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.

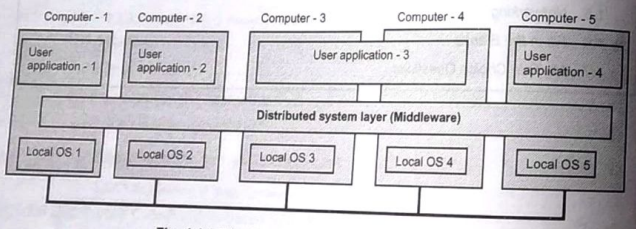


Fig. 1.1.1 Distributed system organized as middleware

- 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.

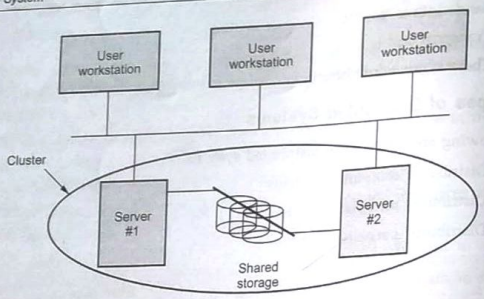


Fig. 1.3.1 Cluster computing

- 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

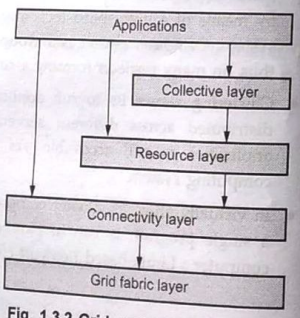


Fig. 1.3.2 Grid protocol architecture

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.

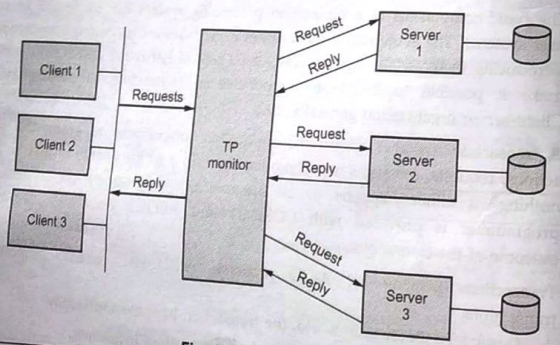


Fig. 1.3.3 TP monitor

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.

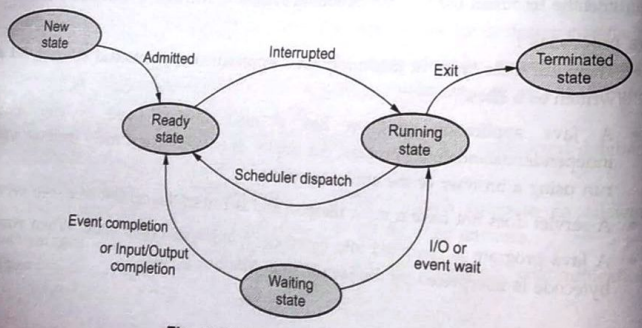


Fig. 1.4.1 Process state diagram

- 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.

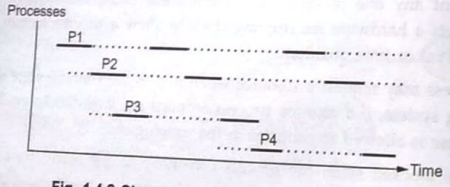


Fig. 1.4.2 Shows timesharing of a resource

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.

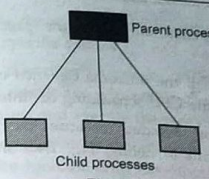


Fig. 1.4.3

- 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.

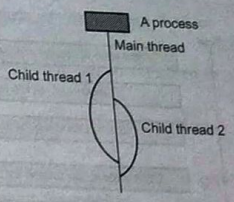


Fig. 1.4.4

- 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.

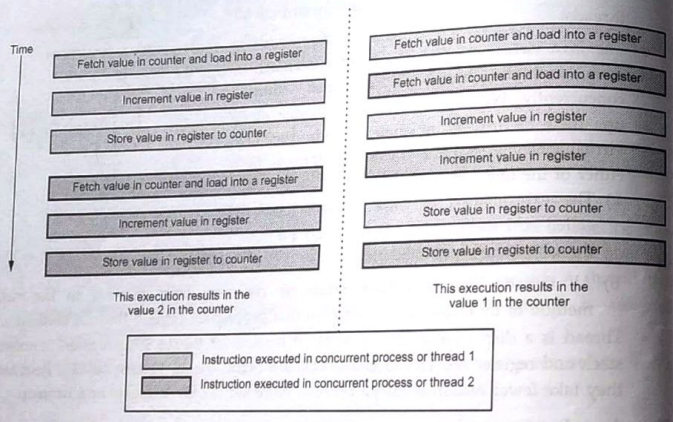


Fig. 1.4.5 Race condition between threads

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.

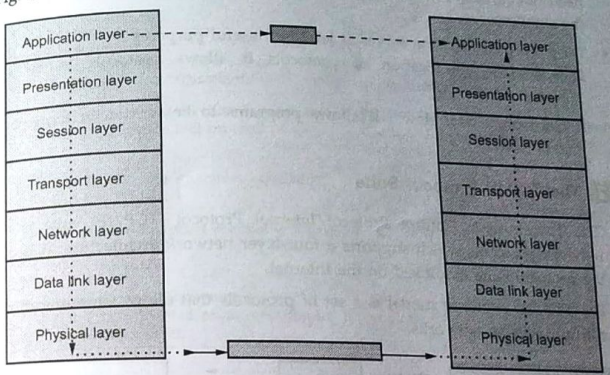


Fig. 1.5.1 OSI seven-layer network architecture

- 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.

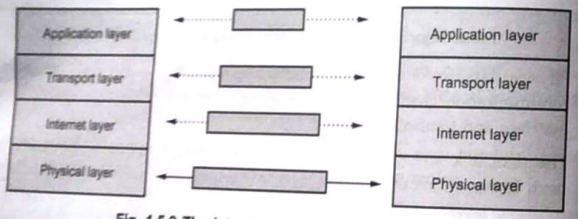


Fig. 1.5.2 The Internet network architecture

- 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.

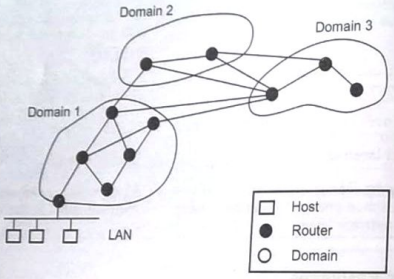


Fig. 1.5.3 Internet topology

- 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.

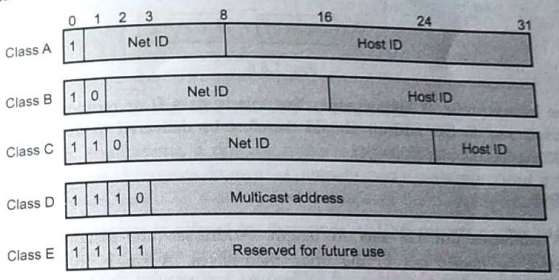


Fig. 1.5.4 IP address

- 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 :

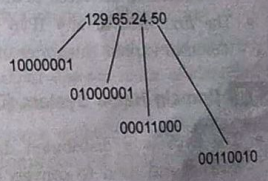


Fig. 1.5.5

- 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 :

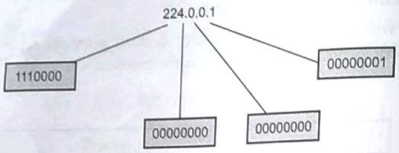


Fig. 1.5.6

- 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

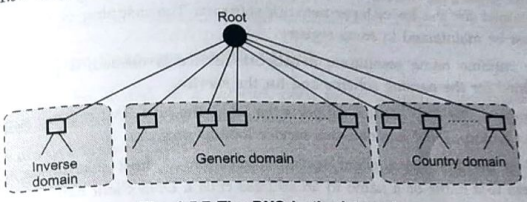


Fig. 1.5.7 The DNS in the Internet

- 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.

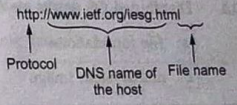


Fig. 1.5.8

- 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.

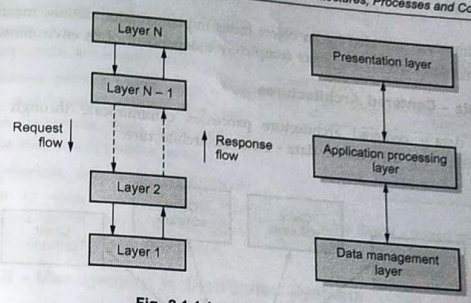


Fig. 2.1.1 Layered architecture

- 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.

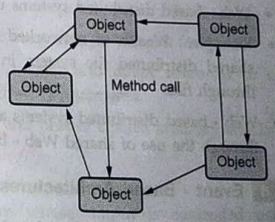


Fig. 2.1.2 Object based architectural style

- 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.

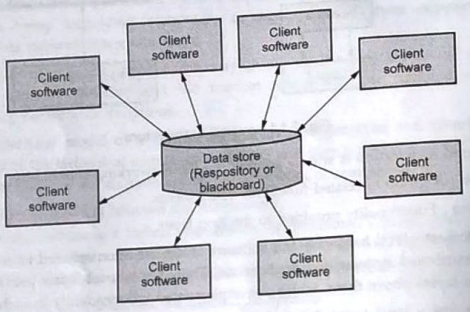


Fig. 2.1.3 Data-centered architecture

- 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**.

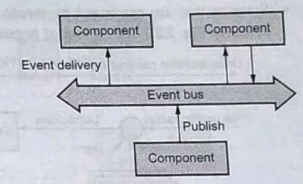


Fig. 2.1.4 Event - based architecture

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.

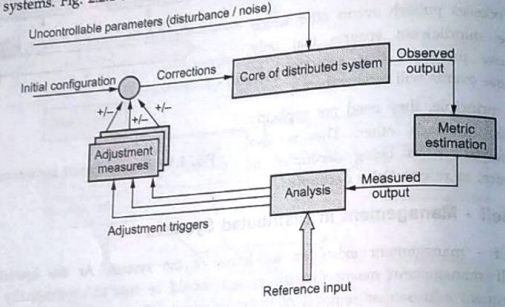


Fig. 2.2.1 Feedback control system

- 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.

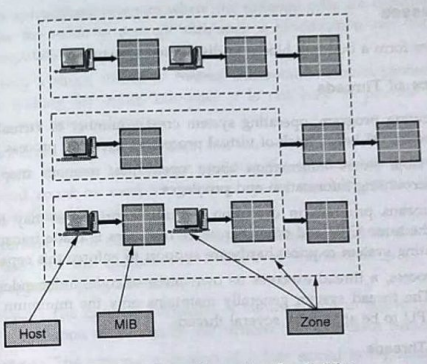


Fig. 2.2.2 Zone hierarchy of astrolabe

- 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.

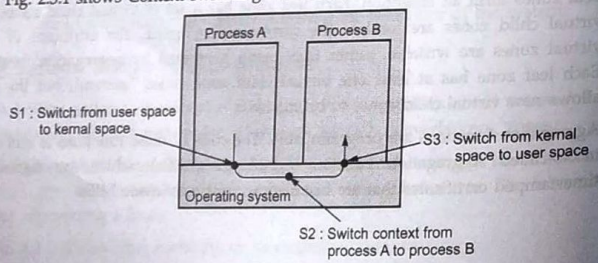


Fig. 2.3.1

- 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.

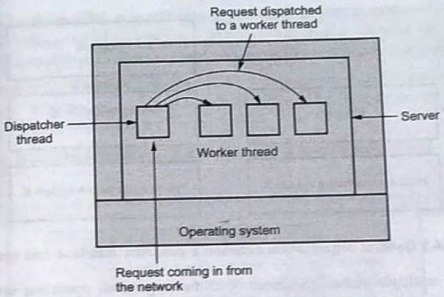


Fig. 2.3.2 Multithreaded server organized in a dispatcher/worker model

- 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.

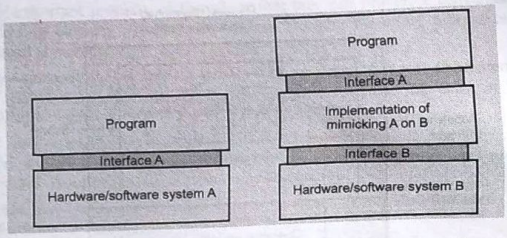


Fig. 2.4.1 General organization between a program, interface and system

- 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.

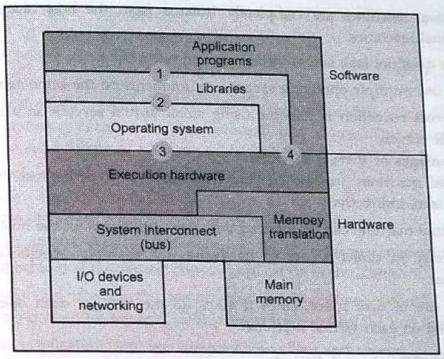


Fig. 2.4.2 Various interfaces offered by computer system

- 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.

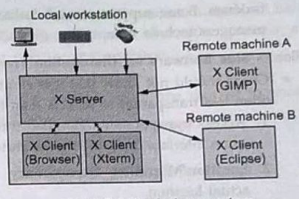


Fig. 2.5.1 X window system

- 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.

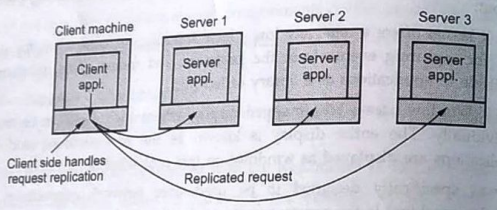


Fig. 2.5.2

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.

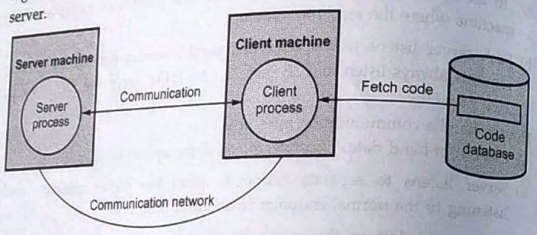


Fig. 2.6.1 Dynamically configuring a client to communicate to a server

- 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.

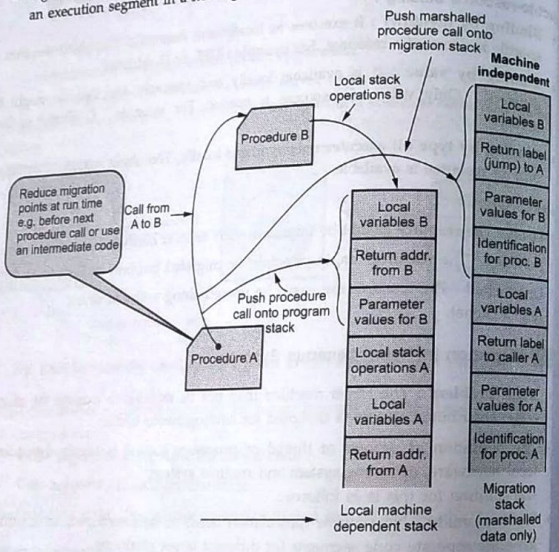


Fig. 2.6.2

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.

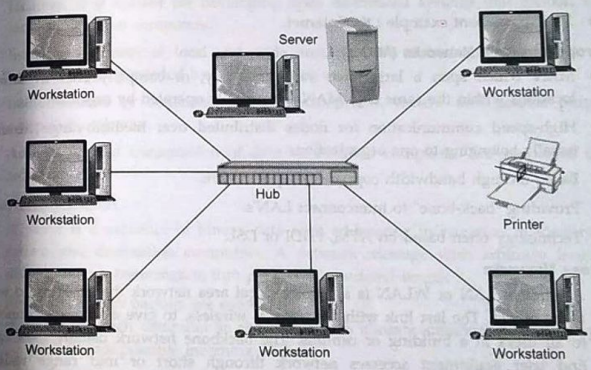


Fig. 2.7.1 LAN

- 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.

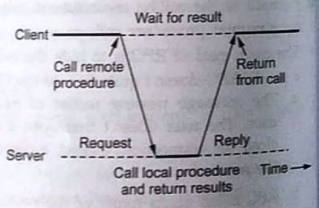


Fig. 2.8.1 RPC model

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.

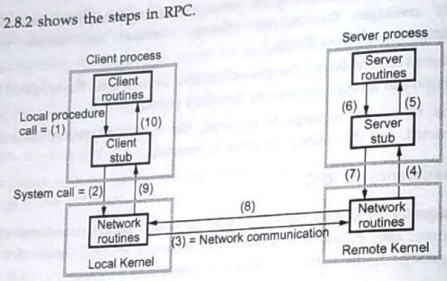


Fig. 2.8.2 Steps in RPC

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.

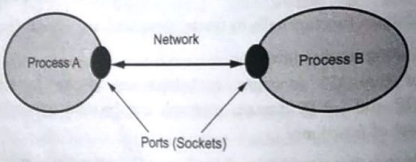


Fig. 2.9.1 Socket and process

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.

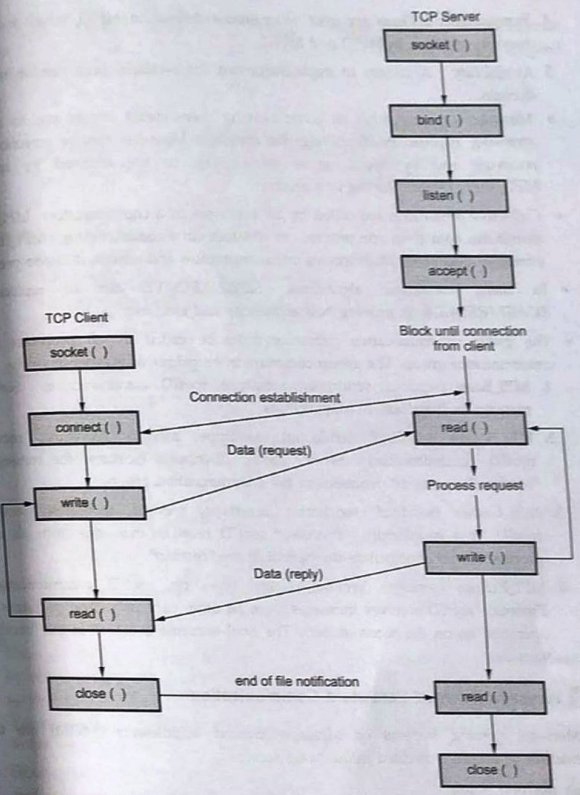


Fig. 2.9.2 Socket function for elementary TCP client server

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

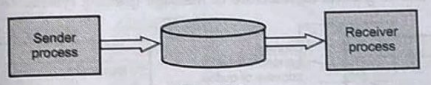


Fig. 2.9.3

2. Sender running and receiver passive

Fig. 2.9.4

3. Sender passive and receiver running

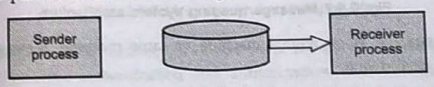


Fig. 2.9.5

4. Sender passive and receiver passive

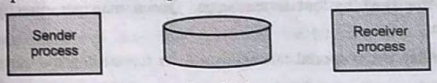


Fig. 2.9.6

- 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.

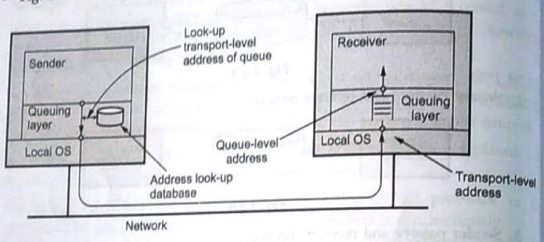


Fig. 2.9.7 Message queuing system architecture

- 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.

