments. Virtualization is an abstraction layer that decouples the physical hardware from the operating system to deliver greater IT resource utilization and flexibility.

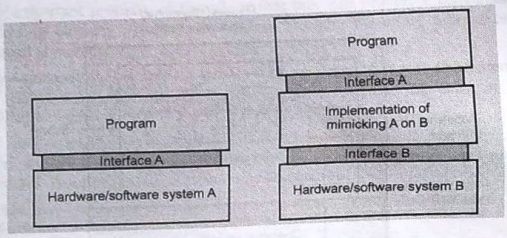

Fig. 2.4.1 General organization between a program, interface and system

- It allows multiple virtual machines, with heterogeneous operating systems to run in isolation, side-by-side on the same physical machine.
- Virtualization helps with scalability and better utilization of hardware resources. It allows legacy software to run on expensive mainframe hardware.
- Virtualization runs multiple different operating systems at the same time and provides a high degree of portability and flexibility.

2.4.1 Architectures of Virtual Machines

- Computer systems generally offer four different types of interfaces, at four different levels :
  1. An interface between the hardware and software, consisting of machine instructions that can be invoked by any program.
  2. An interface between the hardware and software, consisting of machine instructions that can be invoked only by privileged programs, such as an operating system.
  3. An interface consisting of system calls as offered by an operating system.
  4. An interface consisting of library calls, generally forming what is known as an application programming interface (API).
- A virtual machine can support individual processes or a complete system depending on the abstraction level where virtualization occurs. Some VMs support flexible hardware usage and software isolation, while others translate from one instruction set to another.
- A virtual machine is a software construct that mimics the characteristics of a physical server.

- A Virtual Machine (VM) is a software program or operating system that not only exhibits the behavior of a separate computer, but is also capable of performing tasks such as running applications and programs like a separate computer.
- In a pure virtual machine architecture the operating system gives each process the illusion that it is the only process on the machine. The user writes an application as if only its code were running on the system.
- Fig. 2.4.2 shows various interfaces offered by computer system.

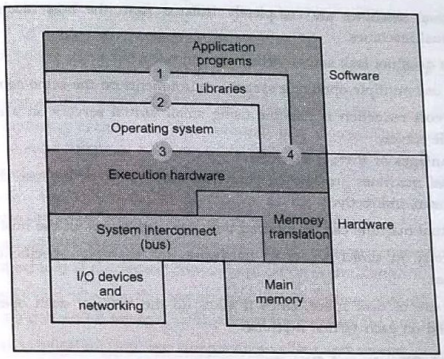

Fig. 2.4.2 Various interfaces offered by computer system

- Virtualization can be implemented at two levels.

1. Process Virtual Machine :
   - Virtualization is done essentially only for executing a single process (program).
   - An abstract instruction set that is to be used for executing applications.
   - For example: Java runtime, Windows emulation (Wine) on Unix/Linux/MacOS.
2. Virtual Machine Monitor :
   - Composed of the host OS and the virtualization software.
   - A layer completely shielding the original hardware but offering the complete instruction set of that same (or other hardware) as an interface.

- Provides a further decoupling between hardware and software allowing movi complete environment from one machine to another.
- Makes it possible to have multiple instances of different operating systems r simultaneously and concurrently on the same platform.
- Examples : Vmware, VirtualBox, Xen, VirtualPC, Parallels etc.
- **Benefits of virtual machine :**
  1. There is no overlap amongst memory as each virtual memory has its ow memory space.
  2. Virtual machines are completely isolated from the host machine and oth virtual machines.
  3. Data does not leak across virtual machines.
  4. Can use multiple operating system environments on the same computer.
  5. The cost reduction is possible using small virtual servers on a more power single server.
- **Disadvantages of Virtual Machine**
  1. Virtual machines are less efficient than real machines because they access th hardware indirectly.
  2. A virtual machine can be infected with the weaknesses of the host machine.
  3. Difficulty in direct access to hardware, for example, specific cards or US devices.
  4. Great use of disk space, since it takes all the files for each operating syste installed on each virtual machine.

2.5 Roles of Client and Server

2.5.1 Client

- A major task of client machines is to provide the means for users to interact wit remote servers. For each remote service the client machine will have a separat counterpart that can contact the service over the network.
- A second solution is to provide direct access to remote services by only offering convenient user interface.
- **Example : The X Window System**
- It is based on a client/server model : a networked computer or workstation run an X server, and client programs running on connected workstations reque services from the server. The server handles input and output devices an generates the graphical displays used by the clients.

- The X Window System displays information and applications in rectangular windows arrayed on a desktop - style screen, known as the root window.
- Fig. 2.5.1 shows X window system.
- The X Window System is a networked display system. A server component, the X server, is responsible for coordinating between all of the clients connected, taking input from the mouse and keyboard, and pushing pixels on the output.

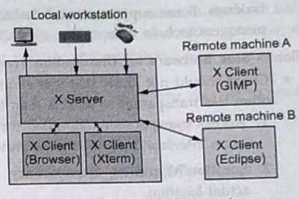

Fig. 2.5.1 X window system

- The X kernel offers a relatively low - level interface for controlling the screen, but also for capturing events from the keyboard and mouse. This interface is made available to applications as a library called Xlib.
- The X Window uses a bit - mapped display where each pixel can be manipulated individually. The entire display is known as the root window, and individual applications are displayed as windows on this root window.
- X was specifically designed to be used over network connections. X kernel (display machine) is the server; remote application is the client.
- It is possible to change the appearance of a window instantly by running a separate program after starting X. This program is called the window manager.
- X splits an application into two components : client and server.
- The server program controls the monitor, keyboard and mouse, while the application itself is the client. The X also runs in a TCP/IP network, it is possible for a client to run on one machine and have its display on another. The X-host client controls access to the server.
- The desktop system from which user run a program is called the X server. The system that hosts and executes the program is called the X client. This is the opposite of normal networking terminology.
- The X Window System is a Graphical User Interface (GUI) that runs many on UNIX and Linux systems. The top layer of the X Window System is the Windows Manager. Desktops are used with a Window Manager, providing specific appearance, applications, and resources.
- The window manager is a special X client that controls the placement and movement of applications, provides title bars and control buttons, menus and

taskbars. Some support virtual desktops and very fancy graphics. Classic windo managers include twm, mwm, olwm, fvwm.

Client - Side Software for Distribution Transparency

- Client should not know that it is communicated via network or not. Distribution often less transparent to servers than to clients.

1. **Access transparency :** Handled through client - side stubs for RPCs. It provid same interface as at the server and hides different machine architectures.
2. **Location/Migration transparency :** Let client - side software keep track actual location.
3. **Replication transparency :** Multiple invocations handled by client stub.
4. **Failure transparency :** It can often be placed only at client.

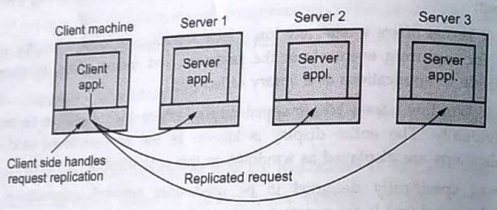

Fig. 2.5.2

2.5.2 Server

- A server is a process that waits for incoming service requests at a speci transport address. In practice, there is a one-to-one mapping between a port and service.
- Type of servers :
  1. Iterative servers : Handles a request itself; can handle only one client at a tim
  2. Concurrent servers : Does not handle a request itself; pass it to a separa thread or another process.
- Super servers : Servers that listen to several ports, i.e., provide sever independent services. In practice, when a service request comes in, they start sub-process to handle the request.
- There are several ways to organize servers :
  a) In the case of an iterative server, the server itself handles the request and, necessary, returns a response to the requesting client.

  b) A concurrent server can be multi-threaded or multi-process. It does not handle the request itself, but passes it to a separate thread or another process, after which it immediately waits for the next incoming request.
- A server can be stateless or stateful.
  a) A stateless server does not remember anything from one request to another. For example, a HTTP server is stateless.
  b) Stateful servers maintains information about its clients.

Design Issues for Servers

- Where clients contact a server ?
  a) In all cases, clients send requests to an end point, also called a port, at the machine where the server is running.
  b) Each server listens to a specific end point : Servers that handle Internet FTP requests always listen to TCP port 21. An HTTP server for the www listen to TCP port 80.
- How to handle communication interrupts ?
  a) Use out-of-band data. Example: to cancel the upload of a huge file.
  b) Server listens to separate endpoint, which has higher priority, while also listening to the normal endpoint (with lower priority).
  c) Send urgent data on the same connection. It can be done with TCP, where the server gets a signal on receiving urgent data.

2.6 Code Migration

- Code migration is the movement of programming code from one machine to other machine. The most complex example of code migration is migrating to an entirely new platform and/or operating system. This not only changes the programming language, but also the machine code behind the language.
- In distributed system, code migration is in the form of process migration.
- Migrating a process to another node in a DS might induce a lot of migration overhead and later follow up costs. Migrating from a heavily loaded node to a lightly/loaded one might improve overall system performance.
- A search query can be implemented a small program, moving from node to node collecting all search results. A client processing a very large amount of data from a specific server may be better off executing on the server machine.

2.6.1 Reasons for Migrating Code

- Following are the reasons for migrating code :
  1. Code migration improves system performance by balancing the load.

2. It also improves application performance by minimizing communication move code closer to the data. For example : move the database qu close to the database.
3. Improve application performance by executing multiple copies of the sa code (Web search).
4. Make application more flexible.
5. Dynamically configure a distributed application.
6. A change from the traditional client/server approach.

- Fig. 2.6.1 shows principle of dynamically configuring a client to communicate to server.

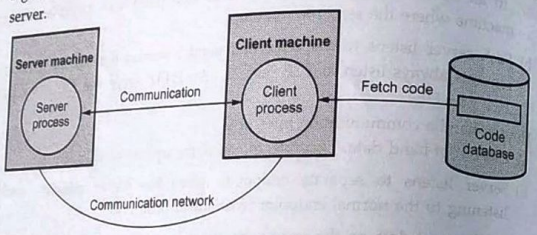

Fig. 2.6.1 Dynamically configuring a client to communicate to a server

- Client first fetches necessary software for future interaction with the server, then invokes the server.
  1. Code segment : It contains the actual code.
  2. Data segment : It contains the state.
  3. Execution state : It contains context of thread executing the object's code.

Models for code migration

- **Weak mobility :** It refers to transfer of code segment only. Program is alway starting from initial state. It is simple method and only requires code is portable It is relatively simple, especially if code is portable. Example of weak mobility i Java applets.
- **Strong mobility :** It is a transfer of execution segment. Any running process ca be stopped, restarted and transferred. This method is complex but powerful.
- **Sender-initiated :** Initiated by machine where code resides. For example uploading program to server such as database. Here security is main issue.
- **Receiver-initiated :** Easy to implement. The target machine takes initiative for code migration. Java applet is an example of receiver-initiated.

2.6.2 Migration and Local Resources

- An object uses local resources that may or may not be available at the target site.
- There is three types of process-to-resource binding happens in the system.

Object-to-resource binding :

- **Binding by identifier :** It executes as locally and remotely. The object requires a specific instance of a resource. For example : URL or IP address.
- **Binding by value :** It is available locally and remotely, but location might be different. Only value of resources is needed. For example : C library or Java library.
- **Binding by type :** It executes only available locally. The object requires that only a type of resource is available.

Resource types :

1. Fixed : The resource cannot be migrated, such as local hardware.
2. Fastened : The resource can, in principle, be migrated but only at high cost.
3. Unattached : The resource can easily be moved along with the object (e.g. a cache).

2.6.3 Migration in Heterogeneous Systems

- **Main problem :** The target machine may not be suitable to execute the migrated code. Distributed system is designed for heterogeneous system.
- The definition of process or thread or processor context is highly dependent on local hardware, operating system and runtime system.
  The solution for this is as follows :
- Weak mobility : No runtime information needs to be transferred, so it suffices to generate separate code segments for different target platforms.
- To make use of an abstract machine that is implemented on different platforms like interpreted languages running on a virtual machine. The virtual machine monitors, allowing migration of complete operating system and applications.
- How to transform the execution segment ? It is highly platform dependent.
- Each execution segments contains the current stack. To transfer an execution segment, make sure no platform dependent data is stored.
- Restrict code migration to specific points within the code, e.g. migration can take place only when a procedure is called; runtime system maintains a copy of the execution stack in a machine independent format-migration stack.

- Migration stack is updated each time a procedure is called or when a return fr the procedure occurs.
- Fig. 2.6.2 shows principle of maintaining a migration stack to support migration an execution segment in a heterogeneous distributed system.

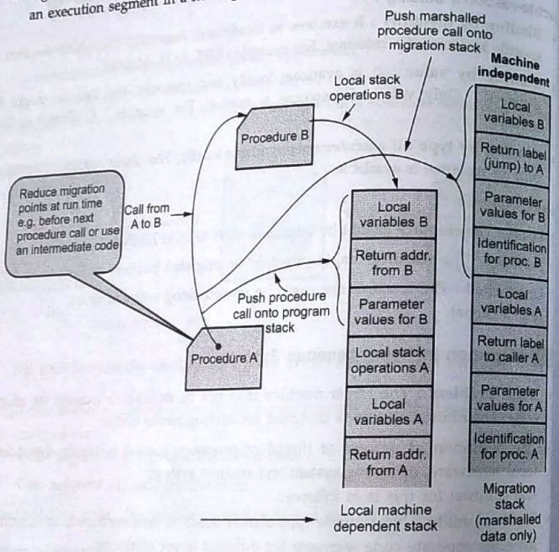

Fig. 2.6.2

2.7 Communication

- Distributed system is a system whose components are located on differ networked computers, which then communicate and coordinate their actions passing messages to one another.
- Communication does not come for free; often communication cost dominates cost of local processing or storage. Sometimes we even assume that everything communication is free.
- Here we discuss some types of communication.

2.7.1 Computer Network

Local Area Networks (LANs)

- LAN which span a limited area such as a company complex, a building, a campus, or even a small office. LANs are usually operated by a single organization. LAN support broadcasting.
- Local Area Networks are privately-owned networks within a small area, usually a single building or campus of up to a few kilometers. Since it is restricted in size, that means their data transmission time can be known in advance, and the network management would be easier.
- Fig. 2.7.1 shows local area network.

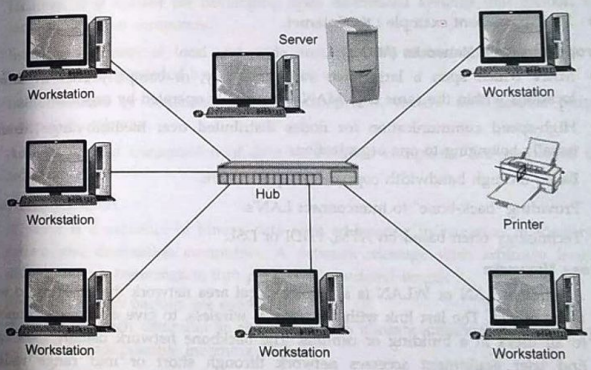

Fig. 2.7.1 LAN

- In many LAN such as Ethernet system bandwidth is the same as the data transfer rate. It carries high-speed communication on proprietary grounds and it is based on twisted copper wire, coaxial cable or optical fibre.
- Total system bandwidth is high and latency is low.
- Most typical solution : Ethernet with 100 Mbps.