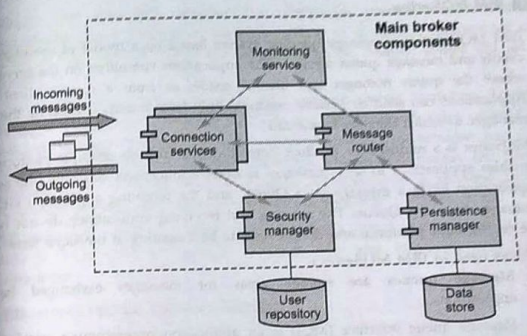


Fig. 2.9.8 Message broker components

- 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.

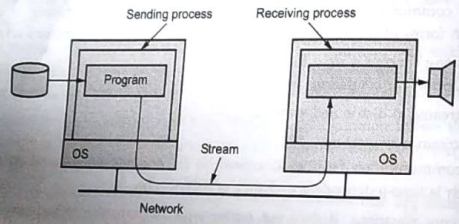


Fig. 2.10.1

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.

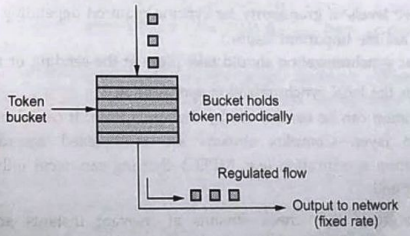


Fig. 2.10.2

- 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.

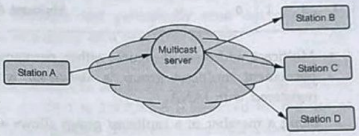


Fig. 2.11.1 Multicast communication

- 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

3 Naming

Syllabus

Names, Identifiers, And Addresses, Flat Naming, Structured Naming, Attribute-Based Naming.

Contents

3.1 Names, Identifiers and Addresses

- An entity can be identified by three types of references.
  a) **Name :** A name is a set of bits or characters that references an entity. Names can be human-friendly.
  b) **Address :** Every entity resides on an access point, and access point has an address. Addresses may be location-dependent.
  c) **Identifier :** Identifiers are names that *uniquely* identify entities. A *true identifier* is a name with the following properties :
     1) An identifier refers to at-most one entity
     2) Each entity is referred to by at-most one identifier
     3) An identifier always refers to the same entity (i.e. it is never reused)

3.1.1 Naming Overview

- A **name** in file systems is associated with an object. **Name resolution** refers to the process of mapping a name to an object. **Name space** is a collection of names which may or may not share an identical resolution mechanism.
- Name service is used by client processes to obtain attributes of resources or objects when given their names.
- The entities named can be : users, computers, network domains, services and remote objects. Names facilitate communication and resource sharing. Descriptive attributes are another mean of identification. Client doesn't know the name of entity, but knows information that describes it. Client requires a service rather than a particular entity that implements it.
- Any process requiring access to a specific resource must poses a name or identifier for that resource. For example : file names, URLs, remote object reference.
- A name in a distributed system is a string of bits or characters used to refer to an entity. To resolve names a naming system is needed. Names are used to share resources, uniquely identify entities and refer to locations. Naming system plays a very important role in achieving the goal of location transparency in a distributed system.
- The naming facility of a distributed operating system enables users and programs to assign character-string names to objects and subsequently use these names to refer to those objects. The locating facility, which is an integral part of the naming facility, maps an object's name to the object's location in a distributed system.

- The naming and locating facilities jointly form a naming system that provides the users with an abstraction of an object that hides the details of how and where an object is actually located in the network.
- It provides a further level of abstraction when dealing with object replicas. Given an object name, it returns a set of the locations of the object's replicas. The naming system plays a very important role in achieving the goal of :
  1. Location transparency,
  2. Facilitating transparent migration and replication of objects,
  3. Object sharing

3.1.1.1 Desirable Features of a Good Naming System

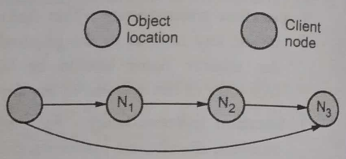
- Features of good naming system are as follows :
  1. **Location transparency :** This feature implies the name of an object should not reveal any hint as to the physical location of the object, directly or indirectly. An object's name should be independent of the physical connectivity or topology of the system, or the current location of the object.
  2. **Location independency :** For performance, reliability, availability and security reasons, distributed systems provide the facility of object migration that allows the movement and relocation of objects dynamically among the various nodes of a system. Location independency means that the name of an object need not be changed when the object's location changes. An object at any node can be accessed without the knowledge of its physical location. An object at any node can issue an access request without the knowledge of its own physical location.
  3. **Scalability :** Distributed systems vary in size ranging from one with a few nodes to one with many nodes. Distributed systems are normally open systems and their size changes dynamically.
  4. **Uniform naming convention :** In the most of the distributed systems, different naming conventions are used for naming different types of objects. For example, file names typically differ from user names and process names. Instead of using such non-uniform naming conventions, a good naming system should use the same naming convention for all types of objects in the system.
  5. **Multiple user-defined names for the same object :** For a shared object, it is desirable that different users of the object can use their own convenient names for accessing it. Therefore, a naming system must provide the flexibility to assign multiple user-defined names to the same object.
  6. **Group naming :** A naming system should allow many different objects to be identified by the same name. Such a facility is useful to support broadcast facility or to group objects for conferencing or other applications.

7. **Meaningful names :** A name can be simply any character string identifying some object. However, for users, meaningful names are preferred to lower level identifiers such as memory pointers, disk block numbers or network addresses.

8. **Performance :** The performance measurement of a naming system is the amount of time needed to map an object's name to its attributes, such as its location. Naming system should be efficient in the sense that the number of messages exchanged in a name-mapping operation should be as small as possible.

9. **Fault tolerance :** A naming system should be capable of tolerating, to some extent, faults that occur due to the failure of a node or a communication link in a distributed system network. That is, the naming system should continue functioning, perhaps in a degraded form, in the event of these failures.

10. **Replication transparency :** In a distributed system, replicas of an object are generally created to improve performance and reliability. A naming system should support the use of multiple copies of the same object in a user-transparent manner.



- The cost is high if the object locating mechanism maps to node N_3 instead of node N_1.

3.2 Flat Naming

- A name space is a collection of all valid names recognized by a particular service. It allows simple but meaningful names to be used.
- A naming space can be represented as a graph with leaf nodes (containing entity information) and directory nodes. Absolute and relative path names are related to a directory node.
- A global name denotes the same entity in the system. A local name depends on where the name is being used.
- Four types of name resolution mechanisms for flat names :

  a) Broadcasting

  b) Forwarding pointers

  c) Home-based approaches

  d) Distributed Hash Tables (DHTs)

- Name space is classified into two types :
  1. Flat name space
  2. Partitioned name space

Flat Name Space

- A name structure that uses only a single attribute for the names is called a flat naming structure. Conceptually simple but unique naming is difficult to achieve without global coordination.
- Flat names are suitable for use either for small name spaces having names for only a few objects or for system-oriented names that need not be meaningful to the users.
- Advantage : Names were convenient and short.
- Disadvantages :
  1. Single central name authority was overloaded.
  2. Flat name spaces cannot generalize to large sets of machines because of the single set of identifiers.
  3. Frequent name address binding changes were costly and cumbersome.

3.2.1 Broadcasting and Multicasting

- A computer that wants to access another computer for which it knows its IP address broadcasts this address.
- The principle used in the Address Resolution Protocol (ARP) to find the data-link address of a machine when given only an IP address. In essence, a machine broadcasts a packet on the local network asking who is the owner of a given IP address.
- When the message arrives at a machine, the receiver checks whether it should listen to the requested IP address. If so, it sends a reply packet containing, for example, its Ethernet address.
- Broadcasting becomes inefficient when the network grows. Not only is network bandwidth wasted by request messages, but, more seriously, too many hosts may be interrupted by requests they cannot answer. One possible solution is to switch to multicasting.
- Multicasting is better when the network grows - send only to a restricted group of hosts. Multicasting can also be used to locate the nearest replica - choose the one whose reply comes in first.

Forwarding Pointers

- To locate mobile entities, concept of forwarding pointers is used. Forwarding pointers enable locating mobile entities. Mobile entities move from one access point to another.
- When an entity moves from place A to another place B, it leaves behind (at A) a reference to its new location at B.
- **Advantage**
  1. **Simple :** As soon as the first name is located using traditional naming service, the chain of forwarding pointers can be used to find the current address.
- **Drawbacks**
  1) The chain can be too long - locating becomes expensive.
  2) All the intermediary locations in a chain have to maintain their pointers.
  3) Vulnerability if links are broken.
- Hence, making sure that chains are short and that forwarding pointers are robust is an important issue.

3.2.2 Home-based Approaches

- Another approach to support mobile entities. A home keep track of where the entity is :
  a) An entity's home address is registered at a naming service.
  b) The home registers the foreign address of the entity.
  c) Clients always contact the home first, and then continues with the foreign location.
- Problems with home-based approaches.
  a) Home address has to be supported as long as the entity lives.
  b) Home address is fixed, which means an unnecessary burden when the entity permanently moves to another location.
  c) Poor geographical scalability.

3.2.3 Distributed Hash Tables

- Distributed hash tables are a form of a distributed database that can store and retrieve information associated with a key in a network of peer nodes that can join and leave the network at any time.
- A Distributed Hash Table (DHT) is a class of a decentralized distributed system that provides a lookup service similar to a hash table: (key, value) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key.

- Chord is a protocol and algorithm for a peer-to-peer distributed hash table. A distributed hash table stores key-value pairs by assigning keys to different computers (known as "nodes"); a node will store the values for all the keys for which it is responsible.
- Chord specifies how keys are assigned to nodes, and how a node can discover the value for a given key by first locating the node responsible for that key. Chord assigns an m-bit identifier (randomly chosen) to each node. A node can be contacted through its network address.
- The **Chord** protocol supports just one operation : Given a key, it will determine the node responsible for storing the key's value. Chord does not itself store keys and values.
- A node generates its identifier by picking a value randomly from the hash space. The node joins the DHT and determines who its predecessor and successor are in the table.
- Predecessor(n) : The node with the highest identifier less than n's identifier, allowing for wraproud.
- Successor(n) : The node with the lowest identifier greater than n's identifier, allowing for wraproud.
- A node is then responsible for its own identifier and the identifiers between its identifier and its predecessor's identifier. Fig. 3.2.1 shows identifier circle for 3 bit identifier.

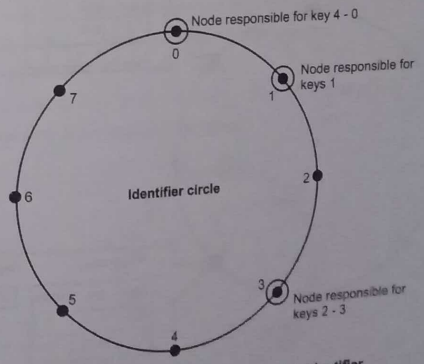


Fig. 3.2.1 Identifier circle for 3 bit identifier

- Internally, chord uses a consistent hash function for mapping keys to node locations. The consistent hash function of chord is based on standard hash functions like SHA1 that produces m bit output. The nodes are hashed based on their IP address, while key, value pair is hashed based on their key.
- An identifiers are arranged on a identifier circle modulo $2^m \Rightarrow$ **Chord ring**
- Key (k) is assigned to the node whose identifier is equal to or greater than the key's identifier. This node is called successor(k) and is the first node clockwise from k. The identifier ring is called *Chord ring*.
- Key k is assigned to the first node whose identifier is equal to or follows (the identifier of) k in the identifier space. This node is the successor node of key k, denoted by successor(k).
- If each node knows only how to contact its current successor node on the identifier circle, all node can be visited in linear order. Queries for a given identifier could be passed around the circle via these successor pointers until they encounter the node that contains the key.
- Fig. 3.2.2 shows chord with finger table.

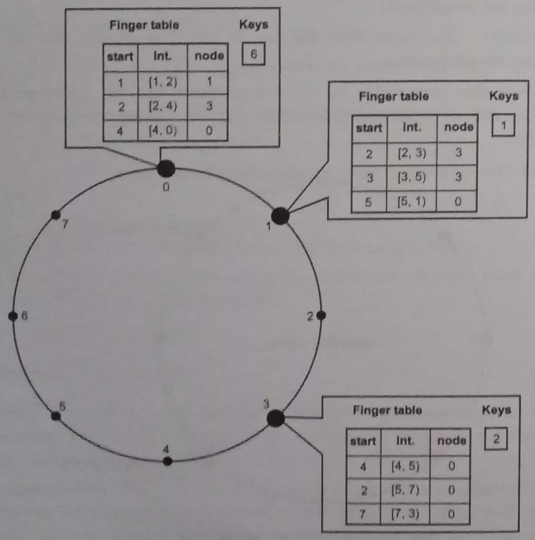


Fig. 3.2.2 Chord with finger table

- The i^{th} entry of node n will contain successor $(n + 2^{i-1})$ mod 2^m. The first entry of finger table is actually the node's immediate successor.
- Every time a node wants to look up a key k, it will pass the query to the closest successor or predecessor (depending on the finger table) of k in its finger table (the "largest" one on the circle whose ID is smaller than k), until a node finds out the key is stored in its immediate successor.
- With such a finger table, the number of nodes that must be contacted to find a successor in an N-node network is O(log N)
- When a node n joins the network, certain keys previously assigned to n's successor now become assigned to n. When node n leaves the network, all of its assigned keys are reassigned to n's successor.
- Each node n maintains a routing table with upto m entries called *finger table.*
- The i^{th} entry in the table at node n contains the identity of the *first* node s that succeeds n by at least 2^{i-1} on the identifier circle.
- s = successor $(n + 2^{i-1})$.

  Where s is called the i^{th} *finger* of node *n*, denoted by *n.finger(i).*
- A finger table entry includes both the Chord identifier and the IP address (and port number) of the relevant node. The first finger of n is the immediate successor of n on the circle.

DHT construction

- Use a logical name space, called the identifier space, consisting of identifiers {0, 1, 2, …, N – 1}. Identifier space is a logical ring modulo N.
- Every node picks a random identifier though Hash H.
- Example : Space N = 16 {0,…,15}
- Five nodes a, b, c, d, e. H(a) = 6, H(b) = 5, H(c) = 0, H(d) = 11, H(e) = 2
- Fig. 3.2.3 shows chord ring and successor.

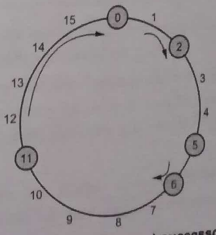


Fig. 3.2.3 Chord ring and successor

- The **successor** of an identifier is the first node met going in clockwise direction starting at the identifier.
- succ(x) : is the first node on the ring with id greater than or equal x. Succ(12) = 0, Succ(1) = 2 and Succ(6) = 6
- Each node points to its successor. The successor of a node n is succ(n+1). Fig. 3.2.4 shows node and its successor.
  i) The 0's successor is succ(1) = 2
  ii) 2's successor is succ(3) = 5
  iii) 5's successor is succ(6) = 6
  iv) 6's successor is succ(7) = 11
  v) 11's successor is succ(12) = 0
- The hashing scheme was designed to let nodes enter and leave the network with minimal disruption.
- To maintain the consistent hashing mapping when a node n joins the network, certain key value pairs previously assigned to n's successor become assigned to n.
- When node n leaves the network, all of its assigned key value pairs are reassigned to n's successor. No other changes in the assignment of key value pairs to nodes need occur.

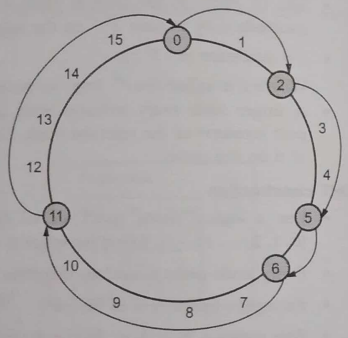


Fig. 3.2.4 Node and its successor

- This hashing function is straightforward to implement in a centralized environment where all machines are known.
- However, such a system does not scale. Therefore, the Chord protocol uses a distributed hash function, in which each node maintains a small routing table.
- Each node (n) maintains a routing table with 160 entries, called the finger table. The i^{th} entry in the table at node n contains a reference to the first node (s), that succeeds n by at least $2^{(i-1)}$ on the identifier circle, i.e., s = successor$(n+2^{(i-1)})$, where $1 \leq i \leq 160$.
- The node s is called the i^{th} finger of node n, and denoted by *n.finger[i].node*. The first finger of n is its immediate successor on the circle.

- When a node n does not know the successor of a key k, it sends a ``find successor'' request to a intermediate node whose ID is closer to k.
- Node n finds the intermediate node by searching its finger table for the closest finger f preceding k, and sends the find successor request to f.
- Node f looks in its finger table for the closest entry preceding k, and sends that back to n. As a result n learns about nodes closer and closer to the target ID.

Data Storing

- Fig. 3.2.5 shows data storing.
- Use globally known hash function, H.
- Each item <key, value> gets identifier H(key) = k.
- Example : H("Ravi") = 12, H("Rupu") = 2, H("Iresh") = 9, H("Ganesh") = 14, H("Vilas") = 4
- Store each item at its successor

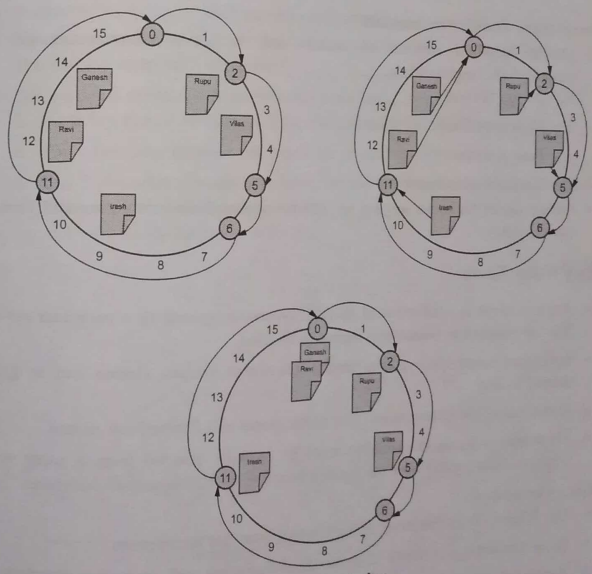


Fig. 3.2.5 Data storing

3.3 Structured Naming

- A name service stores a collection of one or more naming contexts - sets of bindings between textual names and attributes for objects. It provides a general naming scheme for entities (such as users and services) that are beyond the scope of a single service.
- Major operation is to resolve a name to look up attributes from a given name. Other operations required: creating new binding, deleting bindings, listing bound names and adding and deleting contexts.
- Name management is separated from other services.
  a. **Unification :** It is often convenient for resources managed by different services to use the same naming scheme.
  b. **Integration :** It is not always possible to predict the scope of sharing in a distributed system. Without a common name service, the administrative domains may use entirely different naming conventions.

General Name Service Requirements

- Handle arbitrary number of names and to serve arbitrary number of administrative organizations.
  a. A long lifetime
  b. High availability
  c. Fault isolation
  d. Tolerance of mistrust
- Design issues for name services are : Name spaces, Name resolution and Domain name system.

3.3.1 Name Spaces

- A name space is a collection of all valid names recognized by a particular service. It allow simple but meaningful names to be used.
- Structured name space allows similar sub-names without clashes and to group related names.
- Name spaces are of two types : Flat name spaces and Hierarchical names.
- The name assigned to machines must be carefully selected from a name space with complete control over the binding between the names and IP addresses.

i) Flat name spaces :

- The original set of machines on the Internet used flat namespaces.
- These namespaces consisted of sequence of characters with no further structure.
- A name is assigned to an address.