Wide Area Networks (WAN)

- WAN which span a very large geographical area, such as from city to city or across countries and oceans. WANs are usually operated by transmission service providers.

- In most WAN since messages can be transferred in different channe simultaneously, total system bandwidth is different from transfer rate. T communication medium is a set of communication circuits linking a set dedicated computers called routers. They manage the communication network a route messages to their destinations.
- WAN support the communications over long distances. It also covers computers different organizations. WAN support point-to-point communication.
- WAN is high degree of heterogeneity of underlying computing infrastructure. involves routers to manage network and route messages to their destinations.
- It support speeds up to a few Mbps possible, but around 50-100 Kbps mo typical.
- Most prominent example : the Internet.

Metropolitan Area Networks (MANs)

- MAN which span a large area such as a city, or company sites in differe locations within the same city. MANs are usually operated by organizations.
- High-speed communication for nodes distributed over medium-range distance usually belonging to one organization.
- Based on high bandwidth copper and optical fibre.
- Providing "back-bone" to interconnect LAN's.
- Technology often based on ATM, FDDI or DSL.

Wireless Networks

- A wireless LAN or WLAN is a wireless local area network that uses radio wav as its carrier. The last link with the users is wireless, to give a network connectio to all users in a building or campus. The backbone network usually uses cable End user equipment accesses network through short or mid range radio infrared signal transmission.
- Wireless LANs operate in almost the same way as wired LANs, using the sam networking protocols and supporting the most of the same applications.
- Family of Wireless LAN (WLAN) specifications developed by a working group the Institute of Electrical and Electronic Engineers (IEEE). It defines standard f WLANs using the following four technologies :
  a. Frequency Hopping Spread Spectrum (FHSS)
  b. Direct Sequence Spread Spectrum (DSSS)
  c. Infrared (IR)
  d. Orthogonal Frequency Division Multiplexing (OFDM)

Different versions : 802.11a, 802.11b, 802.11g, 802.11e, 802.11f, 802.11i

802.11a offers speeds with a theoretically maximum rate of 54 Mbps in the 5 GHz band.

802.11b offers speeds with a theoretically maximum rate of 11 Mbps at in the 2.4 GHz spectrum band.

802.11g is a new standard for data rates of up to a theoretical maximum of 54 Mbps at 2.4 GHz.

Internetworks

- Several networks linked together to provide common data communication facilities. It is needed for developing open distributed systems that contain very large numbers of computers.
- Integrate a variety of local and wide area network technologies to provide the network capacity needed by each group of users.
- Interconnected by dedicated switching computers, *routers*, and general purpose computers, *gateways*.
- Addressing and transmission of data to included computers are supported by a software layer.

Packet Transmission

- A *packet* is a sequence of binary data with addressing information to identify the source and destination computers. A network message with arbitrary length is divided before transmission into packets of restricted length.
- Restricted length packets are used :
  a. To allow each computer in the network to allocate sufficient buffer storage to hold largest possible incoming packet.
  b. To avoid long waiting for communication channels to be free if long messages were transmitted without subdivision.

Switching Schemes

- A *switching* system is required to transmit information between two arbitrary nodes in the network using shared communications link.
- Long distance transmission is typically done over a network of switched nodes. Nodes not concerned with content of data.
- Four types of switching are used in computer network :
  1. **Broadcast :** It does not require switches. All messages are sent to all connected computers. Each computer is responsible extracting messages addressed to it. This approach is used in Ethernet and wireless networks.

2. **Circuit switching :** This approach used in the telephone system. A physi link is established between the sender and the receiver.
3. **Packet switching :** It also known as store-and-forward. At each switching no (connection point) a computer manages the packets by reading each one in memory, examining its destination, and choosing an outgoing circ appropriately.
4. **Frame relay :** Reading in and storing the whole of each packet introduces performance overhead which can become significant. In ATM networks frame of fixed size is used in place of a packet and only its header needs to examined. The remainder of the frame is simply relayed as a stream of bits.

2.7.2 Types of Communications

- Electronic mail system is an example of persistent communication. With **persiste communication**, a message that has been submitted for transmission is stored the communication middleware as long as it takes to deliver it to the receiver.
- Persistent communication : A message sent is stored by the communicati middleware until it is delivered to the receiver.
- In contrast, with **transient communication**, a message is stored by communication system only as long as the sending and receiving application executing.
- Transient communication : A message sent is stored by the communicati middleware only as long as both the receiver and the sender are executing.

Asynchronous vs. Synchronous Communication

- Asynchronous communication : The sender keeps on executing after sending message. The message should be stored by the middleware.
- Synchronous communication : The sender blocks execution after sending message and waits for response until the middleware acknowledges transmissi or, until the receiver acknowledges the reception, or, until the receiver completed processing the request.

Actual Communication in Distributed Systems

- In the practice of distributed systems, many combinations of persistency synchronisation are typically adopted.
- Persistency and synchronisation should then be taken as two dimensions alo which communication and protocols could be analysed and classified.

Discrete vs. Streaming Communication

- Communication is not always discrete, that is, it does not always happen throu complete units of information.

- Discrete communication is then quite common, but not the only way available and does not respond to all the needs. Sometimes, communication needs to be continuous through sequences of messages constituting a possibly unlimited amount of information.
- Streaming communication : The sender delivers a (either limited or unlimited) sequence of messages representing the stream of information to be sent to the receiver.

2.8 Remote Procedure Calls

- Remote Procedure Call (RPC), originally developed by Sun Microsystems and currently used by many UNIX-based systems, is an Application Programming Interface (API) available for developing distributed applications.
- It allows programs to execute subroutines on a remote system. The caller program, which represents the client instance in the client/server model sends a call message to the server process, and waits for a reply message.
- The call message includes the subroutine's parameters, and the reply message contains the results of executing the subroutine.
- RPC also provides a standard way of encoding data passed between the client servers in a portable fashion called External Data Representation (XDR).
- Traditionally the calling procedure is known as the client and the called procedure is known as the server.
- When making a remote procedure call :
  1. The calling environment is suspended, procedure parameters are transferred across the network to the environment where the procedure is to execute, and the procedure is executed there.
  2. When the procedure finishes and produces its results, its results are transferred back to the calling environment, where execution resumes as if returning from a regular procedure call.
- The main goal of RPC is to hide the existence of the network from a program. As a result, RPC doesn't quite fit into the OSI model :
  a. The message passing nature of network communication is hidden from the user. The user doesn't first open a connection, read and write data, and then close the connection. Indeed, a client often does not even know they are using the network.
  b. RPC often omits many of the protocol layers to improve performance. Even a small performance improvement is important because a program may invoke RPCs often. For example, on (diskless) Sun workstations, every file access is made via an RPC.

- RPC is especially well suited for client-server (e.g., query-response) interaction which the flow of control alternates between the caller and callee.
- Conceptually, the client and server do not both execute at the same time. Inste the thread of execution jumps from the caller to the callee and then back again.
- The procedure call (same as function call or subroutine call) is a well-kno method for transferring control from one part of a process to another, wit return of control to the caller.
- Associated with the procedure call is the passing of arguments from the caller (client) to the callee (the server).
- In most current systems the caller and the callee are within a single process o given host system. This is what we called "local procedure calls".
- In a RPC, a process on the local system invokes a procedure on a remote syste The reason we call this a "procedure call" is because the intent is to make appear to the programmer that a normal procedure call is taking place.
- We use the term "request" to refer to the client calling the remote procedure, a the term "response" to describe the remote procedure returning its result to t client.

2.8.1 RPC Model

- RPC model is similar to the well known and well understood procedure c model used for transfer of control and data within the program.
- RPC mechanism is an extension of the procedure call mechanism in the sense t it enables a call to be made to a procedure that does not reside in the addr space of the calling process. The called procedure may be on the same comput as the calling process or on a different computer.
- In case of RPC, the caller and the callee processes have disjoint address spaces, the remote process procedure has no access to data and variables of the caller's environment.
- RPC method uses message passing schemes for information exchange between the caller and the callee processes. Fig. 2.8.1 shows the typical model of remote procedure call.

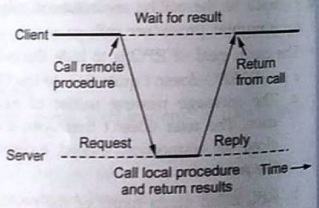

Fig. 2.8.1 RPC model

1. The client process send request message to the server process and waits for a reply message. The request message contains the remote procedure's parameters and other things.
2. Server process executes the procedure and then returns the result of procedure execution in a reply message to the client process.
3. Once the reply message is received, the result of procedure execution is extracted, and the caller's execution is resumed.

2.8.2 Transparency of RPC

- A transparent RPC mechanism is one in which local procedures and remote procedures are indistinguishable to programmers. RPC uses two types of transparency : Syntactic transparency and semantic transparency.
- Syntactic transparency means that a remote procedure call should have exactly the same syntax as a local procedure call. Semantic transparency means that the semantics of a remote procedure call are identical to those of a local procedure call.
- The remote procedure calls differ from local procedure calls in the following ways :
  1. The use of global variables is not possible as the server has no access to the caller program's address space.
  2. Performance may be affected by the transmission times.
  3. User authentication may be necessary.
  4. The location of the server must be known.

2.8.3 Implementing RPC Mechanism

- The idea behind RPC is to make a remote procedure call look as much as possible like a local one. In the simplest form, to call a remote procedure, the client program must be bound with a small library procedure called the client stub that represents the server procedure in the client's address space. Similarly, the server is bound with a procedure called the server stub. These procedures hide the fact that the procedure call from the client to the server is not local.
- Basically, a client-side stub is a procedure that looks to the client as if it were a callable server procedure. A server-side stub looks to the server as if it's a calling client.
- The client program thinks it is calling the server; in fact, it's calling the client stub. The server program thinks it's called by the client; in fact, it's called by the server stub. The stubs send messages to each other to make the RPC happen.

- Fig. 2.8.2 shows the steps in RPC.

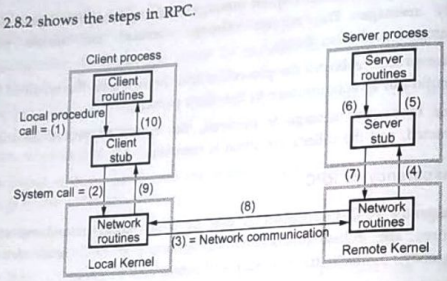

Fig. 2.8.2 Steps in RPC

1. The client calls a local procedure, called the client stub. It appears to the client that the client stub is the actual server procedure that it wants to call. The purpose of the stub is to package the arguments for the remote procedure, possibly put them into some standard format and then build one or more network messages. The packaging of the client's arguments into a network message is termed marshaling.
2. These network messages are sent to the remote system by the client stub. This requires a system call to the local Kernel.
3. The network messages are transferred to the remote system. Either a connection-oriented or a connection-less protocol is used.
4. A server stub procedure is waiting on the remote system for the client's request. It unmarshals the arguments from the network message and possibly converts them.
5. The server stub executes a local procedure call to invoke the actual server function, passing it the arguments that it received in the network messages from the client stub.
6. When the server procedure is finished, it returns to the server stub with return values.
7. The server stub converts the return values, if necessary, and marshals them into one or more network messages to send back to the client stub.
8. The messages get transferred back across the network to the client stub.
9. The client stub reads the network messages from the local Kernel.

10. After possibly converting the return values, the client stub finally returns to the client function. This appears to be a normal procedure return to the client.

2.8.4 Stub Generation

- Stubs can be generated in two ways : Manually and automatically.
- In manually method, user can construct the stub using set of translation function provided by RPC implementer. This method is simple to implement and can handle very complex parameters types.
- More commonly used method for stub generation is automatic method. It uses Interface Definition Language (IDL). IDL is used to define the interface between client and a server.

2.8.5 RPC Messages

- The RPC protocol can be implemented on any transport protocol. In the case of TCP/IP, it can use either TCP or UDP as the transport vehicle. When using UDP, it does not provide reliability. Thus, it is the responsibility of the caller program to employ any needed reliability (using time-outs and retransmissions, usually implemented in RPC library routines). Note that even with TCP, the caller program still needs a time-out routine to deal with exceptional situations, such as a server crash or poor network performance.

1. **RPC call message :** Each remote procedure call message contains the following unsigned integer fields to uniquely identify the remote procedure. Fig. 2.8.3 shows the RPC call message format.

Message identifier	Message type	Client identifier	Remote procedure identifier			Arguments
			Program number	Version number	Procedure number	

Fig. 2.8.3 RPC call message format

   a. Message identifier field consists of a sequence number.
   b. Message type field that is used to distinguish call messages from reply messages.
   c. Client identification field that may be used for two purposes.

2. **RPC Reply Message :** The RPC protocol for a reply message varies depending on whether the call message is accepted or rejected by the network server. The reply message to a request contains information to distinguish the following conditions :

- RPC executed the call message successfully.
- The remote implementation of RPC is not protocol version 2. The lowest an highest supported RPC version numbers are returned.
- The remote program is not available on the remote system.
- The remote program does not support the requested version number. The lowe and highest supported remote program version numbers are returned.
- The requested procedure number does not exist. This is usually a caller-sid protocol or programming error.

2.8.6 Marshalling Arguments and Result

- Parameters must be marshalled into a standard representation. Parameters consis of simple types (e.g., integers) and compound types (e.g., C structures or Pasca records). Moreover, because each type has its own representation, the types of th various parameters must be known to the modules that actually do the conversion For example, 4 bytes of characters would be uninterrupted, while a 4-byte intege may need to the order of its bytes reversed.
- Marshalling is the packing of procedure parameters into a message packet. The RPC stubs call type-specific procedures to marshal or unmarshal all of the parameters to the call.
- On the client side, the client stub marshals the parameters into the call packet; on the server side the server stub unmarshals the parameters in order to call the server's procedure.
- On the return, the server stub marshals return parameters into the return packet; the client stub unmarshals return parameters and returns to the client.

2.8.7 Server Management

- Servers are of two types : Stateful server and stateless server. Classification of server is based on implementations of the server.

Stateful server : It maintains the client's state information for one remote procedure call to the next. Following are the file operation supported by this server.

1. ***Open (filename, mode)*** : Used to open the a file identified by filename in the specified mode.
2. ***Read (fid, n, buffer)*** : Used to get n bytes of data from the file identified by fid into the buffer named buffer.
3. ***Write (fid, n, buffer)*** : After executing this operation , the server takes n bytes of data from the specified buffer.

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 statements causes the server to delete from its file table the file state information of the file identified by fid.

- **Stateless server** : This server does not maintain any client state information. Following are the file operations supported by this server.
  1. **Read (filename, position, n, buffer)** : On execution of this statement, the server returns to the n bytes of data of the file identified by filename.
  2. **Write (filename, position, n, buffer)** : On execution of this statement, it takes n bytes of data from the specified buffer and writes it into the file identified by filename.

Client-Server binding

- Binding is the process of connecting the client and server. The server, when it starts up, exports its interface, identifying itself to a network name server and telling the local runtime its dispatcher address.
- The client, before issuing any calls, imports the server, which causes the RPC runtime to lookup the server through the name service and contact the requested server to setup a connection. The import and export are explicit calls in the code.