- **Advantage :**
  1. Names were convenient and short.
- **Disadvantages :**
  1. Flat name spaces cannot generalize to large sets of machines because of the single set of identifiers.
  2. Single central name authority was overloaded.
  3. Frequent name-address binding changes were costly and cumbersome.

ii) Hierarchical names :

- The partitioning of a namespace must be defined in such a way that it :
  - Supports efficient name mapping.
  - Guarantees autonomous control of name assignment.
- Hierarchical namespaces provides a simple yet flexible naming structure.
- The namespace is partitioned at the top level.
- Authority for names in each partition are passed to each designated agent.
- The names are designed in an inverted-tree structure with the root at the top.
- The tree can have only 128 levels.
- The top level domains are divided into three areas :
  1. Arpa is a special domain used for the address-to-name mappings.
  2. The 3 character domains are called the generic domains.
  3. The 2 character domains are based on the counter codes found in ISO 3166. These are called the country domains.
- Fig. 3.3.1 shows the hierarchy of DNS.

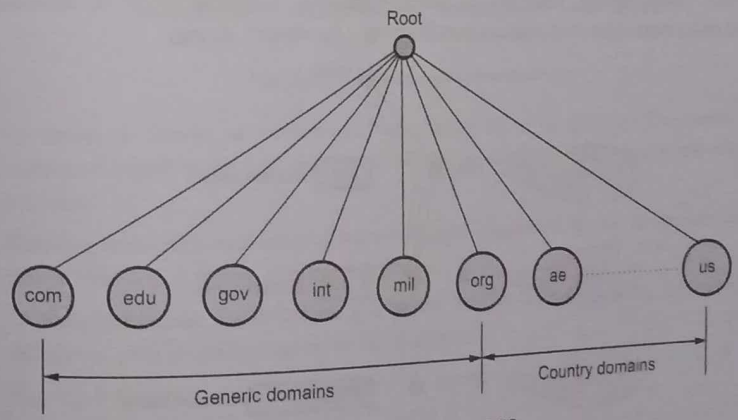


Fig. 3.3.1 Hierarchy of DNS

List of some domain names :

1. **Geographical domain names :**
   - AU Australia • BR Brazil
   - CA Canada • DE Germany
   - ES Spain • FI Finland
   - FR France • GR Greece
   - HU Hungary • IN India
   - IT Italy • JP Japan
   - MX Mexico • NL Netherlands
   - NO Norway • NZ New Zealand
   - SE Sweden • TR Turkey
   - UK United Kingdom • US United States
2. **Organizational domain names :**
   - COM US Commercial • EDU US Educational
   - GOV US Government • INT International
   - MIL US Military • NET Network
   - ORG Non-Profit Organization
   - ARPA Old Style Arpanet
   - NATO Nato field

- In DNS, names are defined in an inverted tree structure with the root at the top. The tree can have only 128 levels : Level 0 to Level 127.
- Each node in the tree has a label, which is a string with a maximum of 63 characters. The root label is a null string , i.e. empty string.

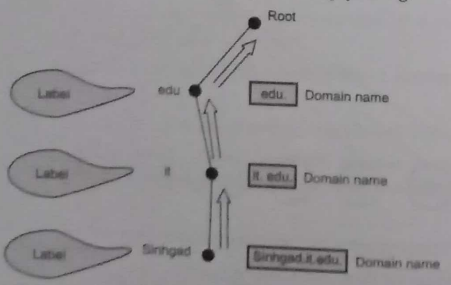


Fig. 3.3.2 Domain name and labels

Each node in the tree has a domain name, A full domain name is a sequence of labels separated by dots(.). Fig. 3.3.2 shows the domain names and labels.

- In fully qualified domain name, label is terminated by a null string. Fully Qualified Domain Name (FQDN) contains the full name of host. For example, **sinhgad.it.edu**
- If a label is not terminated by null switch it is called a Partially Qualified Domain Name (PQDN). It starts from a node but not reach the root.

Hierarchy of Name Servers

- To distribute the information among many computers, DNS servers are used. Creates many domains as there are first level nodes. Fig. 3.3.3 shows hierarchy of name servers.

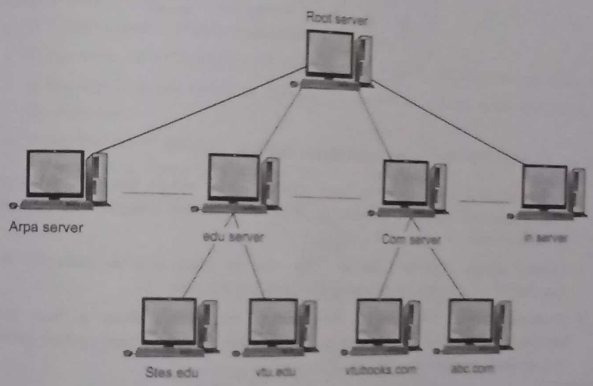


Fig. 3.3.3 Hierarchy of name server

- 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.
- Fig. 3.3.4 shows zones and domains.

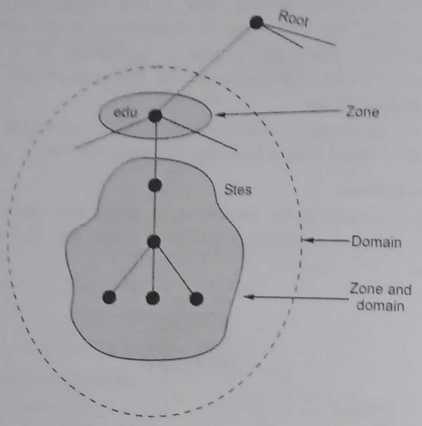


Fig. 3.3.4 Domain and zone

- A root sever is a server whose zone consists of the whole tree. A root server usually does not store any information about domains but delegates its authority to other servers.
- **Primary server** : It stores a file about the zone for which it is an authority. It is responsible for creating, maintaining and updating the zone file.
- **Secondary server** : It transfers the complete information about a zone from another server and stores the files on its local disk. These servers neither creates nor updates the zone files.

3.3.2 Name Resolution

- 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**.

Working :

- Name resolving must also include the type of answer desired (specifying the protocol family is optional).
- 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.

Mapping Domain Names to Addresses

a) The DNS also includes an efficient, reliable, general purpose, distributed system for mapping names to addresses using an independent cooperative system called name servers.

b) Names Servers - are server programs that translate names-to-addresses (maps DN => IP addresses) and usually executes on a dedicated processor.

c) Name Resolvers - Client software that uses one or more name servers in getting a mapped name.

d) Domain name servers are arranged in a conceptual tree structure that corresponds to the naming hierarchy.

Recursive Resolution

- A client request complete translation.
- If the server is authority for the domain name, it checks its database and responds.
- If the server is not authority, it sends the request to another server and waits for the response.
- When the query is finally resolved, the response travel back until it finally reaches the requesting client. This is called recursive resolution.
- Fig. 3.3.5 shows the recursive resolution.

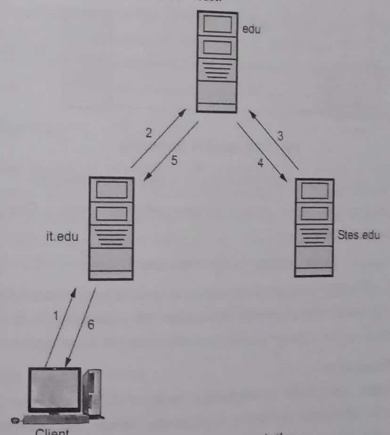


Fig. 3.3.5 Recursive resolution

Iterative Resolution

- Only a single resolution is made and returned (not recursive).
- Client must now explicitly contact different name servers if further resolution is needed.
- If the server is an authority for the name, it sends the answer. If it is not, it returns the IP address of the server that is thinks can resolve the query. The client is responsible for repeating the query to this second server. This process is called iterative resolution because the client repeats the same query to multiple servers.
- Fig. 3.3.6 shows iterative resolution.

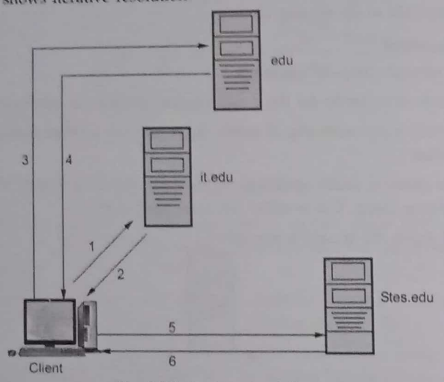


Fig. 3.3.6 Iterative resolution

- Conceptually, name resolution proceeds in a top-down fashion.
- Name resolution can occur in one of two different ways : Recursive resolution and Iterative resolution.
- Name servers use name caching to optimize search costs.
- Time To Live (TTL) is used to determine a guaranteed name binding during it's time interval. When time expires, the cache name binding is no longer valid, so the client must make a direct name resolution request once again.

Reverse Name Resolution

- Reverse name resolution is important task of DNS on the Internet or the translation of IP addresses back to domain names. For example, servers can determine and record the full domain name of machine connecting to them over the network.

- It is not efficient to use the same set of DNS records for reverse name resolution. Instead, a separate domain called "IN-ADDR.ARPA" has been set aside to provide a hierarchy for translating IP addresses into names.
- A DNS lookup of "barg.oo.msstate.edu" would reveal it has the IP address "130.19.60.10". If one has the IP address and wishes to know the name, one must perform a DNS lookup of "10.60.19.130. in -addr.arpa", which will return the name.
- Reverse name resolution fields use the PTR resource record, which points to the correct position in the normal DNS space. The hierarchy under "IN-ADDR.ARPA" can be delegated of course just like any other domain.
- To obtain the IP address of a named server, each host has a client protocol known as the name resolver. On receipt of the name, the client application protocol posses it to the name resolver using the standard interprocess communication primitive supported by the local operating system.
- The resolver then creates a resolution request message in the standard message format of the domain name server protocol.
- A resolver can have multiple request outstanding at any time. Hence the identification field is used to relate a subsequent response message to an earlier request message.
- The name resolver passes the request message to its local domain name server using TCP/IP. If the request is for a server on this network, the local domain name server obtains the corresponding IP address from its DIB and returns it in a reply message.

3.3.3 Name Servers

- To avoid the problems associated with having only a single source of information, the DNS name space is divided into nonoverlapping **zones**. When a resolver has a query about a domain name, it passes the query to one of the local name servers. If the domain being sought falls under the jurisdiction of the name server, it returns the authoritative resource records.
- An **authoritative record** is one that comes from the authority that manages the record and is thus always correct. Authoritative records are in contrast to cached records, which may be out of data.

Resolver looks up a remote name :

- If the domain is remote and no information about the requested domain is available locally, the name server sends a query message to the top level name server for the domain requested.

- Consider the following example of Fig. 3.3.7. A resolver on flits.cs.vu.nl wants to know the IP address of the host india.cs.stes.edu.

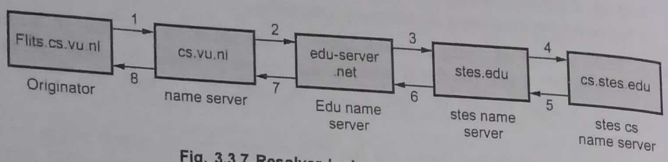


Fig. 3.3.7 Resolver looks up a remote name

Step 1 : It sends a query to the local name server, cs.vu.ne. This query contains the domain name sought, the type (A) and the class (IN).

Step 2 : The local name server has never had a query for this domain before and knows nothing about it. It may ask a few other nearby name servers, but if none of them know, it sends a UDP packet to the server for edu given in its database, edu-server.net.

Step 3 : It is unlikely that this server knows the address of india.cs.stes.edu and probably does not know cs.stes.edu either, but it must know all of its own children, so it forwards the request to the name server for stes.edu.

Step 4 : In turn, this one forwards the request to cs.stes.edu, which must have the authoritative resource records.

Step 5 - 8 : Each request is from a client to a server, the resource record requested works its way back.

- Once these records get back to the cs.vu.nl name server, they will be entered into a cache there, in case they are needed later.

3.3.4 Resource Records

- Different types of resource records are used in DNS. An IP address has a type of A and PTR means pointer query.
- There are about 20 different types of resource records available. Some PR are listed below.
  1) A = It defines an IP address. It is stored as a 32-bit binary value.
  2) CNAME = "Canonical name". It is represented as a domain name.
  3) HINFO = Host information, two arbitrary character strings specifying the CPU and Operating System (OS).
  4) MX = Mail exchange records. It provide domain willing to accept e-mail.

5) PTR = Pointer record used for pointer queries. The IP address is represented as a domain name in the in-addr.arpa domain.
6) NS = Name Server record. These specify the authoritative name server for a domain. They are represented as domain names.

Configuration of DNS :

- The DNS server can be configured manually by editing files in the default WINNT installation path \% SYSTEM ROOT %\ SYSTEM 32 \ DNS. Administration is identical to administration in traditional DNS. These files can be modified using a text editor. The DNS service must then be stopped and restarted.

3.4 Attribute-based Naming

- Attribute-based naming systems are also known as directory services.

3.4.1 Directory Services

- A **directory service** is a software application or a set of applications that stores and organizes information about a computer network's users and network resources, and that allows network administrators to manage users' access to those resources.
- Sometimes users require a service, but they are not concerned with that system entity supplies that service. Attributes may be used as values to be looked up.
- **Directory service** is a service that stores collections of bindings between names and attributes and that looks up entries that match attribute-based specifications. Sometimes called yellow pages services or attribute-based name services. A directory service returns the sets of attributes of any objects found to match some specified attributes.

Discovery Services

- Directory service that registers services provided in a spontaneous networking environment. It Provide an interface for automatically registering and de-registering services, as well as an interface for clients to look up the services they require.
- Directory service is automatically updated as the network configuration changes and meets the needs of clients in spontaneous networks. It also discovers services required by a client (who may be mobile) within the current scope, for example, to find the most suitable printing service for image files after arriving at a hotel.
- Examples of discovery services : Jini discovery service, the 'service location protocol', the 'simple service discovery protocol', the 'secure discovery service'.

- Example of Discovery service : A printer may register its attributes with the discovery service as follows :

'resourceClass = printer, type=laser, color=yes, resolution=600dpi, location=room101, url=http://www.collegeNW.com/services/laserprinter'

3.4.2 LDAP

- LDAP stands for Lightweight Directory Access Protocol. LDAP defines a standard method for accessing and updating information in a directory. It has gained wide acceptance as the directory access method of the Internet and is therefore also becoming strategic within corporate intranets.
- LDAP is based on X.500. It is a fast growing technology for accessing common directory information. Fig. 3.4.1 shows LDAP uses X.500.

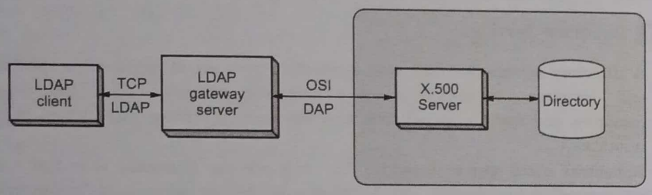


Fig. 3.4.1 LDAP uses X.500

- **Why use LDAP ?**
  1. Centrally manage users, groups and other data.
  2. Don't have to manage separate directories for each application.
  3. Distribute management of data to appropriate people.
  4. Allow users to find data that they need.
  5. Not locked into a particular server.
  6. Ability to distribute servers to where they are needed.
- A directory is a listing of information about objects arranged in some order that gives details about each object. Common examples are a city telephone directory and a library card catalog. In computer terms, a directory is a specialized database, also called a data repository that stores typed and ordered information about objects.
- A directory is often described as a database. But it has special characteristics different from general databases :
  1. They are accessed much more than they are updated. Hence they are optimized for read access.

  2. They are not suited for information that changes rapidly (e.g. number of jobs in a printer queue).
  3. Many directory services don't support transactions.
  4. Directories normally limits the type of information that can be stored.
  5. Databases use powerful query languages like SQL but Directories normally use very simple access methods.
  6. Hence directories can be optimized to economically provide more applications with rapid access.
- LDAP is well suited for
  1. Information that is referenced by many entities and applications.
  2. Information that needs to be accessed from more than one location.
  3. Information that is read more often than it is written.
- LDAP is not well suited for
  1. Information that changes often.
  2. Information that is unstructured.
- Directories are usually accessed using the client/server model of communication. LDAP defines a message protocol used by directory clients and directory servers but does not define a programming interface for the client.
- X.500 organizes directory entries in a hierarchal name space capable of supporting large amounts of information. It also defines powerful search capabilities to make retrieving information easier. Because of its functionality and scalability.
- X.500 is often used together with add-on modules for interoperation between incompatible directory services. It specifies that communication between the directory client and the directory server uses the directory access protocol.
- LDAP defines a communication protocol. Every directory needs a namespace. The LDAP namespace is the system used to reference objects in an LDAP directory. Each object must have a name.
- Namespace hierarchy allows management control. DNS is by definition hierarchical in nature. The LDAP name-space is hierarchical too. LDAP uses strings to represent data rather than complicated structured syntaxes such as ASN.1
- LDAP defines a set of server operations used to manipulate the data stored by the directory. LDAP uses TCP/IP for its communications. For a client to be able to connect to an LDAP directory, it must open a TCP/IP session with the LDAP server.
- LDAP minimizes the overhead to establish a session allowing multiple operations from the same client session. LDAP defines operations for accessing and modifying directory entries such as :

1. Searching for entries meeting user-specified criteria.
2. Adding an entry.
3. Deleting an entry.
4. Modifying an entry.
5. Modifying the distinguished name or relative distinguished name of an entry (move).
6. Comparing an entry.

3.5 Fill in the Blanks

Q.1 _____ space is a collection of names which may or may not share an identical resolution mechanism.

Q.2 Each node n maintains a routing table with up to m entries called _____ .

Q.3 Chord is a protocol and algorithm for a _____ distributed hash table.

Q.4 Attribute-based naming systems are also known as _____ .

Q.5 Name spaces are of two types : _____ spaces and _____ names.

Q.6 Name resolution refers to the process of mapping a name to an _____ .

Q.7 The collection of all directory in an LDAP directory service is called a _____ .

Q.8 A directory node stores a table in which an outgoing edge is represented as a pair _____. Such a table is called a _____ .

Q.9 The DNS name space is _____ organized as a rooted tree.

Q.10 Flat and structured names generally provide a unique and _____ way of referring to entities.

3.6 Multiple Choice Questions

Q.1 Types of name resolution mechanisms for flat names are _____

a broadcasting
b home-based approaches
c distributed Hash Tables (DHTs)
d all of these

Q.2 LDAP stands for _____

a Lightweight Distributed Access Protocol
b Lightweight Directory Access Protocol
c Lightweight Down Access Protocol
d Lightweight Direct Access Protocol

Q.3 Chord is based on _____ hashing.

a direct
b distributed
c non-consistent
d consistent

Q.4 A lowest-level domain, called a leaf domain, typically corresponds to a local-area network in a computer network or a cell in a mobile telephone network.

a Top domain
b Bottom domain
c Leaf domain
d Root domain

Answer Keys for Fill in the Blanks

Q.1	Name	Q.2	finger table
Q.3	peer-to-peer	Q.4	directory services
Q.5	Flat name, Hierarchical	Q.6	object
Q.7	directory information base	Q.8	edge label, node identifier, directory table
Q.9	hierarchically	Q.10	location-independent

Answer Keys for Multiple Choice Questions

Q.1	d	Q.2	b
Q.3	d	Q.4	c

❑❑❑

4 Synchronization

Syllabus

Clock Synchronization, Logical Clocks, Mutual Exclusion, Global Positioning of Nodes, Election Algorithms.

Contents

4.1 Clock Synchronization

- There is no common universal time but the speed of light is constant for all observers irrespective of their velocity.
- Timers in computers are based on frequency of oscillation of a quartz crystal. Each computer has a timer that interrupts periodically.
- Time is also an important theoretical construct in understanding how distributed executions unfold. But time is problematic in distributed systems. Each computer may have its own physical clock, but the clocks typically deviate, and we cannot synchronize them perfectly.

Needs for precision time :

a. Stock market buy and sell orders stock market buy and sell orders

b. Secure document timestamps

c. Distributed network gaming and training

d. Aviation traffic control and position reporting

e. Multimedia synchronization for real-time teleconferencing

f. Event synchronization and ordering

g. Network monitoring measurement and control

- **Each computer in a DS has its own internal clock**
  1. used by local processes to obtain the value of the current time
  2. processes on different computers can timestamp their events
  3. but clocks on different computers may give different times
  4. computer clocks drift from perfect time and their drift rates differ from one another

4.1.1 Absence of a Global Clock

- Due to asynchronous message passing there are limits on the precision with which processes in a distributed system can synchronize their clocks.
- In a distributed system there are as many clocks as there are systems. The clocks are coordinated to keep them somewhat consistent but no one clock has the exact time.
- Even if the clocks were somewhat in sync, the individual clocks on each component may run at a different rate or granularity leading to them being out of sync only after one local clock cycle. Time is only known within a given precision. At frequent intervals, a clock may synchronize with a more trusted clock. However, the clocks are not precisely the same because of time lapses due to transmission and execution.

- Consider a group of people going to a meeting. Each person has a watch. Each watch has a similar, but different time. Even with the error in time, the group is able to meet and conduct business. This is how distributed time works. It is difficult to make temporal order of events and difficult to collect up-to-date information on the state of the entire system.
- Algorithm for designing and debugging of distributed system is more difficult than centralized systems.

4.1.2 Clock and Event

- How to order the events that occur at a single processor ?
- A distributed system is defined as a collection P of N processes p_i, i = 1, 2,... N. Each process executes on a single processor and the processors do not share memory. Each process p_i has a state s_i consisting of its variables.
- Processes communicate only by messages (via a network). We can view each process p_i as to execute a sequence of actions that fall in one of the following categories :
  1. Sending a message;
  2. Receiving a message;
  3. Performing a computation that alters its state s_i.
- We define an event to be the execution of a single action by p_i. Event is the occurrence of a single action that a process carries out as it executes. For example : Send, Receive, change state. The sequence of events within a single process p_i can be totally ordered, which is generally denoted by $e_i \rightarrow e'$ if and only if the event e occurs before e' at p_i.
- We can then define the history of process pi to be the sequence of events that take place within :

$$\text{history } (p_i) = h_i = < e_i^0, e_i^1, e_i^2, \ldots >$$

4.1.3 Physical Clock

- Computer physical clocks are electronic devices that count oscillations occurring in a crystal at a definite frequency. This count can be stored in a counter register. The clock output can be read by software and scaled into a suitable time unit. This value can be used to timestamp any event experienced.
- Most computers today keep track of the passage of time with a battery-backed-up CMOS clock circuit, driven by a quartz resonator. This allows the time keeping to take place even if the machine is powered off. When on, an operating system will

generally program a timer circuit to generate an interrupt periodically. The interrupt service procedure simply adds one to a counter in memory.

- While the best quartz resonators can achieve an accuracy of one second in 10 years, they are sensitive to changes in temperature and acceleration and their resonating frequency can change as they age.
- The only problem with maintaining a concept of time is when multiple entities attempt to do it concurrently. Two watches hardly ever agree. Computers have the same problem: a quartz crystal on one computer will oscillate at a slightly different frequency than on another computer, causing the clocks to tick at different rates.
- The phenomenon of clocks ticking at different rates, creating a ever widening gap in perceived time is known as clock drift. The difference between two clocks at any point in time is called clock skew and is due to both clock drift and the possibility that the clocks may have been set differently on different machines.
- Fig. 4.1.1 shows skew with two clocks.

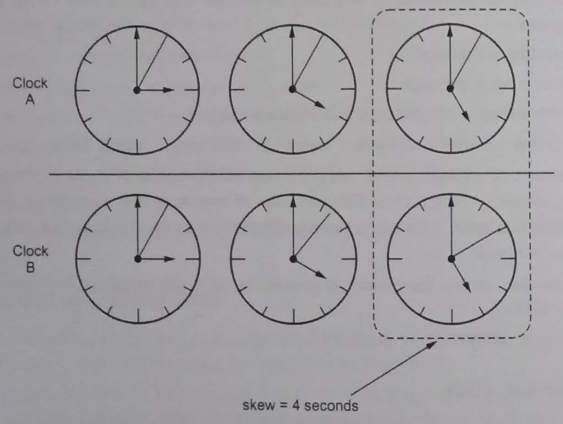


Fig. 4.1.1 Clock drift and clock skew

- Consider two clocks A and B, where clock B runs slightly faster than clock A by approximately two seconds per hour. This is the clock drift of B relative to A. At one point in time, the difference in time between the two clocks is approximately 4 seconds. This is the clock skew at that particular time.

- Successive events will correspond to different timestamps only if the clock resolution is smaller than the rate at which events can occur. The rate at which events occur depends on such factors as the length of the processor instruction cycle.
- Applications running at a given computer require only the value of the counter to timestamp events. The date and time-of-day can be calculated from the counter value. Clock drift may happen when computer clocks count time at different rates.
- Co-ordinated Universal Time (UTC) is an international standard that is based on atomic time. UTC signals are synchronized and broadcast regularly from land-based radio stations and satellites.
- If the computer clock is behind the time service's, it is OK to set the computer clock to be the time service's time. However, when the computer clock runs faster, then it should be slowed down for a period instead of set back to the time service's time directly.
- The way to cause computer's clock run to slow for a period can be achieved in software without changing the rate of the hardware clock. Also called timer, usually a quartz crystal, oscillating at a well-defined frequency.
- A timer is associated with two registers: a counter and a holding register, and counter decreasing one at each oscillation. When the counter gets to zero, an interruption is generated and is called **one clock tick.**
- Crystals run at slightly different rates, the difference in time value is called a clock skew. Clock skew causes time-related failures. Fig. 4.1.2 shows working of computer clock.

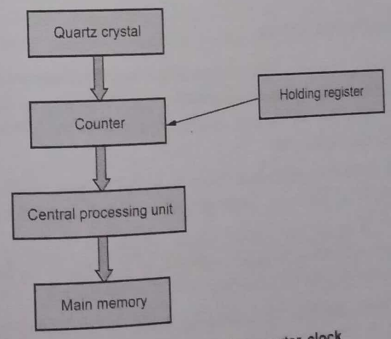


Fig. 4.1.2 Working of computer clock

Working :

1. Oscillation at a well-defined frequency
2. Each crystal oscillation decrements the counter by 1
3. When counter gets 0, its value reloaded from the holding register
4. When counter is 0, an interrupt is generated, which is call a clock tick
5. At each clock tick, an interrupt service procedure add 1 to time stored in memory

- **Synchronization of physical clocks with real-world clock :**
  1. TAI (International Atomic Time) : Cs133 atomic clock
  2. UTC (Universal Co-ordinated Time) : Modern civil time, can be received from WWV (shortwave radio station), satellite, or network time server.
  3. ITS (Internet Time Service) NTS (Network Time Protocol)
- **Some definitions :**
  1. **Transit of the sun :** The event of the sun's reaching its highest apparent point in the sky.
  2. **Solar day :** The interval between two consecutive transits of the sun is called the solar day.
  3. **Coordinated Universal Time (UTC) :** The most accurate physical clocks known use atomic oscillators, whose accuracy is about one part in 10^{13}. The output of these atomic clocks is used as the standard for elapsed real time, known as International Atomic Time. Co-ordinated universal time is an international standard that is based on atomic time, but a so-called leap second is occasionally inserted or deleted to keep in step with astronomical time.

4.1.4 Global Positioning System

- Global Positioning System (GPS) is a satellite - based navigation system made up of a network of satellites placed into orbit by the US Department of Defence. GPS was originally intended for military applications, but in the 1980s the system was made available for civilian use.
- GPS satellites circle the earth twice a day in an MEO orbit. GPS uses 29 satellites each circulating in an orbit at a height of approximately 20,000 km.
- Each satellite has up to four atomic clocks, which are regularly calibrated from special stations on Earth. A satellite continuously broadcasts its position, and time stamps each message with its local time.
- This broadcasting allows every receiver on Earth to accurately compute its own position using, in principle, only three satellites.

- GPS satellites broadcast a data stream at the primary frequency L1 of 1.57 GHz, which carries the coarse - acquisition (C/A) encoded signal to be captured by the receiver's antenna. The GPS receiver measures the time of arrival of the C/A code to a fraction of a millisecond.
- Fig. 4.1.3 shows computing a position in a two - dimensional space.

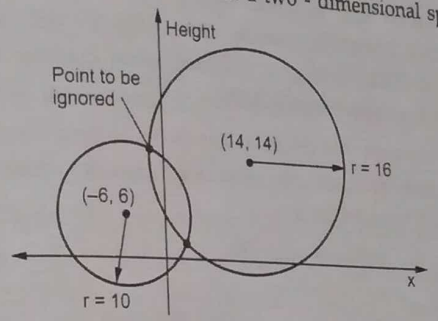


Fig. 4.1.3 Computing a position in a two - dimensional space

- A satellite continuously broadcasts its position, and time stamps each message with its local time. This broadcasting allows every receiver on Earth to accurately compute its own position using, in principle, only three satellites.
- In order to compute a position, consider first the two - dimensional case, in which two satellites are drawn, along with the circles representing points at the same distance from each respective satellite.
- The y-axis represents the height, while the x-axis represents a straight line along the Earth's surface at sea level. the intersection of the two circles is a unique point. Because the GPS receiver does not carry atomic clocks, the measured distances between the receiver and GPS satellites introduce errors originating from the clock error, and the distance is called the pseudo range.
- The real distance is simply computed as :

$$R_i = \sqrt{(X_{si} - X)^2 + (Y_{si} - Y)^2 + (Z_{si} - Z)^2}$$

where

R_i is the real distance between the i^{th} satellite to the receiver P;

C is the speed of light;

Δt_{Ai} is the i^{th} satellite's transmission delay and other errors;

Δt_u is the receiver clock's errors relative to GPS system time;

Δt_{si} is the i^{th} satellite's errors relative to GPS system time.

- Assuming the position of the satellite Si and the receiver P under the geocentric rectangular coordinate system is (X_{Si}, Y_{Si}, Z_{Si}) and (X, Y, Z), respectively.

4.2 Clock Synchronization Algorithms

- A quartz crystal on one computer will oscillate at a slightly different frequency than on another computer, causing the clocks to tick at different rates. The phenomenon of clocks ticking at different rates, creating ever widening gap in perceived time is known as clock drift.
- The difference between two clocks at any point in time is called clock skew and is due to both clock drift and the possibility that the clocks may have been set differently on different machines.
- Fig. 4.2.1 shows the drift rate of clocks

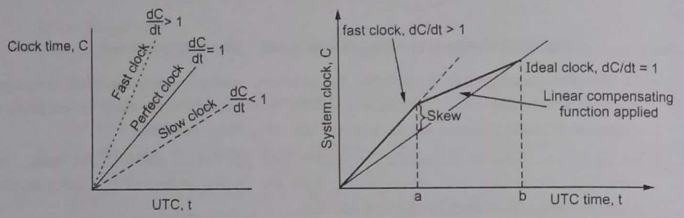


Fig. 4.2.1 Drift rate of clocks

- If a clock is fast, it simply has to be made to run slower until it synchronizes. If a clock is slow, the same method can be applied and the clock can be made to run faster until it synchronizes.
- The operating system can do this by changing the rate at which it requests interrupts. For example, suppose the system requests an interrupt every 17 milliseconds and the clock run a bit too slowly. The system can request interrupts at a faster rate, say every 16 or 15 milliseconds, until the clock catches up. This adjustment changes the slope of the system time and is known as a linear compensating function.

4.2.1 Network Time Protocol

- Cristian's method and the Berkeley algorithm are intended for intranets.
- The Network Time Protocol (NTP) is yet another method for synchronizing clocks that uses a hierarchical architecture where the top level of the hierarchy are servers connected to a UTC time source such as a GPS unit.

4.2.1.1 Localized Averaging Distributed Algorithms

- Each node exchanges its clock time with its neighbors.
- Then sets its clock time to the average of its own clock and the clock times of its neighbors

Network time protocol

Goals :

1. Enable clients across the Internet to be accurately synchronized to UTC despite message delays.
2. Provide a reliable service that can survive lengthy losses of connectivity.
3. Enable clients to synchronize frequently and offset the effects of clock drift.
4. Provide protection against interference; authenticate that the data is from a trusted source.

- The NTP servers are connected in a logical hierarchy, where servers in level n are synchronized directly to those in level n – 1. The logical hierarchy can be reconfigured as servers become unreachable or failed.
- NTP servers synchronize with one another in one of three modes in the order of increasing accuracy :
  1. Multicast on high speed local LANs
  2. Procedure call mode
  3. Symmetric mode

Features of NTP :

1. To provide a service enabling clients across the Internet to be synchronized accurately to UTC; despite the large and variable message delays encountered in Internet communication.
2. To provide a reliable service that can survive lengthy losses of connectivity.
3. To enable clients to resynchronize sufficiently; to offset the rate of drift found in most computers.
4. To provide protection against interference with the time service; whether malicious or accidental. The time service uses authentication techniques to check that timing data originate from the claimed trusted sources. It also validates the return addresses of messages sent to it.

- NTP servers synchronize modes are as follows :
  1. Multicast
  2. Procedure-call
  3. Symmetric mode.

1. **Multicast mode :**
   - Multicast mode is intended for use on a high - speed LAN.
   - One or more servers periodically multicasts the time to the servers running in other computers connected by the LAN, which set their clocks assuming a small delay.
   - This mode can only achieve relatively low accurate, but ones which nonetheless are considered sufficient for many purposes.
2. **Procedure - call mode :**
   - Procedure call mode is similar to the operation of Cristian's algorithm.
   - In this mode, one server accepts requests from other computers, which it processes by replying with its timestamp.
   - This mode is suitable where higher accuracies are required that can be achieved with multicast or where multicast is not supported in hardware.
   - For example, file servers on the same or a neighboring LAN , which need to keep accurate timing information for file accesses, could contact a local master server in procedure - call mode.
3. **Symmetric mode :**
   - Symmetric mode is intended for use by the master servers that supply time information in LANs and by the higher levels of the synchronization subnet, where the highest accuracies are to be achieved.
   - A pair of servers operating in symmetric mode exchange messages bearing timing information.
   - Timing data are retained as part of an association between the servers that is maintained in order to improve the accuracy of their synchronization over time.
   - In all modes, messages are delivered unreliably, using the standard UDP Internet transport protocol. In procedure-call mode and symmetric mode, messages are exchanged in pairs.

4.2.2 Berkeley Algorithm

- Time server is a active machine.
- The server polls each machine periodically, asking it for the time. The time at each machine may be estimated by using Cristian's method to account for network delays.
- When all the results are in, the master computes the average time (including its own time in the calculation).

- Instead of sending the updated time back to the slaves, which would introduce further uncertainty due to network delays, it sends each machine the offset by which its clock needs adjustment. The operation of this algorithm is shown in Fig. 4.2.2.

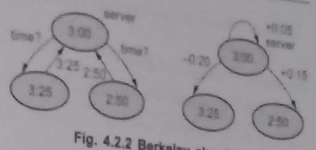


Fig. 4.2.2 Berkeley algorithm

- Three machines have times of 3 : 00, 3 : 25, and 2 : 50. The machine with the time of 3 : 00 is the server. It sends out a synchronization query to the other machines in the group.
- Each of these machines sends a timestamp as a response to the query. The server now averages the three timestamps : the two it received and its own, computing
  $$= (3 : 00 + 3 : 25 + 2 : 50)/3 = 3 : 05$$
- Now it sends an offset to each machine so that the machine's time will be synchronized to the average once the offset is applied. The machine with a time of 3 : 25 gets sent an offset of - 0 : 20 and the machine with a time of 2 : 50 gets an offset of + 0 : 15. The server has to adjust its own time by + 0 : 05.
- The algorithm also has provisions to ignore readings from clocks whose skew is too great. The master may compute a fault-tolerant average i.e. averaging values from machines whose clocks have not drifted by more than a certain amount. If the master machine fails, any other slave could be elected to take over.

4.3 Logical Clocks

- For a certain class of algorithms, it is the internal consistency of the clocks that matters. The convention in these algorithms is to speak of logical clocks.
- Lamport showed clock synchronization need not be absolute. What is important is that all processes agree on the order in which events occur.
- A logical clock Cp of a process p is a software counter that is used to timestamp events executed by p so that the happened-before relation is respected by the timestamps.

4.3.1 Event Ordering

- A person with one watch knows what time it is. But a person with two or more watches is never sure.
- Lamport defined a relation called **happens before,** represented by $\rightarrow$.

- The relation → on a set of events of a system is the relation satisfying three conditions :

Conditions of happens before

1. If **a** and **b** are events in the same process, and **a** comes before **b**, then a → b.
2. If **a** is the sending event of a message **msg** by one process, and **b** is the receipt event of **msg**, then
   a → b.
3. If **a** → **b**, **b**→ **c**, then **a** → **c**.

- The order of events occurring at different processes can be critical in a distributed application.
- To determine the order of the events that occur at different processes in a distributed application, two obvious points are :
  - If two events occurred at the same process, then they occurred in the order in which it observes them.
  - Whenever a message is sent between processes, the event of sending the message occurred before the event of receiving the message.
- The ordering obtained by generalizing these two relationships is called **happen-before** relation.
- Lamport developed a "happens before" notation to express the relationship between events : a → b means that a happens before b.
- If a represents the timestamp of a message sent and b is the timestamp of that message being received, then a → b must be true; a message cannot be received before it is sent. This relationship is transitive.
- If a → b and b → c then a → c. If a and b are events that take place in the same process the a → b is true if a occurs before b.
- The importance of measuring logical time is in assigning a time value to each event such that everyone will agree on the final order of events. That is, if a → b then clock(a) < clock(b) since the clock must never run backwards.
- If a and b occur on different processes that do not exchange messages then a→b is not true. These events are said to be concurrent : there is no way that a could have influenced b.
- We write x →$_p$ y if two events x and y occurred at a single process p and x occurred before y. Using this restricted order we can define the happened-before relation, denoted by → , as follows :
  1. HB1 : If ∃ process p : x →$_p$ y, then x → y.
  2. HB2 : For any message m, send(m) → rcv(m),

- where send(m) is the event of sending the message, and rcv(m) is the event of receiving it.
  3. HB3 : If x, y and z are events such that x → y and y → z, then x→ z.
- If x → y, then we can find a series of events occurring at one or more processes such that either HB1 or HB2 applies between them. The sequence of events need not be unique.
- If two events are not related by the → relation (i.e., neither a→ b nor b→a), then they are concurrent
  (a || b).
- a → b → c → d f ; e → f but a || e.
- **Example : Event ordering**

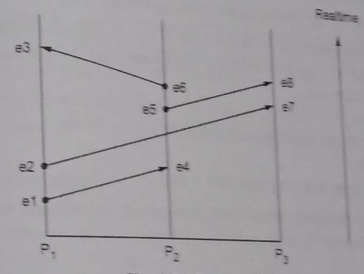


Fig. 4.3.1 Event ordering

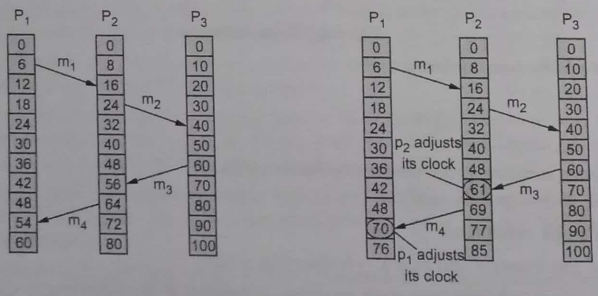
- **Possible event ordering :**
  1. e1 → e2 → e3
  2. e1 → e4 → e5 → e6 → e3
  3. e2 → e7 → e8
- Event ordering is not possible in this condition :
  e2 → e4 and e5 → e7

Logical clock condition :

1. For any events a and b, if a → b then C(a) < C(b)
2. From the definition, the clock condition is satisfied if the following two conditions hold :
   i. Condition 1 : If a and b are events in P_i and a comes before b, then $C_i(a) < C_i(b)$.
   ii. Condition 2 : If a is the sending of a msg by P_i and b is the receipt of the msg by P_j, then $C_i(a) < C_j(b)$.