2.8.8 RPC Problems

- RPC works really well if all the machines are homogeneous.
- Complications arise when the two machines use different character encodings, e.g. EBCDIC or ASCII.
- Byte-ordering is also a problem : Intel machines are little-endian and Sun Sparc's are big-endian.
- Extra mechanisms are required to be built into the RPC mechanism to provide for these types of situations - this adds complexity.

2.8.9 Call Semantics

- An RPC implementation may support more than one set of semantics for the RPC call. Which call semantics are used by a developer depends on the requirements of the application.
- A client makes an RPC to a service at a given server. After a time-out expires, the client may decide to resend the request. If after several tries there is no success, what may have happened depends on the call semantics :

1. Maybe call semantics

- After a RPC time-out (or a client crashed and restarted), the client is not sure the RP may or may not have been called.
- This is the case when no fault tolerance is built into RPC mechanism.
- Clearly, maybe semantics is not desirable.

2. At-least-once call semantics

- With this call semantics, the client can assume that the RP is executed at le once.
- Can be implemented by retransmission of the (call) request message on time-out
- Acceptable only if the server's operations are idempotent. That is $f(x) = f(f(x))$.

3. At-most-once call semantics

- When a RPC returns, it can assume that the Remote Procedure (RP) has be called exactly once or not at all.
- Implemented by the server's filtering of duplicate requests and caching of replies
- This ensures the RP is called exactly once if the server does not crash duri execution of the RP.
- When the server crashes during the RP's execution, the partial execution may le to erroneous results.
- In this case, we want the effect that the RP has not been executed at all.
- At-most-once call semantics are for those RPC applications which require guarantee that multiple invocations of the same RPC call by a client will not processed on the server.
- Such applications usually maintain state information on the server and more th one invocation of the same RPC call must be detected in order to avoid corrupti of the state information.

2.8.10 Lightweight Remote Procedure Call

- Lightweight Remote Procedure Call (LRPC) is a communication facility design and optimized for communication between protection domains on the sa machine.
- In contemporary small-Kernel operating systems, existing RPC systems incur unnecessarily high cost when used for the type of communication t predominates between protection domains on the same machine.

- By reducing the overhead of same-machine communication, LRPC encourages both safety and performance.
- LRPC combines the control transfer and communication model of capability systems with the programming semantics and large-grained protection model of RPC.
- Server S exports interface to remote procedures and client C on same machine imports interface. OS Kernel creates data structures including an argument stack shared between server and client.
- RPC execution
  1. Push arguments onto stack.
  2. Trap to Kernel.
  3. Kernel changes memory map of client to server address space.
  4. Client thread executes procedure.
  5. Thread traps to Kernel upon completion.
  6. Kernel changes the address space back and returns control to client

2.9 Message-Oriented Communication

- In distributed system, communication is hiding from user by using RPC and RMI. But it is true that any mechanism is not proper.
- Message-oriented communication is a way of communicating between processes.
- Message-oriented communications are of two types : synchronous or asynchronous communication, and transient or persistent communication.

2.9.1 Persistence and Synchronicity in Communication

- Let us consider the computer network where the applications are executed on the host machine. Host machine is connected to the network of the communication server. This server is responsible for passing messages between the hosts.
- Persistence is a prerequisite for certain forms of communication and sharing.
- E-mail system is an example of persistent communication.
- Each host runs an application by which a user can compose, send, receive and read messages. Each host is connected to a mail server and every message is first stored in one of the output buffers of the local mail server.
- The server removes messages from its buffers and sends them to their destination. The target mail server stores the message in an input buffer for the designated

receiver. The interface at the receiving host offers a service to the receiver's u agent by which the latter can regularly check for incoming mail.

- In **synchronous communication**, the sender blocks waiting for the receiver engage in the exchange. **Asynchronous communication** does not require both sender and the receiver to execute simultaneously. So, the sender and recipient a loosely-coupled.
- Client/Server computing is generally based on a model of synchrono communication: Client and server have to be active at the time of communicatio Client issues request and blocks until it receives reply.
- Server essentially waits only for incoming requests, and subsequently proces them.

Drawbacks of synchronous communication

1. Client cannot do any other work while waiting for reply
2. Failures have to be dealt with immediately (the client is waiting)
3. In many cases the model is simply not appropriate (mail, news)

- The amount of time messages are stored determines whether the communication transient or persistent. **Transient** communication stores the message only wh both partners in the communication are executing. If the next router or receiver not available, then the message is discarded. It works like a tradition store-and-forward router.
- **Persistent** communication, on the other hand, stores the message until t recipient receives it.
- A typical example of asynchronous persistent communication is Message-Orient Middleware (MOM). Message-oriented middleware is also called message-queuing system, a message framework, or just a messaging system.
- MOM can form an important middleware layer for enterprise applications on t Internet. In the publish and subscribe model, a client can register as a publisher a subscriber of messages. Messages are delivered only to the relevant destinatio and only once, with various communication methods including one-to-many many-to-many communication. The data source and destination can be decouple under such a model.
- **Persistent Asynchronous Communication :** Each message is either persistentl stored in a buffer at the local host or at the first communication server. T e-mail system is an example.

- **Persistent Synchronous Communication :** Messages can be persistently stored at the receiving host and a sender is blocked until this happens.
- **Transient Asynchronous Communication :** The message is temporarily stored at a local buffer at the sending host, after which the sender immediately continues. UDP is an example for this type.
- **Transient Synchronous Communication :** The sender is blocked until the message is stored in a local buffer at the receiving host, or until the message is delivered to the receiver for further processing, or until it receives a reply message from the other side.

2.9.2 Message Oriented Transient Communication

1. Socket

- Socket interface is a protocol independent interface to multiple transport layer primitives. In order to write applications which need to communicate with other applications.
- Socket is an abstraction that is provided to an application programmer to send or receive data to another process.
- Data can be sent to or received from another process running on the same machine or a different machine.
- It is like an endpoint of a connection. It exists on either side of connection and identified by IP Address and Port number.
- Sockets works with UNIX I/O services just like files, pipes and FIFO.
- API stands for Application Programming Interface. It is an interface to use the network. Socket API defines interface between application and transport layer.
- The API defines function calls to create, close, read and write to/from a socket.

Advantages of using socket interface

- Syntax of the API functions is independent of the protocol being used. Ex:- TCP/IP and UNIX domain protocols can be used by applications using a common set of functions.
- Gives way to better portability of applications across protocol suites.
- Hides the finer details of the protocols from application programs thereby yielding faster and bug free application development.

- Sockets are referenced through socket descriptors which can be passed direct UNIX system I/O calls. File I/O and socket I/O are exactly similar from programmer perspective.

Sockets versus file I/O

- Working with sockets is very similar to working with files. The socket ( ) accept ( ) functions both return handles (file descriptor) and reads and writ the sockets requires the use of these handles (file descriptors).
- In Linux, sockets and file descriptors also share the same file descriptor table. is, if you open a file and it returns a file descriptor with value say 8, and immediately open a socket, you will be given a file descriptor with value reference that socket.
- Even though sockets and files share the same file descriptor table, they are very different. Sockets have addresses associated with them whereas files do notice that this distinguishes sockets form pipes, since pipes do not have addr with which they associate.
- You cannot randomly access a socket like you can a file with lseek ( ). So must be in the correct state to perform input or output.

Socket abstraction

- Socket is the basic abstraction for network communication in the socket Socket defines an endpoint of communication for a process.
- Operating system maintains information about the socket and its connec Fig. 2.9.1 shows the socket and process.