4.3.2 Lamport Timestamp

- Lamport algorithm is used to synchronize the logical clock. It uses happens-before relationship to provide a partial ordering of events :
  1. All processes use a counter (clock) with initial value of zero.
  2. The counter is incremented by and assigned to each event, as its timestamp.
  3. A send (message) event carries its timestamp.
  4. For a receive (message) event the counter is updated by Max (receiver – counter, message – timestamp) + 1
- Goal of the lamport algorithm is to assign totally ordered timestamps to events.
- **Requirements**
  a. if $a \rightarrow b$ then $T(a) < T(b)$
  b. $T(a) \neq T(b)$ for any two different events a and b
- Happens-before relation is a transitive, so if $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$. if two events , x and y happen in different processes that do not exchange messages, then $x \rightarrow y$ is not true.
- **Algorithm**
  a. if $C_i(a) \neq C_j(b)$, then $T(a) = C_i(a)$ and $T(b) = C_i(b)$
  b. if $C_i(a) = C_j(b)$ and $i < j$, then $T(a) < T(b)$
- Consider three processes where each process runs on different machines with its own clock and speed. It is shown in Fig. 4.3.2.



(a) Process with its own clock runs at different rate

(b) Lamport's algorithm

Fig. 4.3.2 Clock timestamp

- The processes run on different machines, each with its own clock and running with own speed.
- When the clock has ticked 6 times in process P_1 , it has ticked 8 times in process P_2 and 10 times in process P_3. Each clock runs at a constant rate, but rate varies according to the crystals.
- At time 6, process P_1 sends message m_1 to process P_2 . The clock in process 2 reads 16 when it arrives. Process 2 will conclude that it took 10 ticks to reach from process 1 to process 2.
- According to this reasoning, message m_2 from process 2 to process 3 takes 16 ticks
- In Fig. 4.3.2 (a) , message m_3 from process 3 to process 2 leaves at 60 and arrives at 56. Similarly, message m_4 from process 2 to process 2 leave at 64 and arrive at 54. These values are not possible.
- Lamport solution is given in Fig. 4.3.2 (b) which uses happen-before relation. Since message m_3 left at 60, it must arrive at 61 or later. So each message carry its sending time according to the sender's clock.
- When message arrives and the receiver's clock shows a value prior to the time the message was sent, the receiver fast forwards its clock to be one more than sending time.

Totally ordered multicasting :

- We can use the logical clocks satisfying the clock condition to place a total ordering on the set of all system events. Simply order the events by the times at which occur.
- To break the ties, lamport proposed the use of any arbitrary total ordering of the processes, i.e. process id
- Using this method, we can assign a unique timestamp to each event in a distributed system to provide a total ordering of all events. Very useful in distributed system for solving the mutual exclusion problem
- We sometimes need to guarantee that concurrent updates on a replicated database are seen in the same order everywhere :
- P_1 adds ₹ 100 to an account (initial value : ₹ 1000)
- P_2 increments account by 1 %
- There are two replicas. Fig. 4.3.3 shows the updating a replicated database and leaving it in an inconsistent state.

Result : In absence of proper synchronization :

replica#1 ₹ : 1111, while replica #2 ₹ : 1110.

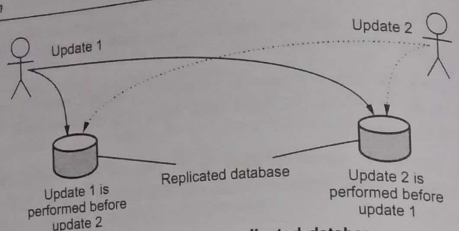


Fig. 4.3.3 Updating replicated database

- Examples for Lamport's timestamp

Initial condition : C(P) = 0, C(Q) = 2, C(R) = 0

Processor ids : pid(P) = 0, pid(Q) = 1, pid(R) = 2

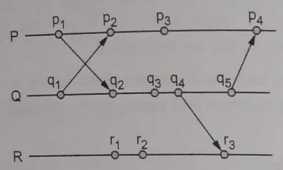


Fig. 4.3.4

Partially ordered Lamport's timestamps :

1. $p_1 = 1,\ p_2 = 4,\ p_3 = 5,\ p_4 = 8$
2. $q_1 = 3,\ q_2 = 4,\ q_3 = 5,\ q_4 = 6,\ q_5 = 7$
3. $r_1 = 1,\ r_2 = 2,\ r_3 = 7$

Totally ordered Lamport's timestamps :

1. $p_1 = 10,\ p_2 = 40,\ p_3 = 50,\ p_4 = 80$
2. $q_1 = 31,\ q_2 = 41,\ q_3 = 51,\ q_4 = 61,\ q_5 = 71$
3. $r_1 = 12,\ r_2 = 22,\ r_3 = 72$

Limitation of Lamport's logical clock

- If $a \rightarrow b$ then $C(a) < C(b)$, but if $C(a) < C(b)$ then $a \rightarrow b$ is not necessary true, i.e. concurrency information is lost

4.3.3 Vector Timestamp

- With Lamport's clocks, one cannot directly compare the timestamps of two events to determine their precedence relationship.

- The main problem is that a simple integer clock cannot order both events within a process and events in different processes. Collin Fidge developed an algorithm that overcomes this problem.
- A vector clock system is a mechanism that associates timestamps with events (local states) such that comparing two events' timestamps indicates whether those events (local states) are causally related.
- Fidge's clock is represented as a **vector** $[C_1, C_2, \ldots, C_n]$ with an integer clock value for each process (C_i contains the clock value of process i)
- Each process P_i has an array VC_i [1..n], where VC_i [j] denotes the number of events that process P_i knows have taken place at process P_j.
- When P_i sends a message m, it adds 1 to VC_i [i], and sends VC_i along with m as vector timestamp vt(m). Result : upon arrival, recipient knows P_i s timestamp.
- When a process P_j delivers a message m that it received from P_i with vector timestamp $t_s(m)$, it updates each VC_j [k] to max{VC_j [k], $t_s(m)$[k]} and increments VC_j [j] by 1.
- We can now ensure that a message is delivered only if all causally preceding messages have already been delivered.

Adjustment

- P_i increments VC_i [i] only when sending a message, and P_j "adjusts" VC_j when receiving a message (i.e., effectively does not change VC_j [j]).
- In the time-stamping system, each process P_i has a vector of integers $VC_i[1..n]$ (initialized to [0,...,0]) that is maintained as follows :
- Each time process P_i produces an event (send, receive, or internal), it increments its vector clock entry $VC_i[i]$ ($VC_i[i] := VC_i[i]+1$) to indicate that it has progressed.
- When a process P_i sends a message m, it attaches to it the current value of VC_i. Let m.VC denote this value.
- When P_i receives a message m, it updates its vector clock as
- Note that $VC_i[i]$ counts the number of events that P_i has so far produced.

$$\forall x: VC_i[x] := \max(VC_i[x], m.VC[x]) \text{ (abbreviated as } VC_i := \max(VC_i, m.VC))$$

- $VC_i[j]$ represents the number of events P_j produced that belong to the current causal past of P_i. When a process P_i produces an event e, it can associate with that event a vector timestamp whose value equals the current value of VC_i.

Example : Assign the Lamport's clock values for all the events in the above timing diagram. Assume that each process's logical clock is set to 0 initially.

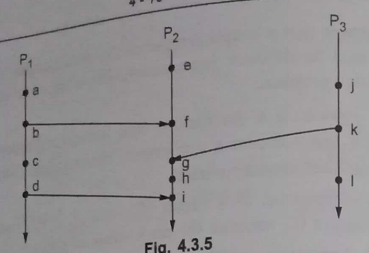


Fig. 4.3.5

Solution : Lamport clocks :

$2 < 5$ since $b \rightarrow h$,

$3 < 4$ but $c \rightarrow g$.

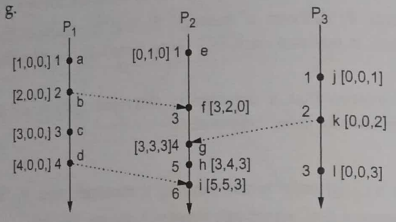


Fig. 4.3.6

4.4 Mutual Exclusion

- Mutual exclusion ensures that concurrent processes make a serialized access to shared resources or data. It requires that the actions performed by a user on a shared resource must be atomic.
- In a distributed system neither shared variables nor a local kernel can be used in order to implement mutual exclusion. Thus, mutual exclusion has to be based exclusively on message passing, in the context of unpredictable message delays and no complete knowledge of the state of the system.
- **Mutual exclusion :** Makes sure that concurrent process access shared resources or data in a serialized way. If a process, say P_i , is executing in its critical section, then no other processes can be executing in their critical sections.
- **Example :** Updating a DB or sending control signals to an I/O device
- Problem of mutual exclusion frequently arises in distributed systems whenever concurrent access to shared resources by several sites is involved.
- Mutual exclusion is the fundamental issue in the design of distributed systems.

- **Entry section :** The code executed in preparation for entering the critical section
- **Critical section :** The code to be protected from concurrent execution
- **Exit section :** The code executed upon leaving the critical section
- **Remainder section :** The rest of the code
- Each process cycles through these sections in the order : *remainder, entry, critical, exit.*

4.4.1 Requirement of Mutual Exclusion

1. Freedom from deadlocks : Two or more sites should not endlessly wait for message that will never arrive.
2. Freedom from starvation : A site should not wait indefinitely while other sites repeatedly access the CS.
3. Strict fairness : Requests are served in the (logical) order in which they arrive.
4. Fault tolerance : An algorithm should be able to detect and recover from failures.

4.4.2 Algorithm for Mutual Exclusion

- Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.

System model

- The system consists of N sites, $S_1, S_2, \ldots, S_N$. We assume that a single process is running on each site. The process at site S_i is denoted by pi .
- At any instant, a site may have several requests for critical section. A site queues up these requests and serves them one at a time.
- Site may in one of the three states :
  1. Requesting CS
  2. Executing CS
  3. Neither requesting nor executing requests for CS

Classification of Mutual Exclusion

- Different types of algorithm are used to solve problem of mutual exclusion in distributed system. But these algorithms differ in their communication topology. Topology may be ring, bus, star etc. They also maintain different types of information.
- These algorithms are divided into two classes :
  1. Non - token based : Require multiple rounds of message exchanges for local states to stabilize.

2. Token based : Permission passes around from one site to another. Site is allowed to enter its critical section if it possesses the token and it continues to hold the token until the execution of the critical section is over.

4.4.3 Performance Metrics for Mutual Exclusion Algorithms

1. Message complexity : The number of messages required per CS execution by a site.
2. Synchronization delay : After a site leaves the CS, it is the time required before the next site enters the CS.
3. Response time : The time interval a request waits for its CS execution to be over after its request messages have been sent out.
4. System throughput : The rate at which the system executes requests for the CS.

   System throughput = 1/(SD+E)

   where SD is the synchronization Delay and E is the average critical section execution time

- Fig. 4.4.1 shows the synchronization delay.

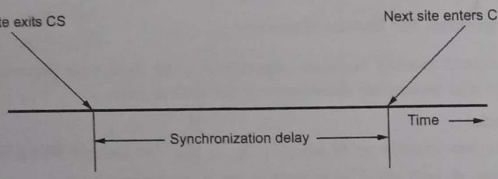


Fig. 4.4.1 Synchronization delay

- Synchronization delay (*sd*) : A leaves → B enters
- Response time : A requests → A leaves
- Throughput : CS requests handled per time unit
  = 1 / (*SD* + *E*) where *E* is average execution time of CS.
- Performance of mutual exclusion algorithm depends upon the loading condition of the system.
- Performance may depend on whether load is low or high. Best case, worst case, and average cases are all of interest.
- If the load is high then there is always pending requests for mutual exclusion.
- Fig. 4.4.2 shows the response time.

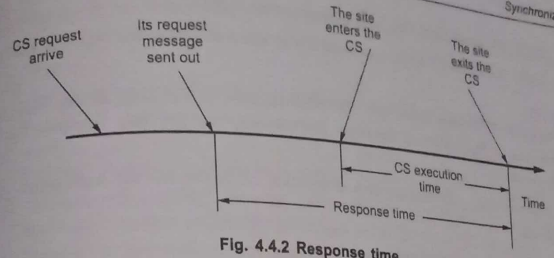


Fig. 4.4.2 Response time

4.4.4 Central Server Algorithm

- The simplest way to ensure mutual exclusion is through the use of a centralized server. Here server grants the permission to enter the critical section. Fig. 4.4.3 shows concept of centralized server algorithm.

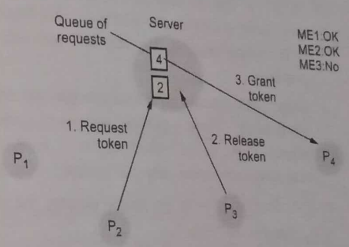


Fig. 4.4.3 Centralized server algorithm

- There is a conceptual token; processes must be in possession of the token in order to execute the critical section.
- The centralized server maintains ownership of the token. To request the token; a process sends a request to the server. If the server currently has the token it immediately responds with a message, granting the token to the requesting process.
- When the process completes the critical section it sends a message back to the server, relinquishing the token.
- If the server doesn't have the token, some other process is "currently" in the critical section. In this case the server queues the incoming request for the token and responds only when the token is returned by the process directly ahead of the requesting process in the queue.
- Given our assumptions that no failures occur it is straight forward to see that the central server algorithm satisfies the safety and liveness properties.
- ME1 : (safety) At most one is executing in the critical section at a time. (OK)

- ME2 : (liveness) Requests eventually succeed. (OK)
- ME3 : (-> ordering) Happened-before is granted at entries for requests. (NO)

Properties

- Satisfies ME1 and ME2, but not ME3 because of network delays may reorder requests. There are two messages per request, one per exit, exit does not delay process.
- Problem with performance and availability of server are the bottlenecks for this algorithm.

4.4.5 Ring - Based Algorithm

- A simple way to arrange for mutual exclusion without the need for a master process, is to arrange the processes in a logical ring. The ring may ofcourse bear little resemblance to the physical network or even the direct links between processes.
- Token passes in one direction through the ring. The token passes around the ring continuously. When a process receives the token from its neighbour, if it does not require access to the critical section it immediately forwards on the token to the next neighbour in the ring.
- If it requires access to the critical section, the process :
  1. Retains the token
  2. Performs the critical section and then :
  3. To relinquish access to the critical section
  4. Forwards the token on to the next neighbour in the ring.
- Fig. 4.4.4 shows ring based algorithm.

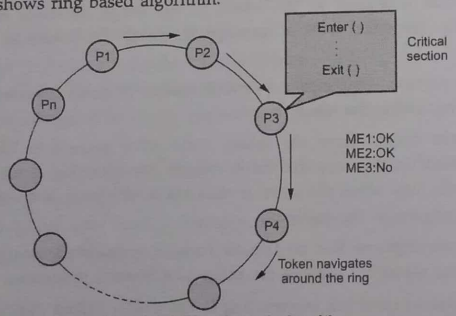


Fig. 4.4.4 Ring based algorithm

- Once again it is straight forward to determine that this algorithm satisfies the safety and liveness properties. However once again we fail to satisfy the fairness property.
- Suppose again we have two processes P1 and P4 consider the following events
  1. Process P1 wishes to enter the critical section but must wait for the token to reach it.
  2. Process P1 sends a message m to process P4.
  3. The token is currently between process P1 and P4 within the ring, but the message m reaches process P4 before the token.
  4. Process P4 after receiving message m wishes to enter the critical section
  5. The token reaches process P4 which uses it to enter the critical section before process P1.

Performance

- Constant bandwidth consumption
- Entry delay between 0 and N message transmission times
- Synchronization delay between 1 and N message transmission times

4.4.6 Algorithm using Multicast and Logical Clocks

- Ricart - Agrawala algorithm is an optimization on Lamport algorithm.
- Ricart - Agrawala algorithm uses only two types of messages : *REQUEST* and *REPLY*.
- It is assumed that all processes keep a logical clock which is updated according to the clock rules.
- The algorithm requires a total ordering of requests. Requests are ordered according to their global logical timestamps; if timestamps are equal, process identifiers are compared to order them.
- The process that requires entry to a CS multicasts the request message to all other processes competing for the same resource. Process is allowed to enter the CS when all processes have replied to this message. The request message consists of the requesting process' timestamp (logical clock) and its identifier.
- Each process keeps its state with respect to the CS : released, requested, or held.

Algorithm :

Requesting the critical section :

1. Request when Si wants to enter the critical section, it broadcast timestamped REQUEST message to all site.

2. When a process receives a REQUEST message, it may be in one of three states :

Case 1 : The receiver is not interested in the critical section, send reply (OK) to sender.

Case 2 : The receiver is in the critical section; do not reply and add the request to a local queue of requests.

Case 3 : The receiver also wants to enter the critical section and has sent its request. In this case, the receiver compares the timestamp in the received message with the one that it has sent out. The earliest timestamp wins. If the receiver is the loser, it sends a reply (OK) to sender. If the receiver has the earlier timestamp, then it is the winner and does not reply. Instead, it adds the request to its queue.

Executing the critical section :

3. When site Si enters critical section after it has received REPLY message from all the site in its request set.

Releasing the critical section :

4. When site Si exits the CS, it sends RELEASE(*i*) messages to all sites in its request set Ri.
5. When a site Sj receives the RELEASE(*i*) message from site, it sends a REPLY(*j*) message to the next site waiting in the queue and deletes that entry from the queue. If the queue is empty, then the site updates its state to reflect that the site has not sent out any REPLY message.

Optimization :

- Once site Si has received a REPLY message from a site Sj, the authorization implicit in this message remains valid until Si sends a REPLY message to Sj.
- Fig. 4.4.5 shows operation of Ricart-Agrawala algorithm.

Step 1 : Site S1 and S2 are making request for critical section.

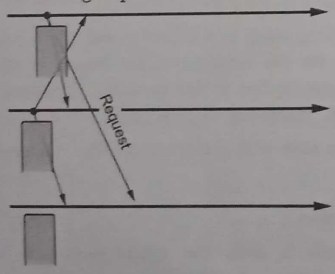


Fig. 4.4.5 (a)

Step 2 : Site S2 enters the critical section.

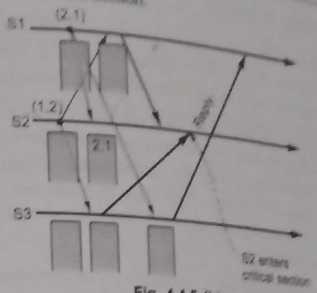


Fig. 4.4.5 (b)

Step 3 : Site S2 exits the CS and sends a REPLY messages to S1.

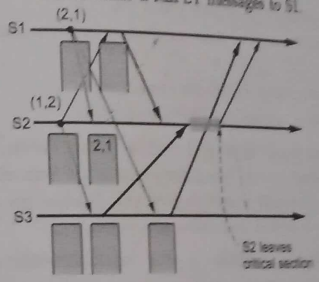


Fig. 4.4.5 (c)

Step 4 : Site S1 enters the critical section

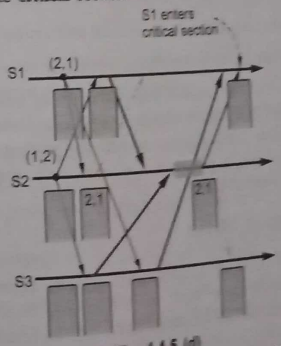


Fig. 4.4.5 (d)

4.4.7 Maekawas Voting Algorithm

- The main idea of the algorithm is to let the process that want to enter the critical section to compete for votes. Every process P has a voting district Sp. It is required that for all i and j, S_i and S_j have at least one common element.
- When a process wishes to enter the critical section, it sends a vote request to every member of its voting district. When the processor receives replies from all the members of the district, it can enter the critical section. When a processor receives a vote request, it responds with a "YES" vote if it has not already cast its vote. When a processor exits the critical section, it informs the voting district, which can then vote for other candidates.
- Each process P_i is associated with a voting set V_i of processes. The set V_i for the process P_i is chosen such that :
  1. $P_I \in V_i$: A process is in its own voting set.
  2. $V_i \cap V_j \neq \{\}$: There is at least one process in the overlap between any two voting sets.
  3. $|V_i| = |V_j|$: All voting sets are the same size.
  4. Each process P_i is contained within M voting sets.
- When a processor wants to enter a critical section, it sends a request to all members of its district. It may enter, if it gets a grant from all members. When a processor receives a request it answers with yes, if it has not already cast its vote. On exit it informs its district to enable a new voting.
- As before each process maintains a state variable which can be one of the following :
  1. **Released :** Does not have access to the critical section and does not require it.
  2. **Wanted :** Does not have access to the critical section but does require it.
  3. **Held :** Currently has access to the critical section.
- In addition each process maintains a boolean variable indicating whether or not the process has "voted". Of course voting is not a one-time action. This variable really indicates whether some process within the voting set has access to the critical section and has yet to release it. To begin with, these variables are set to "Released" and False respectively.

4.5 Global Positioning of Nodes

- When the number of nodes in a distributed system increases, then it becomes difficult for any node to keep track of the others. Such knowledge may be important for executing distributed algorithms such as routing, multicasting, data placement, searching, etc.

- In geometric overlay networks each node is given a position in an m - dimensional geometric space, such that the distance between two nodes in that space reflects a real - world performance metric.
- In other words, given two nodes P and Q, then the distance d (P, Q) reflects how long it would take for a message to travel from P to Q and vice versa.
- Fig. 4.5.1 shows computing a node's position in a two-dimensional space.

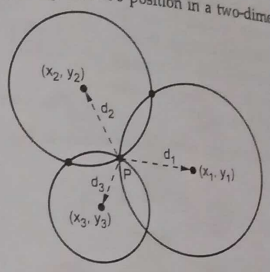


Fig. 4.5.1

- in GPS, node P can compute is own coordinates (x_P, y_P) by solving the three equations with the two unknowns x_P and y_P :

$$d_i = \sqrt{(x_i - x_p)^2 + (y_i + y_p)^2} \quad (i = 1,2,3)$$

4.6 Election Algorithms

- Many distributed algorithms require one process to act as co-ordinator, initiator, or otherwise perform some special role. It does not matter which process take on this special responsibility, but one of them has to do it.
- If all processes are exactly the same, with no distinguishing characteristics, there is no way to select one of them to be special.
- In general, election algorithms attempt to locate the process with the highest process number and designate it as co-ordinator.
- The goal of an election algorithm is to ensure that when an election starts, it concludes with all processes agreeing on who the new co-ordinator is to be.

4.6.1 The Bully Algorithm

- When any process notices that the co-ordinator is no longer responding to requests, it initiates an election. A process P, holds an election as follows :
  1. P sends an ELECTION message to all processes with higher numbers.

2. If no one responds, P wins the election and becomes co-ordinator.
3. If one of the higher-ups answers, it takes over. P's job is done.

- Fig. 4.6.1 (a) shows the bully algorithm. The group consists of eight processes, numbered from 0 to 7. Previously process 7 was the co-ordinator, but it has just crashed. Process 4 is the first one to notice this, so it sends ELECTION messages to all the processes higher than it, namely 5, 6, and 7.

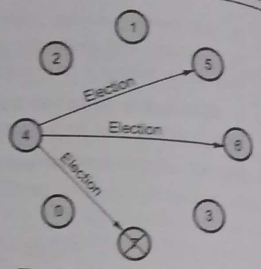


Fig. 4.6.1 (a) Bully algorithm

- Processes 5 and 6 both respond with OK, as shown in Fig. 4.6.1 (b).

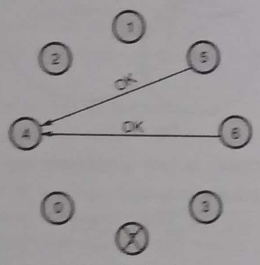


Fig. 4.6.1 (b)

- Upon getting the first of these responses, 4 knows that its job is over. It knows that one of these will take over and become co-ordinator.
- In Fig. 4.6.1 (c), both 5 and 6 hold elections, each one only sending messages to those processes higher than itself.

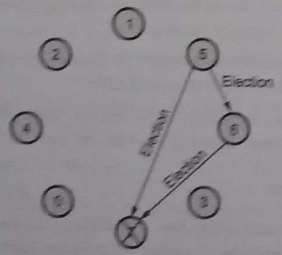


Fig. 4.6.1 (c)

- In Fig. 4.6.1 (d) process 6 tells 5 that it will take over. At this point 6 knows that 7 is dead and that it (6) is the winner.

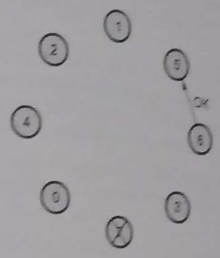


Fig. 4.6.1 (d)

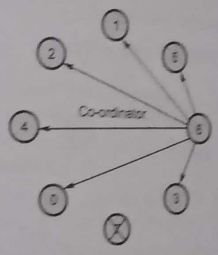


Fig. 4.6.1 (e)

- If there is state information to be collected from disk or elsewhere to pick up where the old co-ordinator left off, 6 must now do what is needed. When it is ready to take over, 6 announces this by sending a CO-ORDINATOR message to all running processes.
- When 4 gets this message, it can now continue with the operation it was trying to do when it discovered that 7 was dead, but using 6 as the co-ordinator this time. In this way the failure of 7 is handled and the work can continue.
- If process 7 is ever restarted, it will just send all the others a CO-ORDINATOR message and bully them into submission.

4.6.2 A Ring Algorithm

- When any process notices that the co-ordinator is not functioning, it builds an ELECTION message containing its own process number and sends the message to its successor.
- If the successor is down, the sender skips over the successor and goes to the next member along the ring, or the one after that until a running process is located. In each step, the sender adds its own process number to the list in the message effectively making itself a candidate to be elected as co-ordinator.
- Eventually, the message gets back to the process that started it all. That process recognizes this event when it receives an incoming message containing its own process number.
- At that point, the message type is changed to CO-ORDINATOR and circulated once again, this time to inform everyone else who the co-ordinator is and who the members of the new ring are. When which message has circulated once, it is removed and everyone goes back to work.
- Fig. 4.6.2 shows election algorithm using a ring.

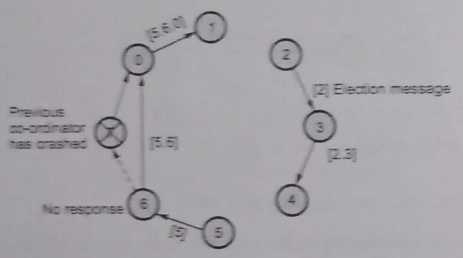


Fig. 4.6.2 Election algorithm using a ring

- If two processes, 2 and 5 discover simultaneously that the previous co-ordinator, process 7 has crashed. Each of these builds an ELECTION message and each of them starts circulating its message, independent of the other one.
- Both messages will go all the way around, and both 2 and 5 will convert them into CO-ORDINATOR messages with exactly the same number and in the same order. When both have gone around again, both will be removed. It does not harm to have extra message circulating at worst it consumes a little bandwidth, but this not considered wasteful.

4.6.3 Comparison between Ring and Bully Algorithm

Parameters	Ring	Bully
Asynchronous	Yes	No
Allows processes to crash	No	Yes
Satisfies Safety	Yes	Yes/No
Dynamic process identifiers	Yes	No
Dynamic configuration of processes	Maybe	Maybe
Best case performance	2 (N)	N - 2
Worst case performance	3 (N) - 1	O (N^2)

4.7 Fill in the Blanks

Q.1 Clock devices can be programmed to generate _____ at regular intervals in order that, for example, time slicing can be implemented.

Q.2 The instantaneous difference between the readings of any two clocks is called their _____.

Q.3 A clock's _____ failure is said to occur when the clock stops ticking altogether; any other clock failure is an arbitrary failure.

Q.4 The _____ algorithm eliminates readings from faulty clocks.

Q.5 The _____ defines an architecture for a time service and a protocol to distribute time information over the Internet.

Q.6 A _____ logical clock is a monotonically increasing software counter.

Q.7 A _____ is a service that processes queries about whether a particular process has failed.

Q.8 An algorithm for choosing a unique process to play a particular role is called an _____ algorithm.

Q.9 _____ clocks measure the time of day.

4.8 Multiple Choice Questions

Q.1 To enforce _____ two functions are provided enter-critical and exit-critical, where each function takes as an argument the name of the resource that is the subject of competition.

a deadlock b synchronization
c mutual exclusion d starvation

Q.2 A synchronization subnet is _____.

a the collection of NTP servers on the Internet.
b an IP multicast group used for clock synchronization
c the set of machines addressed by an NTP server operating in multicast mode
d the set of NTP servers with which you are currently synchronizing.

Q.3 IP multicast uses :

a Reliable multicast.
b Unreliable multicast.
c Atomic multicast.
d User - selectable reliability.

Q.4 The ring election algorithm works by _____.

a having all nodes in a ring of processors send a message to a coordinator who will elect the leader.
b sending a token around a set of nodes. Whoever has the token is the coordinator.
c sending a message around all available nodes and choosing the first one on the resultant list.
d building a list of all live nodes and choosing the largest numbered node in the list.

Q.5 The Ricart and Agrawala distributed mutual exclusion algorithm is _____.

a more efficient and more fault tolerant than a centralized algorithm.
b more efficient but less fault tolerant than a centralized algorithm.
c less efficient but more fault tolerant than a centralized algorithm.
d less efficient and less fault tolerant than a centralized algorithm.

Q.6 A client has a time of 5:05 and a server has a time of 5:25. Using the Berkeley algorithm, the client's clock will be set to :

a 5 : 15 b 5 : 20
c 5 : 25 d 5 : 30

Q.7 Which offers the most fault-tolerant message delivery?

a Atomic multicast.
b Totally ordered reliable multicast.
c Causally ordered reliable multicast.
d Hardware multicast.

Q.8 A bully election algorithm :

a Picks the first process to respond to an election request.
b Relies on majority vote to pick the winning process.
c Assigns the role of coordinator to the process holding the token at the time of election.
d Picks the process with the largest ID.

Q.9 Which mutual exclusion algorithm works when the membership of the group is unknown ?

a Centralized b Ricart - Agrawala
c Lamport d Token Ring.

Q.10 What are global locks ?

a To local resources
b They synchronize access to global resources
c They synchronize access to local and global resources
d None of above

Answer Keys for Fill in the Blanks

1.	interrupts	2.	skew
3.	crash	4.	Berkeley
5.	Network Time Protocol	6.	Lamport
7.	failure detector	8.	election
9.	Physical		

Answer Keys for Multiple Choice Questions

1.	c	2.	a
3.	b	4.	d
5.	d	6.	a
7.	a	8.	d
9.	a	10.	d

❑❑❑

5 Consistency, Replication and Fault Tolerance

Syllabus

Introduction To Replication, Data-Centric Consistency Models, Client-Centric Consistency Models, Replica Management, Consistency Protocols, Basics of Fault Tolerance, Process Resilience, Reliable Client-Server Communication, Reliable Group Communication, Distributed Commit, Recovery.

Contents

5.1 Introduction to Replication

- Replication refers to the maintenance of copies at multiple sites. Replication is a technique for enhancing services. A logical object is implemented by a collection of physical copies called **repglicas.**
- Motivation for replication is to improve a service's performance, to increase its availability, or to make it fault tolerant.

1. **Performance :** Placing copies of data close to the processes using them can improve performance through reduction of access time. If there is only one copy, then the server could become overloaded.
2. **Increase availability :** Factors that affect availability are server failures and network partitions. User requires services to be highly available. The availability of the service is that, if there are n replicated servers each of which would crash in a probability of p.
3. **Fault tolerance :** Highly available data is not necessarily strictly correct data. Guarantee strictly correct behavior despite a certain number and type of faults. It requires strict data consistency between all replicated servers. Replication of read-only data is simple, but replication of mutable data incurs overheads in form of protocol.
4. **Autonomous operation :** In a distributed system that provides file replication as a service to their clients, all files required an a client for operation during a limited time period may be replicated on the file server residing at the client's node. This will facilitate temporary autonomous operation of client machines. A distributed system having this feature can support detachable, portable machines.

5.1.1 Reasons for Replication

- There are two primary reasons for replicating data including reliability and performance.
- Data are generally replicated to enhance reliability or improve performance.
- One of the major problems is keeping replicas consistent. Informally, this means that when one copy is updated we need to ensure that the other copies are updated as well; otherwise the replicas will no longer be the same.

1. Data are replicated to increase the reliability of a system.
- If a file system has been replicated it may be possible to continue working after one replica crashes by simply switching to one of the other replicas.
- Also, by maintaining multiple copies, it becomes possible to provide better protection against corrupted data.

- For example, imagine there are three copies of a file and every read and write operation is performed on each copy.
- It can safe against a single, failing write operation, by considering the value that is returned by at least two copies as being the correct one.

2. Replication for performance
- **Scaling in numbers :** Replication for performance is important when the distributed system needs to scale in numbers and geographical area. Scaling in numbers occurs, for example, when an increasing number of processes needs to access data that are managed by a single server. In that case, performance can be improved by replicating the server and subsequently dividing the work.
- **Scaling in geographical area :** The basic idea is that by placing a copy of data in the proximity of the process using them, the time to access the data decreases. As a consequence, the performance as perceived by that process increases.

5.1.2 Replication as Scaling Technique

- Replication and caching is used for system scalability. Scalability issue generally appears in the form of performance problem.
- Performance is increases by reducing access time. This is possible when multiple copies of data is placed near to the object.
- It is necessary to keep up to date of data but it requires more bandwidth.
- Example : object is replicated N times. We consider R is read frequency and W is write frequency. If R<<W, it gives high consistency overhead and wasted messages.
- Keeping multiple copies of data is itself an issue of scalability problem. Collection of copies is consistent when the copies are always the same. The read operation performed at any copy will always return the same result. At the same time, when an update operation is performed on one copy, the update should be propagated to all copies before a subsequent operations takes place.
- What semantics to provide ? Tight consistency requires globally synchronized clocks. Global synchronization simply takes a lot of communication time, especially when replicas are spread across a wide area network. Solution to this problem is to **use loosen consistency**.
- The variety of consistency semantics possible for this. In this case copies are not always the same everywhere.

5.2 Data - Centric Consistency Models

- The general organization of a logical data store, physically distributed and replicated across multiple machines is shown in Fig. 5.2.1. Each process that can access data has its own local copy and write operations are propagated to the other copies.

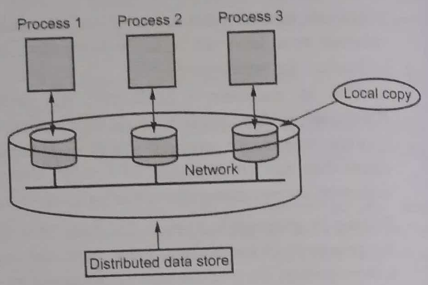


Fig. 5.2.1

- A data store is a distributed collection of storages accessible to clients.
- Consistency model is a contract between processes and the data store. If processes obey certain rules, data store will work correctly. All models attempt to return the results of the last write for a read operation.

5.2.1 Strict Consistency

- Every read to a memory location x returns the value most recently written to x. It requires a global time. A write is immediately visible to all processes.
- A shared memory system is said to support the strict consistency model if the value returned by the read operation on a memory address is always the same as the value written by the most recent write operation to that address.
- Not suitable in distributed systems. Difficult to achieve in real systems as network delays can be variable.

5.2.2 Sequential Consistency

- Linearizability is too strict for most practical purposes. The strongest memory model for DSM that is used in practice is sequential consistency.
- Any read to a memory location x should have returned (in the actual execution) the value stored by the most recent write operation to x in this sequential order.
- In this model, writes must occur in the same order on all copies; reads however can be interleaved on each system, as convenient.
- A DSM system is said to be sequentially consistent if for any execution there is some interleaving of the series of operations issued by all the processes that satisfies the following two criteria :

1. **SC1 :** The interleaved sequence of operations is such that if occurs in the sequence, then either the last write operation that occurs before it in the interleaved sequence is , or no write operation occurs before it and a is the initial value of x
2. **SC2 :** The order of operations in the interleaving is consistent with the program order in which each individual client executed them.

- The result of the execution of a parallel program is the same as if the program is executed on a single processor in a sequential order :

P: write x; write y; read x;

Q: read y; write x; read x;

- Some legitimate sequential orders :

P write x; P write y; P read x; Q read y; Q write x; Q read x;

P write x; Q read y; P write y; Q write x; P read x; Q read x;

Q read y; Q write x; P write x; P write y; P read x; Q read x

- All processors see the same sequence of memory references

a. Concurrently P : write x; Q : write x;

b. One process sees P writes first, then Q, then every process sees the same order.

- If write requests are processed exclusively, sequential consistency can be achieved.
- A sequential consistency memory model provides one - copy/single - copy semantics because all the processes sharing a memory location always see exactly the same contents stored in it.

5.2.3 Linearizability

- The result of any execution is the same as if the operations by all processes on the data were executed in some total order.
- The operations of each individual process appear in this sequence are in the order as how they actually happened in real time :

a. Bring in server's view to define the ordering of concurrent events.

b. Real times of how activities have actually happened are defined by the actions performed on the servers. It is define by the actual enqueuing time of each request.

c. Non-overlapping requests have to follow the order of the requests enqueuing times.

d. Overlapping requests : Enqueuing times of the requests are in different orders on different servers can have arbitrary order, but sequentially consistent.

5.2.4 Causal Consistency

- Operations that are causally related must be seen by all processes in the same corresponding order. Concurrent writes from different processors do not have any causal relationship and can be seen in different order by different processors. There is no need to write exclusively, cheaper write operations.
- P1 reads x and then writes y then writing to y is causally related to reading from x since y's value may have been computed from x.
- Any write to x by P1 before x was read by P2 has causal relation to the subsequent write by P2 to y.

a. P1: write x; P2: read x; write y;

b. If P2 reads x got P1's x value, then P2's write has to be after P1's write

- Need to keep track of dependency relations in order to determine whether two events are causally related. This model is expensive.
- **Following sequence obeys causal consistency :**

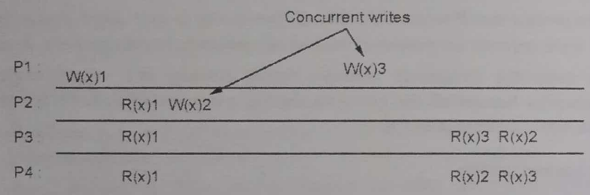


Fig. 5.2.2

- This sequence does not obey causal consistency :

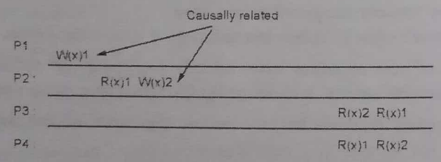


Fig. 5.2.3

5.2.5 Pipelined RAM Consistency

- Any pair of writes that are both done by a single processor are received by all other processors in the same order.

- A pair of writes from different processes may be seen in different orders at different processors.
- Following sequence is allowed with PRAM consistent memory. (Refer Fig. 5.2.4.)