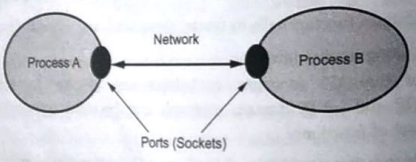

Fig. 2.9.1 Socket and process

Socket Creation

int socket (int family, int type, int protocol);

Parameters :

1. family	:	AF_INET or PF_INET (These are the IP4 family)
2. type	:	SOCK_STREAM (for TCP) or SOCK_DGRAM (for UDP)
3. protocol	:	IPPROTO_TCP (for TCP) or IPPROTO_UDP (for UDP) or use 0

- If successful, socket ( ) returns a socket descriptor, which is an integer, and – 1 in the case of a failure.
- An example call :

```
if ((sd = socket(AF_INET, SOCK_DGRAM, 0) < 0)
{
    printf(socket() failed.);
    exit(1);
}
```

- Creating a socket is in some ways similar to opening a file. This function creates a file descriptor and returns it from the function call. You later use this file descriptor for reading, writing and using with other socket functions.
- Remember that the sockets API are generic. There must be a generic way to specify endpoint addresses. TCP/IP requires an IP address and port number for each endpoint address. Other protocol suites (families) may use other schemes.

Elementary TCP sockets

- In UNIX, whenever there is a need for IPC within the same machine, we use mechanism like signals or pipes. When we desire a communication between two applications possibly running on different machines, we need **Sockets**.
- Sockets are treated as another entry in the UNIX open file table.
- Sockets provide an interface for programming networks at the transport layer.
- Network communication using Sockets is very much similar to performing file I/O. In fact, socket handle is treated like file handle.
- Socket-based communication is programming language independent.
- To the kernel, a socket is an endpoint of communication. To an application, a socket is a file descriptor that lets the application read/write from/to the network.

- A server (program) runs on a specific computer and has a socket that is bound a specific port. The server waits and listens to the socket for a client to mak connection request.
- To review, there are five significant steps that a program which uses TCP m take to establish and complete a connection. The server side would follow th steps :
  1. Create a socket.
  2. Listen for incoming connections from clients.
  3. Accept the client connection.
  4. Send and receive information.
  5. Close the socket when finished, terminating the conversation.
- In the case of the client, these steps are followed :
  1. Create a socket.
  2. Specify the address and service port of the server program.
  3. Establish the connection with the server.
  4. Send and receive information.
  5. Close the socket when finished, terminating the conversation.
- Only steps two and three are different, depending on if it's a client or ser application.
- Fig. 2.9.2 shows a timeline of the typical scenario that takes place between a T client and server.
  (Refer Fig. 2.9.2 on next page).

2. Message passing interface

- MPI primarily addresses the message-passing parallel programming model : d is moved from the address space of one process to that of another process throu cooperative operations on each process.
- A communicator is a collection of processes that can send messages to each oth There is a default communicator whose group contains all initial processes, cal MPI_COMM_WORLD. A process is identified by its rank in the group associa with a communicator.
- MPI is hardware independent. It is designed for parallel applications.

Reasons for using MPI :

1. Standardization : MPI is the only message passing library which can be consider a standard.

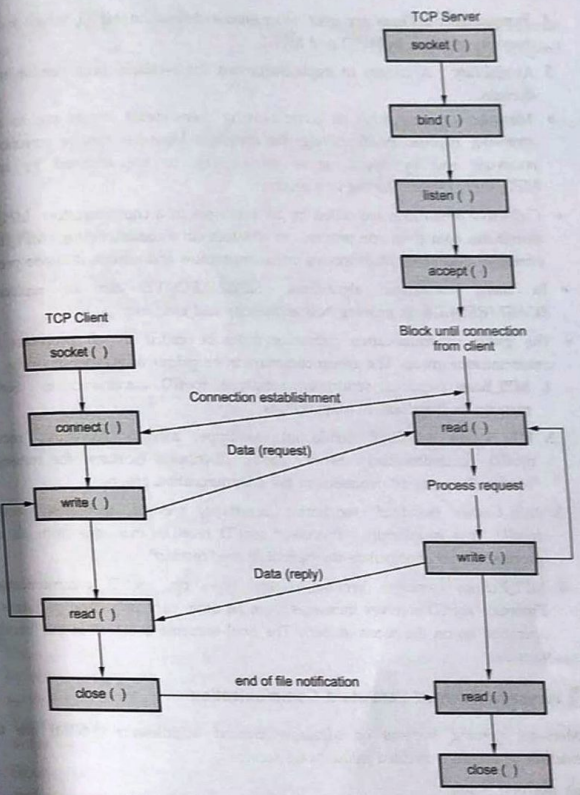

Fig. 2.9.2 Socket function for elementary TCP client server

2. Portability : There is little or no need to modify your source code when you port your application to a different platform that supports the MPI standard.
3. Performance Opportunities : Vendor implementations should be able to exploit native hardware features to optimize performance.

4. Functionality : There are over 440 routines defined in MPI-3, which includes
majority of those in MPI-2 and MPI-1.
5. Availability : A variety of implementations are available, both vendor and pu
domain.

- Messages are sent with an accompanying user-defined integer tag, to assist
receiving process in identifying the message. Messages can be screened at
receiving end by specifying a specific tag, or not screened by specify
MPI_ANY_TAG as the tag in a receiver.
- Collective operations are called by all processes in a communicator. MPI_BCA
distributes data from one process to all others in a communicator. MPI_REDU
combines data from all processes in communicator and returns it to one proces
- In many numerical algorithms, SEND/RECEIVE can be replaced
BCAST/REDUCE, improving both simplicity and efficiency.
- The group communication primitive must be called by all processes in
communicator group. The group communication primitive is synchronously.
1. MPI_Bcast (sendbuf, sendcount, sendtype, rootID, communicator) : Send
message in "sendbuf" to all processes.
2. MPI_Scatter (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvty
rootID, communicator) : Sender rootID distributes (scatters) the message
"sendbuf" among all processes in the communication group.
3. MPI_Gather (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvty
rootID, com municator) : Processor rootID receives messages from all ot
processors and concatenate the output in the "recvbuf".
4. MPI_Reduce (sendbuf, recvbuf, count, type, op, rootID, communicator
Processor rootID receives messages from all other processors and performs
operation op on the received data. The final outcome is stored in the "recvb
variable.

2.9.3 Message Oriented Persistent Communication

- Message queuing systems or Message-Oriented Middleware (MOM) are
example of message oriented middleware service.

Message queuing model

- It supports asynchronous persistent communication. It provides an intermedi
storage for message while sender/receiver is inactive. Example application is em
system.
- It communicates by inserting messages in queues. The sender is only guaran
that message will be eventually inserted in recipient's queue.

- It is example of loosely coupled communication. Sender and receiver can execute completely independently of each other. Four combination of loosely coupled communications using queues are as follows :

1. Sender running and receiver running

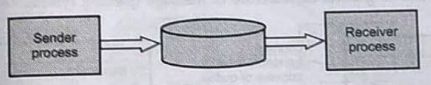

Fig. 2.9.3

2. Sender running and receiver passive

Fig. 2.9.4

3. Sender passive and receiver running

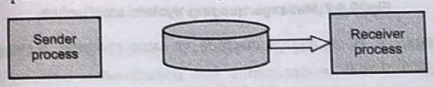

Fig. 2.9.5

4. Sender passive and receiver passive

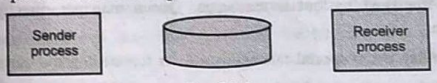

Fig. 2.9.6

- Message queuing allows distributed applications to communicate asynchronously by sending messages between the applications. The messages from the sending application are stored in a queue and are retrieved by the receiving application. The applications send or receive messages through a queue by sending a request to the message queuing system.
- Sending and receiving applications can use the same message queuing system or different ones, allowing the message queuing system to handle the forwarding of the messages from the sender queue to the recipient queue.
- Queued messages can be stored at intermediate nodes until the system is ready to forward them to the next node. At the destination node, the messages are stored in a queue until the receiving application retrieves them from the queue.

- Message delivery is guaranteed even if the network or application fails. T provides for a reliable communication channel between the applications.