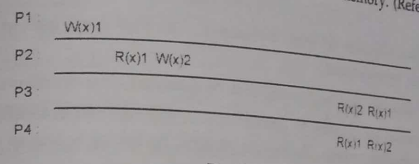


Fig. 5.2.4

- P3 and P4 observe the writes by P1 and P2 in different orders, although W(x)1 and W(x)2 are potentially causally related.
- PRAM consistency model is simple and easy to implement and also has good performance.
- PRAM consistency can be implemented by simply sequencing the write operations performed at each node independently of the write operations performed on other nodes.

5.2.6 Weak Consistency

- PRAM consistency still unnecessarily restrictive for many applications : requires that writes originating in single process be seen everywhere in order.
- A synchronization variable is introduced. When synchronization completes, all writes done on that processor are propagated outward and all writes done on other processors are brought in.
- The weak consistency has three properties :

1. Accesses to synchronization variables are sequentially consistent.
2. No access to a synchronization variable is allowed to be performed until all previous writes have completed everywhere.
3. No data access (read or write) is allowed to be performed until all previous accesses to synchronization variables have been performed.

- Let us consider following example :

P1 : W(x)1 W(x)2 S

P2 : R(x)1 R(x)2 S

P3 : R(x)2 R(x)1 S

A valid sequence of events for weak consistency :

P1 : W(x)1 W(x)2 S

P2 : S R(x)1

- An invalid sequence for weak consistency : P2 must get 2 instead of 1 because it is already synchronized.
- Shared data can only be counted on to be consistent after synchronization is done.
- All processes see accesses to synchronization variables in same order. Accessing a synchronization variable "flushes the pipeline" by forcing writes to complete.
- By doing synchronization before reading shared data, a process can be sure of getting the most recent values.
- Weak consistency requires the programmer to use locks to ensure reads and writes are done in the proper order for data that needs it.

5.2.7 Entry Consistency

- With release consistency, all local updates are propagated to other copies or servers during release of shared data.
- With *entry consistency*, each shared data item is associated with a synchronization variable.
- In order to access consistent data, each synchronization variable must be explicitly acquired.
- Release consistency affects all shared data but entry consistency affects only those shared data associated with a synchronization variable.
- A data store exhibits entry consistency if it meets all of the following conditions :

1. Synchronization variable is not allowed to perform operation with respect to a process until all updates of shared data is not performed.
2. Before using a shared variable by process, no other process is allowed to use the shared resources.
3. Owner is only access of synchronization variable, for other process, non-exclusive mode synchronization is also not allowed.

5.3 Client - Centric Consistency Models

- System wide view on data store is provided by data - centric consistency model. Client - centric consistency models are generally used for applications that lack simultaneous updates, i.e., most operations involve reading data.

- The following are very weak, client-centric consistency models :

1. Eventual consistency
2. Monotonic reads
3. Monotonic writes
4. Read your writes
5. Writes follow reads.

5.3.1 Eventual Consistency

- The data stores offer a very weak consistency model, called eventual consistency. The **eventual consistency** model states that, when no updates occur for a long period of time, eventually all updates will propagate through the system and all the replicas will be consistent.
- Systems such as domain name system and world wide web can be viewed as applications of large scale distributed and replicated databases that tolerate a relatively high degree of inconsistency. They have in common that if no updates take place for a long time, all replicas will gradually and eventually become consistent. This form of consistency is called ***eventual consistency***.
- Eventual consistency requires only that updates are guaranteed to propagate to all replicas. Eventual consistent data stores work fine as long as clients always access the same replica.
- Eventual consistency for replicated data is fine if clients always access the same replica. Client centric consistency provides consistency guarantees for a single client with respect to the data stored by that client.
- **What happens when different replicas are accessed ?**

Example : Consider a distributed database to which you have access through your notebook. Assume your notebook acts as a front end to the database. At location A you access the database doing reads and updates. At location B you continue your work, but unless you access the same server as the one at location A, you may detect inconsistencies, because :

1. Your updates at A may not have yet been propagated to B
2. You may be reading newer entries than the ones available at A
3. Your updates at B may eventually conflict with those at A

- Fig. 5.3.1 shows distributed database for mobile user. (Refer Fig. 5.3.1 on next page)
- For the mobile user example, eventual consistent data stores will not work properly. Client - centric consistency provides guarantees for a single client concerning the consistency of access to a data store by that client. No guarantees are given concerning concurrent accesses by different clients.

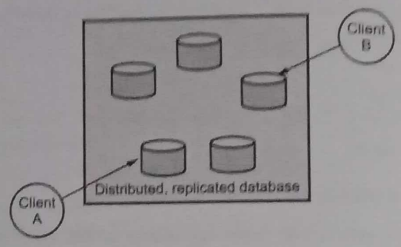


Fig. 5.3.1 Distributed database for mobile user

5.3.2 Monotonic - Read Consistency

- Assume read operations by a single process P at two different local copies of the same data store.
- Once read, subsequent reads on that data items return same or more recent values.
- Example : Automatically reading your personal calendar updates from different servers.
- Monotonic Reads guarantees that the user sees all updates, no matter from which server the automatic reading takes place.
- Example : Reading (not modifying) incoming mail while you are on the move.
- Each time you connect to a different e-mail server, that server fetches (at least) all the updates from the server you previously visited.
- Example : The read operations performed by a single process P1 at two different local copies of the same data store.
- The vertical axis shows the two different local copies of the data store. We called as Location1 and Location2.
- Horizontal axis shows the time. Operations carried out by a single process P1 in boldface are connected by a dashed line representing the order in which they are carried out.
- Fig. 5.3.2 shows monotonic read operation.

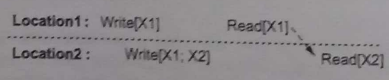


Fig. 5.3.2 (a) A monotonic - read consistent data store

- Process P1 first performs a read operation on X at Location1, returning the value of X1. This value results from the write operations in Write (X1) performed at Location1. Later, P1 performs a read operation on X at Location2, shown as Read (X2).
- To guarantee monotonic-read consistency, all operations in Write (X1) should have been propagated to Location2 before the second read operation takes place.

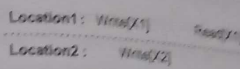


Fig. 5.3.2 (b) A data store that does not provide monotonic reads

- Situation in which monotonic-read consistency is not guaranteed. After process P1 has read X1 at Location1, it later performs the operation Read (X2) at Location2. But, only the write operations in Write (X2) have been performed at Location2. No guarantees are given that this set also contains all operations contained in Write (X1).

5.3.3 Monotonic-Write Consistency

- In a monotonic-write consistent store, the following condition holds :

A write operation by a process on a data item x is completed before any successive write operation on x by the same process.

- A write operation on a copy of item x is performed only if that copy has been brought up to date by means of any preceding write operation, which may have taken place on other copies of x. If need be, the new write must wait for old ones to finish.
- Example : Updating a program at server S2, and ensuring that all components on which compilation and linking depends, are also placed at S2.
- Example : Maintaining versions of replicated files in the correct order everywhere.
- The write operations performed by a single process P at two different local copies of the same data store
- Resembles to PRAM, but here we are considering consistency only for a single process (client) instead of for a collection of concurrent processes.
- Fig. 5.3.3 shows monotonic - write consistent data store and data store that does not provide monotonic-write consistency.

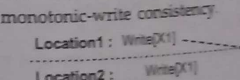


Fig. 5.3.3 (a) A monotonic - write consistent data store

- The Write(X2) requires that Write(X1) is updated on Location2 before it.

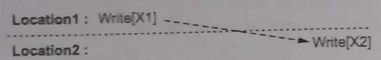


Fig. 5.3.3 (b) Store that does not provide monotonic - write consistency

- Write(X1) has not been propagated to Location2
- Example 1 : Updating a program at server S2, and ensuring that all components on which compilation and linking depends, are also placed at S2.
- Example 2 : Maintaining versions of replicated files in the correct order everywhere.

5.3.4 Read Your Writes

- It is closed related to monotonic reads consistency. A write operation is always completed before a successive read operation by the same process.
- Example : Editor and browser, if not integrated, you may not read-your-writes of an HTML page.
- Example : Updating your Web page and guaranteeing that your Web browser shows the newest version instead of its cached copy.

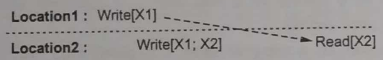


Fig. 5.3.4 (a) A data store that provides read-your-writes consistency

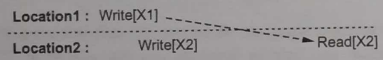


Fig. 5.3.4 (b) A data store that does not provides read-your-writes consistency

Writes Follow Reads

- Updates are propagated as the result of previous read operation.
- Any successive write operation on x by a process will be performed on a copy of x that is most recently read by that process.
- Ex : comments on news group, let A an article read recently, R the response to that article, then R must follows A.

5.4 Replica Management

- Key issue for any distributed system that supports replication is to decide where, when, and by whom replicas should be placed, and subsequently which mechanisms to use for keeping the replicas consistent.
- The placement problem itself should be split into two sub-problems: that of placing replica servers, and that of placing content.
- Replica-server placement is concerned with finding the best locations to place a server that can host (part of) a data store.
- Content placement deals with finding the best servers for placing content. Before content placement can take place, replica servers will have to be placed first.
- In the following, take a look at these two different placement problems, followed by a discussion on the basic mechanisms for managing the replicated content.
- Replica - server placement is often more of a management and commercial issue than an optimization problem. Fig. 5.4.1 shows choosing a proper cell size for server placement.

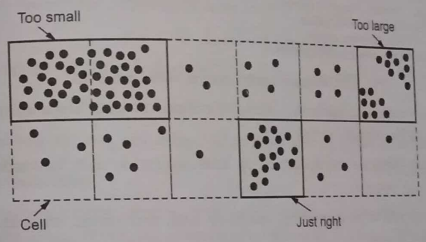


Fig. 5.4.1 Choosing a proper cell size for server placement

5.5 Consistency Protocols

- Consistency protocol describes an implementation of a specific consistency model.

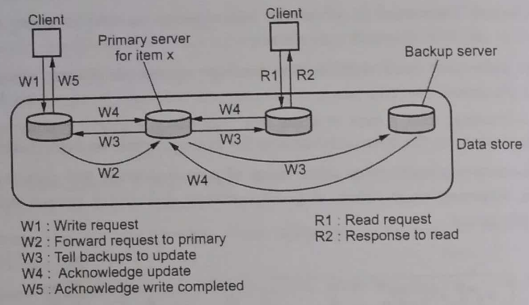
5.5.1 Primary - Based Protocols

- It is used for sequential consistency. Each data item is associated with a "primary" replica.
- The primary is responsible for coordinating writes to the data item.
- There are two types of Primary-Based Protocol :

1. Remote-Write 2. Local-Write

1. Remote - Write Protocols

- It is primary backup protocols. All writes are performed at a single (remote) server.
- Read operations can be carried out locally. This model is typically associated with traditional client/server systems.
- Example : Fig. 5.5.1 shows primary based remote write protocol.



W1 : Write request
W2 : Forward request to primary
W3 : Tell backups to update
W4 : Acknowledge update
W5 : Acknowledge write completed

R1 : Read request
R2 : Response to read

Fig. 5.5.1 Primary based remote write protocol

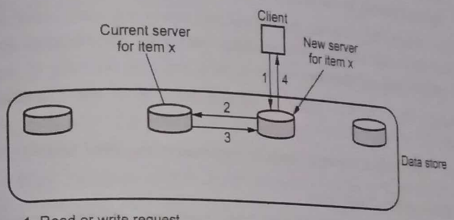
1. A process wanting to perform a write operation on data item x, forwards that operation to the primary server for x.
2. The primary performs the update on its local copy of x, and forwards the update to the backup servers.
3. Each backup server performs the update as well, and sends an acknowledgment back to the primary.
4. When all backups have updated their local copy, the primary sends an acknowledgment back to the initial process.

- All of those writes can take a long time. Using a non-blocking write protocol to handle the updates can lead to fault tolerant problems.
- As the primary is in control, all writes can be sent to each backup replica IN THE SAME ORDER, making it easy to implement sequential consistency.

2. Local - Write Protocols

- It is fully migrating approach. A single copy of the data item is still maintained.
- Upon a write, the data item gets transferred to the replica that is writing. The status of primary for a data item is transferrable.

- Process : Whenever a process wants to update data item x, it locates the primary copy of x, and moves it to its own location.
- Example : Fig. 5.5.2 shows local write protocol.
- Primary-based local - write protocol in which a single copy is migrated between processes (prior to the read/write).



1. Read or write request
2. Forward request to current server for x
3. Move item x to client's server
4. Return result of operation on client's server

Fig. 5.5.2 : Local write protocol

5.5.2 Replicated - Write Protocols

- It is Distributed - Write Protocols. Writes can be carried out at any replica.
- There are two types : Active Replication and Majority Voting (Quorums).

5.5.2.1 Active Replication

- In active replication model, there are multiple replica managers, each with equivalent roles. The replica manager's operate as a group and each front end (client interface) multicasts requests to a group of RM's.
- Requests are processed by all RM's independently. Client interface compares all replies received and can tolerate N out of 2N+1 failure, i.e. consensus when N + 1 identical response received. This model also can tolerate Byzantine failure.

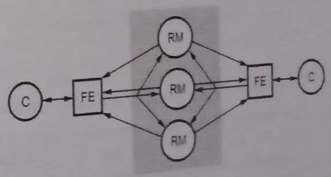


Fig. 5.5.3 Active replication

- Fig. 5.5.3 shows active replication.

- The sequence of events when a client requests an operation to be performed is as follows :

1. Request : Client request is sent to group of RM's using totally ordered reliable multicast, each sent with unique request id.
2. Co-ordination : The group communication system delivers the request to every correct replica manager in the same order.
3. Execution : Each RM processes the request and sends response/result back to the front end. Front end collects responses from each RM. The response contains the client's unique request identifier.
4. Agreement : No agreement phase is needed, because of the multicast delivery semantics.
5. Response : Each replica manager sends its response to the front end.

Fault Tolerance in Active Replication

- Replica manager's work as replicated state machines, playing equivalent roles. That is, each responds to a given series of requests in the same way. This is achieved by running the same program code at all RMs. If any RM crashes, state is maintained by other correct RMs.
- This system implements sequential consistency. The total order ensures that all correct replica managers process the same set of requests in the same order. Each front end's requests are served in FIFO order.
- So, requests are FIFO - total ordered. But if clients are multi-threaded and communicate with one another while waiting for responses from the service, we may need to incorporate causal-total ordering.

5.5.2 Quorum based Protocols

- A quorum is the minimum number of votes that a distributed transaction has to obtain in order to be allowed to perform an operation in a distributed system. A quorum - based technique is implemented to enforce consistent operation in a distributed system.
- A quorum system engages a designated minimum number of the replicas for every read or write, this number is called the **read quorum** or **write quorum.**
- Let there be N servers. To complete a write, a client must successfully place the updated data item on more than N/2 servers. The updated data item will be assigned a new version number that is obtained by incrementing its current version number.

- To read the data, the client must read out the copies from any subset of at least N/2 servers. From these, it will find out the copy with the highest version number and accept that copy.
- Only one writer at a time can achieve write quorum. Every reader sees at least one copy of the most recent read.

5.6 Basics of Fault Tolerance

- A system is said to fail when it cannot meet its promises. If a distributed system is designed to provide its users with a number of services, the system has failed when one or more of those services cannot be provided.
- An **error** is a part of a system's state that may lead to a failure. The cause of an error is called a **fault**.
- A good fault - tolerant system design requires a careful study of failures causes of failures, and system responses to failures. A fault is a malfunction, possibly caused by a design error, manufacturing error, programming error, physical damage, deterioration in the course of time, harsh environmental conditions and unexpected inputs.
- **Fault-Tolerance :** The system can provide services even in the presence of faults.

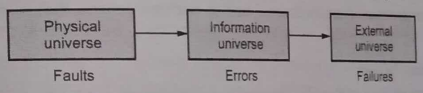


Fig. 5.6.1

Requirements :

1. **Availability :** It is defined as the property that a system is ready to be used immediately. The fraction of the time that a system meets its specification. The probability that the system is operational at a given time t.
2. **Reliability :** It refers to the property that a system can run continuously without failure. Typically used to describe systems that cannot be repaired or where the continuous operation of the system is critical.
3. **Safety :** It refers to the situation that when a system temporarily fails to operate correctly.
4. **Maintainability :** It refers to how easy a failed system can be repaired.

5.6.1 Failure Models

- Defines the ways in which failure may occur in order to provide an understanding of its effects.

- A taxonomy of failures which distinguish between the failures of processes and communication channels is provided :

1. *Omission* failures : Process or channel failed to do something.
2. *Arbitrary* failures : Any type of error can occur in processes or channels (worst).
3. *Timing* failures : Applicable only to synchronous distributed systems where time limits may not be met.

Fault models : Following are the fault models

- Omission faults
- Arbitrary faults
- Timing faults

- Faults can occur both in processes and communication channels. The reason can be both software and hardware faults.
- Fault models are needed in order to build systems with predictable behaviour in case of faults.
- Of course, such a system will function according to the predictions, only as long as the real faults behave as defined by the "fault model".

1. Omission failures

- A process or communication channel fails to perform actions that it is supposed to do.

Process omission failures :

- Process has crashed and can be detected using timeouts.
- Fail-stop process crash is one that can be detected with certainty by other processes.
- Process crash is called **Fail-stop**

i. If other processes can detect certainly that process has been crashed. *(fail to respond)*

ii. Can be produced in synchronous systems only where message delivery is guaranteed.

- In an asynchronous system a timeout can indicate only that a process is not responding. It may have crashed or may be slow, or the messages may not have arrived.

Communication omission failures :

- Communication primitives are **send** and **receive**.
- **Send-omission :** Loss of messages between the sending process and the outgoing message buffer (both inclusive).
- **Channel omission :** Loss of message in the communication channel.

- **Receive-omission :** Loss of messages between the incoming message buffer and the receiving process (both inclusive).

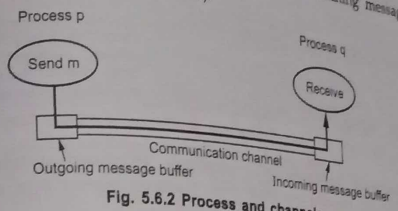


Fig. 5.6.2 Process and channels

Arbitrary failures :

- **Arbitrary process failure :** Arbitrarily omits intended processing steps or takes unintended processing steps.
- **Arbitrary channel failures :** Messages may be corrupted, duplicated, delivered out of order, incur extremely large delays; or non - existent messages may be delivered.
- Above two are Byzantine failures, e.g., due to hackers, man-in-the-middle attacks, viruses, worms, etc.
- A variety of Byzantine fault-tolerant protocols have been designed in literature.
- Arbitrary failures in processes cannot be detected by seeing whether the process responds to invocations, because it might arbitrarily omit to reply.
- Communication channel also suffer from arbitrary failures. For examples : Messages contents can be corrupted, a duplicate message can be sent or message can be lost on its way. Rare and can be detected by checksum or message numbering.
- Omission and arbitrary failures are as follows :

Sr. No.	Class of failure	Affects	Description
1.	Fail-stop or Crash-stop	Process	Process halts and remains halted. Other processes may detect this state.
2.	Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
3.	Send-omission	Process	A process completes a send, but the message is not put in its outgoing message buffer.

4.	Receive - omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
5.	Arbitrary (Byzantine)	Process or Channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

Timing failures :

- Timing failures are applicable in synchronous distributed systems where time limits are set on process execution time, message delivery time and clock drift rate.
- In an asynchronous distributed system, an overloaded server may respond too slowly, but we can not say that it has a timing failure since no guarantee has been offered.
- Timing failures are listed below :

Sr. No.	Class of failure	Affects	Description
1.	Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
2.	Performance	Process	Process exceeds the bounds on the interval between two steps.
3.	Performance	Channel	A message's transmission takes longer than the stated bound.