General architecture of message queuing system

- Fig. 2.9.7 shows general architecture of message queuing system.

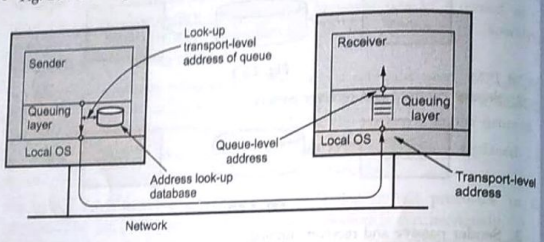

Fig. 2.9.7 Message queuing system architecture

- Source queue : It is on source machine or same machine. Message can be re only from local queue.
- Destination queue : Message is stored on the queue where queue contai specification of the destination.
- Queue is managed by queue managers. Queue manager directly interact wi application.
- Router or relay is the special manager used for forwarding the messages.
- It works at the application level.

Message broker

- A Message Queue broker provides delivery services for a Message Queu messaging system. Message delivery relies upon a number of supporti components that handle connection services, message routing and deliver persistence, security, and logging.
- A message server can employ one or more broker instances. Broker componen are shown in Fig. 2.9.8.
- Message delivery in a Message queue messaging system from producing clients t destinations, and then from destinations to one or more consuming clients performed by a broker.

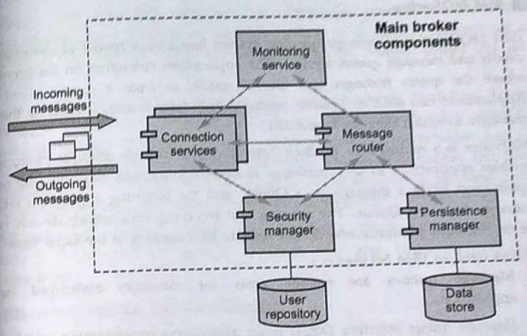

Fig. 2.9.8 Message broker components

- To perform message delivery, a broker must set up communication channels with clients, perform authentication and authorization, route messages appropriately, guarantee reliable delivery, and provide data for monitoring system performance.
- To perform this complex set of functions, a broker uses a number of different internal components, each with a specific role in the delivery process.
- The Message Router component performs the key message routing and delivery service, and the others provide important support services upon which the Message Router depends.

Main broker service components and functions

1. Message Router : Manages the routing and delivery of messages.
2. Connection Services : Manages the physical connections between a broker and clients, providing transport for incoming and outgoing messages.
3. Persistence Manager : Manages the writing of data to persistent storage so that system failure does not result in failure to deliver messages.
4. Security Manager : Provides authentication services for users requesting connections to a broker and authorization services for authenticated users.
5. Monitoring Service : Generates metrics and diagnostic information that can be written to a number of output channels that an administrator can use to monitor and manage a broker.

2.9.4 IBM MQSeries

- IBM MQSeries is a message queuing system based on a model of message queue clients and message queue servers. The applications run either on the server node where the queue manager and queues reside, or from a remote client node. Applications can send or retrieve messages only from queues owned by the queue manager to which they are connected.
- MQSeries is a middleware product from IBM that runs on multiple platforms and enables applications to send messages to other applications. Basically, the sending application PUTs a message on a Queue, and the receiving application GETs the message from the Queue. The sending and receiving applications do not have to be on the same platform, and do not have to be executing at the same time
- **Terms used in IBM MQSeries :**
  1. **Message queues are** storage areas for messages exchanged between applications.
  2. Message queue interface (MQI) is an application programming interface for applications that want to send or receive messages through IBM MQSeries queues.
  3. MQSeries clientconfiguration is an MQSeries configuration where the queue manager and message queues are located on a different computer or node than the application software.
  4. **MQSeries serverconfiguration** is an MQSeries configuration where the queue manager and message queues are located on the same (local) computer or node as the application software.
  5. **Queue manager** provides the message queuing facilities that applications use, and manages the queue definitions, configuration tables, and message queues.
  6. **Triggers** is an MQSeries feature that enables an application to be started automatically when a message event, such as the arrival of a message, occurs.
- **Application-specific messages** are put into, and removed from **queues**. Queues always reside under the regime of a **queue manager**. Processes can put messages only in local queues, or through an RPC mechanism.

Message transfer :

- Messages are transferred between queues.
- Message transfer between queues at different processes, requires a **channel**.
- At each endpoint of channel is a **Message Channel Agent (MCA)**. It is used for setting up channels using lower-level network communication facilities and wrapping messages from/in transport-level packets.

- Channels are inherently unidirectional. MQSeries provides mechanisms to automatically start MCAs when messages arrive, or to have a receiver set up a channel . Any network of queue managers can be created; routes are set up manually.
- A channel provides a communication path between Queue Managers. There are two types of channels - Message Channels and MQI channels (also called Client channels). Message channels provide a communication path between two queue managers on the same, or different, platforms. A message channel can transmit messages in one direction only. If two-way communication is required between two queue managers, two message channels are required.
- **MQI channels** connect an MQSeries client to a queue manager on a server machine. It is used for transfer of MQI calls and responses only and is bi-directional.

2.10 Stream-Oriented Communication

- Message oriented communication implies request-response and can only be used when communication speed does not affect correctness. But timing is crucial in certain forms of communication. Solution is stream oriented communication.
- Following concept is used for this :
  1. Support for continuous media
  2. Streams in distributed systems
  3. Stream management
- All communication facilities discussed so far are essentially based on a discrete, which is time-independent exchange of information.
- In many situations, it does not matter when a particular communication process takes place. But what if we are attempting to serve audio or video, or a combination of both? Time dependent data can be served using **streams.**
- Streams can be simple and complex. Streams support single sink and multiple sinks.
- Multimedia systems use **stream-oriented** communications. The timing of the data delivery is critical in such systems. Such communication is used for **continuous media** such as audio where the temporal relationships between different data items are meaningful as opposed to **discrete media** such as text.
- Continuous representation media : The temporal relationship between data items is important to the meaning of the data. Examples include video and audio data.
- Discrete representation media : Not time dependent. Example : still images, text, executables.

- Data streams handle continuous media. Data stream is a sequence of data units.
- **Data streams** have several modes
  1. **Asynchronous transmission mode** places no timing constraints on the data items in a stream.
  2. **Synchronous transmission mode** gives a maximum end-to-end delay for each item in a data stream.
  3. **Isochronous transmission mode** gives both maximum and minimum delays.
- Streams can be either **simple** or **complex**. Related sub-streams will need to be synchronized. Streams can be seen as a channel between a source and a sink. Source could be a file or multimedia capture device and sink could be a file or multimedia rendering device.
- Fig. 2.10.1 shows setting up a stream between two processes across a network. Streams can be set up between different machines, or directly between devices, or both. Streams are unidirectional and there is generally a single source, and one or more sinks.

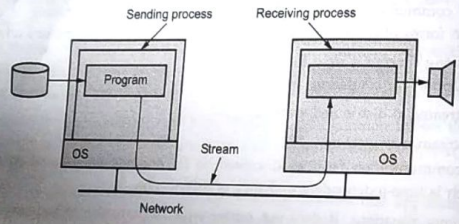

Fig. 2.10.1

Streams and QoS

- Time-dependent requirements are generally expressed as Quality of Service (QoS) requirements. Streams are all about timely delivery of data. How do you specify this Quality of Service (QoS)? Basics :
  1. The required bit rate at which data should be transported.
  2. The maximum delay until a session has been set up.
  3. The maximum end-to-end delay.
  4. The maximum delay variance, or jitter.
  5. The maximum round-trip delay.

Flow specification :

Sr. No.	Characteristics of the input
1	Maximum data unit size (bytes)
2	Token bucket rate (bytes/sec)
3	Token bucket size (bytes)
4	Maximum transmission rate (bytes/sec)

Sr No.	Service required
1	Loss sensitivity (bytes)
2	Loss interval (μsec)
3	Burst loss sensitivity (data units)
4	Minimum delay noticed (μsec)
5	Maximum delay variation (μsec)
6	Quality of guarantee

Token Bucket Algorithm

- In token bucket bursts of up to n packets can be sent at once, which gives faster response to sudden bursts of input.
- The regulator collects tokens in a bucket, which fills-up at steady drip rate by packets. When a packet arrives at the regulator, the regulator sends the packet if the bucket has enough tokens. Otherwise, the packet waits either until the buckets has enough tokens.
- Fig. 2.10.2 shows token bucket generator.

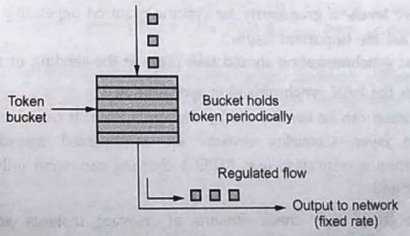

Fig. 2.10.2

- If the bucket is already fill of tokens, incoming tokens overflow and are n available to future packets. Thus, at any time, the largest burst a source can sen into the network is roughly proportional to the size of the leaky bucket.
- The regulator delays a packet if does not have sufficient number of tokens fo transmission. A counter keeps track of tokens, the counter is incremented by on every ΔT and decremented by one whenever a packet is sent.
- When the counter hits zero, no packets may be sent. Smoother traffic can b obtained by putting a leaky bucket after the token bucket.

The Resource Reservation Protocol (RSVP)

- **RSVP** is a protocol for enabling resource reservations in network routers.
- RSVP allows multiple senders to transmit to multiple groups of receivers, permit individual receivers to switch channels freely and optimizes bandwidth use whil at the same time eliminating congestion.
- RSVP uses multicast routing using spanning trees.
- Group address is assigned to each group. To send to a group a sender puts th group address in its packets. The standard multicast routing algorithm then build a spanning tree covering all group members.
- RSVP process receives and stores the specification parameters, checks if there an available resources and checks if receiver has permission to make the reservation
- RSVP is not a routing protocol and its works in conjunction with routing protocols.

Stream Synchronization

- An important issue is that different streams must be synchronized :
  1. Continuous with discrete
  2. Continuous with continuous (more difficult)
  3. Different levels of granularity for syncing required depending on situation
- Following are the important issues :
  1. Whether synchronization should take place at the sending or receiving side ?
  2. What is the local synchronization specification ?
- Synchronization can be carried out by the application. It can also be supplied by a middleware layer. Complex streams are multiplexed according to a given synchronization specification (e.g. MPEG). Syncing can occur either at the sending or receiving end.
- Monitor programs that check streams at relevant instants and adjust rate if necessary. Multimedia middleware systems offer a collection of interfaces to control and synchronize stream.

2.11 Multicasting

- Multicast communication allows a process to send the same message to a group of processes. As multicast operations can provide the programmer with delivery guarantees that are difficult to realize for the application programmer using ordinary unicast operations.
- Group communication that simplifies building reliable efficient distributed systems. Most current distributed operating systems are based on Remote Procedure Call. The idea is to hide the message passing and make the communication look like an ordinary procedure call. Fig. 2.11.1 shows multicast communication.

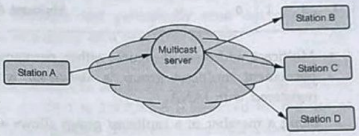

Fig. 2.11.1 Multicast communication

- Multicast messages provides a useful infrastructure for constructing distributed systems with the following characteristics :
  1. **Replicated services :** A replicated service consists of a group of members. Client requests are multicast to all the members of the group, each of which performs an identical operation. Even when some of the members fail, clients can still be served.
  2. **Better performance :** Performance of service is increase by using data replication. User's computer is used for replication. Each time the data change, the new value is multicast to the processes managing the replicas.
  3. **Propagation of event notifications :** Multicast to a group may be used to notify processes when something happens. For example, a news system might notify interested users when a new message has been posted on a particular newsgroup.
- Group view is the lists of the current group members. When a membership change occurs, the application is notified of the new membership.

2.11.1 IP Multicast

- IP multicast is built on top of the Internet Protocol. IP multicasting is to allow a device on an IP internetwork to send datagram's not to just one recipient but to an arbitrary collection of other devices.

- A multicast group is specified by a Class D Internet address. Class D addresses are reserved for multicast : 224.0.0.0 to 239.255.255.255 and are used as group addresses.
- A multicast group is specified by an Internet address whose first 4 bits are 1110 in IPv4.

1	1	1	0	Multicast Address (28 bits)

- Multicast addresses designate either permanent or transient multicast groups. A permanent multicast group is associated with an IP multicast address that is registered with IANA.
- Being a member of a multicast group allows a computer to receive IP packets sent to the group. IP multicast is a bandwidth-conserving technology that reduces traffic by simultaneously delivering a single stream of information to potentially thousands of corporate recipients and homes.
- A multicast address is designed to enable the delivery of datagrams to a set of hosts that have been configured as members of a multicast group in various scattered sub-networks.
- Applications that take advantage of multicast include video conferencing, corporate communications, distance learning and distribution of software, stock quotes and news.
- A host may send IP multicast by using a multicast address as the destination address. A host manages a table of groups and local application processes that belong to this group. When a multicast message arrives at the host, it delivers copies of it to all of the local processes that belong to that group.
- A host acts as a member of a group only if it has at least one active process that joined that group. The membership of multicast groups is dynamic, allowing computers to join or leave at any time and to join an arbitrary number of groups. It is possible to send datagrams to a multicast group without being a member.
- IP multicast uses on UDP protocol. An application program performs multicasts by sending UDP datagrams with multicast addresses and ordinary port numbers. An application program can join a multicast group by making its socket join the group, enabling it to receive messages to the group.
- Addresses in the range from 224.0.1.0 through 238.255.255.255 are called globally scoped addresses. These addresses are used to multicast data between organizations and across the Internet.

- Some of these addresses have been reserved for use by multicast applications through IANA. For example, IP address 224.0.1.1 has been reserved for Network Time Protocol (NTP).
- **Internet Group Management Protocol (IGMP) is** used to dynamically register individual hosts in a multicast group on a particular LAN. Hosts identify group memberships by sending IGMP messages to their local multicast router. Under IGMP, routers listen to IGMP messages and periodically send out queries to discover which groups are active or inactive on a particular subnet.
- **Multicast routers** execute a multicast routing protocol to define delivery paths that enable the forwarding of multicast datagrams across an internetwork. The block of multicast addresses ranging from 224.0.0.1 to 224.0.0.255 is reserved for the use of routing protocols and other low-level topology discovery or maintenance protocols. Multicast routers should not forward a multicast datagram with a destination address in this range, regardless of its TTL.

Multicast address allocation

Sr No.	IP Address	Use
1.	224.0.0.0 to 224.0.0.225	Local Network Control Block for multicast traffic within a given local network.
2.	224.0.1.0 to 224.0.1.225	Internet Control Block
3.	224.0.2.0 to 224.0.255.0	Ad Hoc Control Block for traffic that does not fit any other block.
4.	239.0.0.0 to 239.255.255.255	Administratively Scoped Block which is used to implement a scoping mechanism for multicast traffic

- Multicast addresses may be permanent or temporary.

Failure model for multicast datagram's

- They suffer from omission failures.
- Omission Failures : messages dropped, checksum errors, lack of buffer space etc.
- Both send-omissions and receive-omissions can occur in multicast datagram.
- This can be called unreliable multicast, because it does not guarantee that a message will be delivered to any member of a group. Messages can arrive out-of-order.
- Applications that use UDP need to provide their own checks
- **Advantage :** efficient data distribution