Masking failures :

- Knowledge of the failure characteristic of a component can enable us to develop a reliable service which uses such components which can fail.
- For example : Converting failure, checksum, retransmit message, replication, restoring information (convert arbitrary failure to omission failure).
- A service masks a failure, either by hiding it altogether or by converting it into a more acceptable type of failure.

Reliability of one-to-one communication :

- Reliable communication is defined in terms of validity and integrity. Correct message delivery in presence of failure.

1. **Validity :** Any message in the outgoing message buffer is eventually delivered to the incoming message buffer.
2. **Integrity :** The message received is identical to one sent, and no messages are delivered twice.
3. **Threats :** Malicious users and protocols.

5.6.2 Failure Masking by Redundancy

- There are three kinds of fault tolerance approaches :
  1. Information redundancy : Extra bit to recover from garbled bits.
  2. Time redundancy : Do again.
  3. Physical redundancy : Add extra components. There are two ways to organize extra physical equipment : **active replication** (use the components at the same time) and **primary backup** (use the backup if one fails).
- In the active replication technique, also called state-machine approach. All replicas play the same role : There is no centralized control.
- The active replication technique requires that the invocations of client processes be received by the non-faulty replicas in the same order. This requires an adequate communication primitive, ensuring the order and the atomicity property.

Types of Redundancy :

1. Hardware Redundancy : Based on physical replication of hardware.
2. Software Redundancy : The system is provided with different software versions of tasks, preferably written independently by different terms.
3. Time Redundancy : Based on multiple executions on the same hardware in different times.
4. Information Redundancy : Based on coding data in such a way that a certain number of bit errors can be detected and/or corrected.

Triple modular redundancy :

- Triple Modular Redundancy (TMR) uses three identical modules, performing identical operations, with a majority voter determining the output. Fig. 5.6.3 shows TMR.

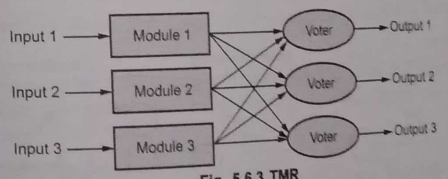


Fig. 5.6.3 TMR

- Using of multiple levels of redundancy for fault tolerance was established.
- Using automata theory (Logic gates, state machines, combinatorial/sequential logic) to model digital circuits and computational operations. By making reliable computers from less reliable components.
- Triple modular redundancy with triplicated voters can be used to overcome susceptibility to voter failure. The voter is no longer a single point of failure in the system.

5.7 Process Resilience

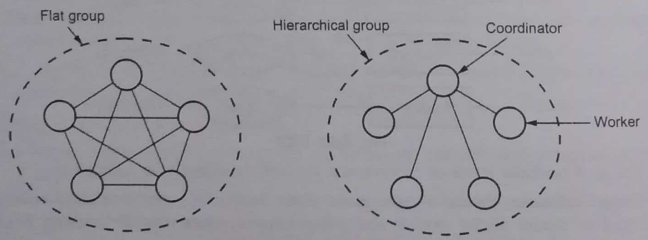
- Resilience can be defined as the capability of the system to understand and manage failures occurring in the system. Resiliency of a system is directly proportional to its up - time and availability. The more resilient the systems, the more available it is to serve users.
- Processes can be made fault tolerant by arranging to have a group of processes, with each member of the group being identical.
- A message sent to the group is delivered to all of the "copies" of the process (The group members), and then only one of them performs the required service. If one of the processes fail, it is assumed that one of the others will still be able to function.

5.7.1 Design Issue

- To tolerate a faulty process, organize several identical processes into a group. A group is a single abstraction of a collection of processes.
- So we can send a message to a group without explicitly knowing who are they, how many are there, or where are they (e.g., e-mail groups, newsgroups).
- Key property : When a message is sent, all members of the group must receive it. So, if one fails, the others can take over for it.
- Groups could be dynamic. We need mechanisms to manage groups and membership (e.g., join, leave, be part of two groups)

Flat groups versus hierarchical groups

- Fig. 5.7.1 shows communication in a flat group and communication in a simple hierarchical group.



(a) Communication in a flat group

(b) Communication in a simple hierarchical group

Fig. 5.7.1

1. Communication in a flat group :

- All the processes are equal and decisions are made collectively.
- There is no single point-of-failure, however decision making is complicated as consensus is required.
- Good for fault tolerance as information exchange immediately occurs with all group members. May impose overhead as control is completely distributed, and voting needs to be carried out.
- Harder to implement.

2. Communication in a simple hierarchical group :

- One of the processes is elected to be the coordinator, which selects another process (a worker) to perform the operation.
- Not really fault tolerant or scalable
- However, easier to implement

5.7.2 Failure Masking and Replication

- A failure of a system occurs when the system cannot meet its promises.
- Failures are caused by faults. A fault is an anomalous condition. There are three categories of faults :

1. Transient faults : Occur once and never reoccur (e.g., wireless communication being interrupted by external interference).
2. Intermittent faults : Reoccur irregularly (e.g., a loose contact on a connector).
3. Permanent faults : Persist until the faulty component is replaced (e.g., software bugs).

- **Fault tolerance** means that a system can provide its services even in the presence of faults.
- Failure masking is a fault tolerance technique that hides occurrence of failures from other processes. The most common approach to failure masking is **redundancy.**
- Three types of redundancy :

1. **Information redundancy :** Add extra bits to allow recovery from garbled bits
2. **Time redundancy :** Repeat an action if needed
3. **Physical redundancy :** Add extra equipment or processes so that the system can tolerate the loss or malfunctioning of some components.

Process Resilience :

- Processes can be made fault tolerant by arranging to have a group of processes, with each member of the group being *identical*.
- A message sent to the group is delivered to all of the "copies" of the process (the group members), and then *only one* of them performs the required service.
- If one of the processes fail, it is assumed that one of the others will still be able to function (and service any pending request or operation).

5.7.3 Byzantine Agreement Problem

- The Problem : "Several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. After observing the enemy, they must decide upon a common plan of action. Some of the generals may be traitors, trying to prevent the loyal generals from reaching agreement."
- Three or more generals are agree to attack or to retreat. Once the commander is issues the order, lieutenants to the commander are to decide to attack or retreat.
- But the one or more of the generals may be treacherous, i.e. faulty.
- If the commander is treacherous, he proposes attacking to one general and retreating to another.
- If a lieutenant is treacherous, he tells one of his peers that the commander told him to attack and another that they are to retreat.
- Source processor broadcasts its values to others. Solution must meet following objectives :
  Agreement : All non-faulty processors agree on the same value.
  Validity : If source is nonfaulty, then the common agreed value must be the value supplied by the source processor.
- "If source is faulty then all non - faulty processors can agree on any common value". "Value agreed upon by faulty processors is irrelevant"
- Fig. 5.7.2 shows Byzantine agreement.

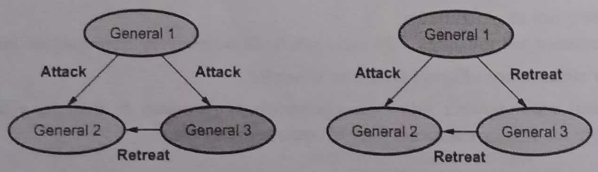


Fig. 5.7.2 Byzantine agreement

- No solution for three processes can handle a single traitor. In a system with m faulty processes agreement can be achieved only if there are $2m+1$ (more than 2/3) functioning correctly.

5.7.4 Failure Detection

- Each pair of processes is connected by reliable channels and processes are independent from each other. Processes only fail by crashing.
- Underlying network components may suffer failures, but reliable protocols recover. Reliable channel *eventually* delivers message. No bound as in an asynchronous system.
- A collection of processes is split into several groups that cannot communicate. Failure of router between two networks may mean that a collection of four processes is split into two pairs. Here intra - pair communication is possible but because of router fail this communication is also not possible. This is called as **network partition.**
- Fig. 5.7.3 shows network partition.
- Failure detector is object/code in a process that detects failures of other processes. Failure detector is not every time accurate. This is category as unreliable failure detector. Unreliable failure detector is one of two values : ***Unsuspected*** *or* ***Suspected.***

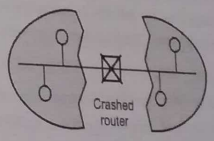


Fig. 5.7.3 Network partition

- **Unsuspected :** Failure is unlikely, for example : Failure detector has recently received communication from unsuspected peer. This may be inaccurate.
- **Suspected :** Indication that peer process failed. For example : No message received in quite sometime. Also this may be inaccurate because peer process hasn't failed, but the communication link is down, or peer process is much slower than expected.

A simple algorithm

- If we assume that all messages are delivered within some bound, say D seconds. Then we can implement a simple failure detector as :
- Every process p sends a "p is still alive" message to all failure detector processes periodically, once every T seconds.

- If a failure detector process does not receive a message from process q within T + D seconds of the previous one then it marks q as "Suspected".
- If we choose our bound D too high then often a failed process will be marked as "Unsuspected". A synchronous system has a known bound on the message delivery time and the clock drift and hence can implement a reliable failure detector. An asynchronous system could give one of three answers : "*Unsuspected, Suspected or Failed*" choosing two different values of D.
- In fact we could instead respond to queries about process p with the probability that p has failed, if we have a known distribution of message transmission times. For example : If you know that 90 % of messages arrive within 2 seconds and it has been two seconds since your last expected message you can conclude there is a : *NOT* a 90 % chance that the process p has failed.

5.8 Reliable Client - Server Communication

- A communication channel may lose and/or corrupt messages.
- How to handle communication failures? Use a reliable transport protocol (e.g., TCP) or handle at the application layer.
- **Techniques for reliable communication**

a. Use redundant bits to detect bit errors in packets

b. Use sequence numbers to detect packet loss

c. Recover from corrupted/lost packets using.

- **Five types of failures can occur in RPC**

1. Client cannot locate server
2. Server crashes after receiving a request
3. Client request is lost
4. Server response is lost
5. Client crashes after sending a request.

a. Server Crashes after Receiving a Request

- The client cannot tell if the crash occurred before or after the request is carried out
- Three possible semantics

1. **At - least-once :** keep trying until a reply is received
2. **At - most-once :** give up immediately and report back failure
3. **Exactly once :** desirable but not achievable.

b. Lost Request / Reply Messages

- Client waits for reply message, resents the request upon timeout

- Problem : Upon timeout, client cannot tell whether the request was lost or the reply was lost
- Client can safely resend the request for idempotent operations
- An idempotent operation is an operation that can be safely repeated
- E.g., reading the first line of a file is idempotent, transferring money is not
- For non-idempotent operations, client can add sequence numbers to requests so that the server can distinguish a retransmitted request from an original request
- Server need keep track of the most recently received sequence number from each client. Server will not carry out a retransmitted request, but will send a reply to the client.
- **Client Crashes after Sending a Request**
- What happens to the server computation, referred to as an orphan ?
- Extermination : Client explicitly kills off the orphan when it comes back up Client stub makes a log entry on disk before sending an RPC message.
- Reincarnation: When a client reboots, it broadcasts a new epoch number: when server receives the broadcast, it kills the computations that were running on behalf of the client.
- Expiration : Each RPC is associated with an expiration time T.
- The call is aborted when the expiration time is reached. If RPC cannot finish within T, the client must ask for another quantum.
- If after a crash the client waits a time T before rebooting, all orphans are sure to be gone.

5.9 Reliable Group Communication

- Reliable multicast services guarantee that all messages are delivered to all members of a process group.
- Small group : multiple, reliable point-to-point channels will do the job, however, such a solution scales poorly as the group membership grows.
- What happens if a process joins the group during communication ?
- Worse : What happens if the sender of the multiple, reliable point-to-point channels crashes half way through sending the messages ?
- Reliability, deals with recovering from communication failures, such as buffer overflows and garbled packets. Because reliability is more difficult to implement for group communication than for point-to-point communication, a number of existing operating systems provide unreliable group communication, whereas

almost all operating systems provide reliable point-to-point communication, for example, in the form of RPC.

Basic Reliable - Multicasting Schemes :

- Simple solution to reliable multicasting when all receivers are known and are assumed not to fail.
- The sending process assigns a sequence number to outgoing messages. Assume that messages are received in the order they are sent.
- Each multicast message is stored locally in a history buffer at the sender. Assuming the receivers are known to the sender, the sender simply keeps the message in its history buffer until each receiver has returned an acknowledgment.
- If a receiver detects it is missing a message, it may return a negative acknowledgment, requesting the sender for a retransmission
- Another important design decision in group communication is the ordering of messages sent to a group. Roughly speaking, there are four possible orderings: no ordering, FIFO ordering, causal ordering, and total ordering.

5.9.1 Message Ordering

- R_1 and R_2 receive m_1 and m_2 in a different order. Fig. 5.9.1 shows no ordering constraint for message delivery.

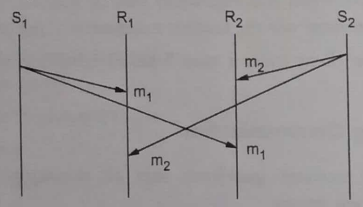


Fig. 5.9.1 No ordering

- Some message ordering required,

1. Absolute ordering
2. Consistent/Total ordering
3. Causal ordering
4. FIFO ordering.

Absolute ordering

- Fig. 5.9.2 shows absolute ordering. In this ordering, all messages are delivered to all receiver processes in the exact order in which they were sent.

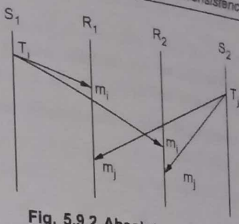


Fig. 5.9.2 Absolute ordering

- **Rule :** mi must be delivered before m_j if $T_i < T_j$
- **Implementation :** A clock synchronized among machines is required. A sliding time window used to commit message delivery whose timestamp is in this window. Window size is properly chosen taking into consideration the maximum possible time that may be required by a message to go from one machine to other machine in the network.
- **Example :** Distributed simulation
- **Drawbacks :**

1. Too strict constraint
2. No absolute synchronized clock
3. No guarantee to catch all tardy messages.

Consistent / Total ordering

- It ensures that all messages are delivered to all receiver processes in the same order.
- Fig. 5.9.3 shows consistent ordering.
- **Rule :** Messages received in the same order, regardless of their timestamp.
- **Implementation :** A message sent to a sequencer, assigned a sequence number, and finally multicast to receivers. A message retrieved in incremental order at a receiver.
- **Example :** Replicated database updates
- **Drawback :** A centralized algorithm.

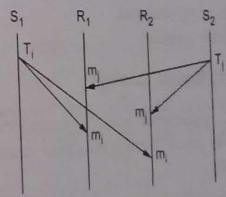


Fig. 5.9.3 Consistent ordering

Causal ordering

- If two message sending events are not causally related, the two messages may be delivered to the receivers in any order.
- Two message sending events are said to be causally related if they are correlated by the happened before relation.
- Fig. 5.9.4 shows the causal ordering.

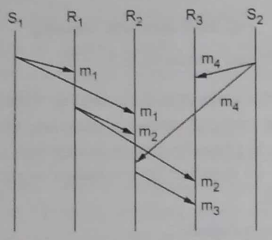


Fig. 5.9.4 Causal ordering

- **Rule : Happened-before** relation
  - If e_i^k, $e_i^l \in h$ and $k < l$, then $e_i^k \in e_i^l$,
  - If e_i = send(m) and e_j = receive(m), then $e_i \in e_j$,
  - If $e \rightarrow e'$ and $e' \rightarrow e''$, then $e \rightarrow e''$
- **Implementation :** Use of a vector message.
- **Example :** Distributed file system.
- **Drawbacks :**

1. Vector as an overhead.
2. Broadcast assumed.

5.10 Distributed Commit

- Some applications perform operations on multiple databases. For example : Transfer funds between two bank accounts or debiting one account and crediting another.
- We would like a guarantee that either all the databases get updated, or none does.
- Distributed commit problem : Operation is committed when all participants can perform it. Once a commit decision is reached, this requirement holds even if some participants fail and later recover.
- Commit protocols are used to ensure atomicity across sites.

- Transaction which executes at multiple sites must either be committed at all the sites, or aborted at all the sites. But it is not acceptable to have a transaction committed at one site and aborted at another.
- The two-phase commit (2PC) protocol is widely used.
- The three-phase commit (3PC) protocol is more complicated and more expensive, but avoids some drawbacks of two-phase commit protocol. This protocol is not used in practice.
- Transaction behave as one operation :

1. **Atomicity :** all-or-none, if transaction failed then no changes apply to the database
2. **Consistency :** there is no violation of the database integrity constraints
3. **Isolation :** partial results are hidden (due to incomplete transactions)
4. **Durability :** the effects of transactions that were committed are permanent.

5.10.1 Atomic Commit Protocols

- Atomicity principle requires that either all the distributed operations of a transaction complete or all abort. At some stage, client executes close Transaction ( ). Now, atomicity requires that either all participants or the coordinator commit or *all* abort.
- In a distributed transaction, the client has requested the operations at more than one server. So, need to ensure safety property in real-life implementation. Never have some agreeing to commit, and others agreeing to abort.

One-phase commit protocol

- The coordinator tells the participants whether to commit or abort. What is the problem with that ?
- This does not allow one of the servers to decide to abort - it may have discovered a deadlock or it may have crashed and been restarted.
- In this protocol, when the client requests a commit, it does not allow a server to make a unilateral decision to abort a transaction. Because commit transaction is related to concurrency control.
- **Advantages of One-phase commit protocol :**

1. Simple protocol fewer overheads.
2. Low latency due to fewer disks writes.
3. Useful in case of low bandwidth networks, as lesser messages are exchanged.
4. In most cases all the updates will be logged before commit so durability is assured.

- **Disadvantages of one-phase commit protocol :**

1. The greatest disadvantage is that it can only handle immediate consistency constraints as there is no voting phase.
2. It can not handle the deferred consistency constraints.

5.10.2 Two Phase Commit Protocols

- Two phase commit (2PC) is the standard protocol for making commit and abort atomic. It is designed to allow any participant to choose to abort a transaction. If one part of the transaction is aborted, then the whole transaction must also be aborted.
- Protocol first phase : Each participant votes for the transactions to be committed or aborted. Once a participant has voted to commit a transaction, it is not allowed to abort it.
- Protocol second phase : Every participant in the transaction carries out the joint decision. If any one participant votes to abort, then the decision must be to abort the transaction. If all the participants vote to commit, then the decision is to commit the transaction.
- The protocol is implemented in two phases :

Phase 1 : Preparation

1. The coordinator sends a PREPARE TO COMMIT message to all subordinates.
2. The subordinates receive the message; write the transaction log, using the write-ahead protocol; and send an acknowledgment (YES/PREPARED TO COMMIT or NO/NOT PREPARED) message to the coordinator.
3. The coordinator makes sure that all nodes are ready to commit, or it aborts the action.

   If all nodes are PREPARED TO COMMIT, the transaction goes to Phase 2. If one or more nodes reply NO or NOT PREPARED, the coordinator broadcasts an ABORT message to all subordinates.

Phase 2 : The Final COMMIT

1. The coordinator broadcasts a COMMIT message to all subordinates and waits for the replies.
2. Each subordinate receives the COMMIT message, and then updates the database using the DO protocol.

3. The subordinates reply with a COMMITTED or NOT COMMITTED message to the coordinator. If one or more subordinates did not commit, the coordinator sends an ABORT message, thereby forcing them to UNDO all changes.

- The objective of the two-phase commit is to ensure that each node commits its part of the transaction; otherwise, the transaction is aborted. If one of the nodes fails to commit, the information necessary to recover the database is in the transaction log, and the database can be recovered with the DO-UNDO-REDO protocol.

Time-out actions in the Two phase commit

- It is to avoid blocking forever when a process crashes or a message is lost. Fig. 5.10.1 shows the comunication in two phase commit protocol.

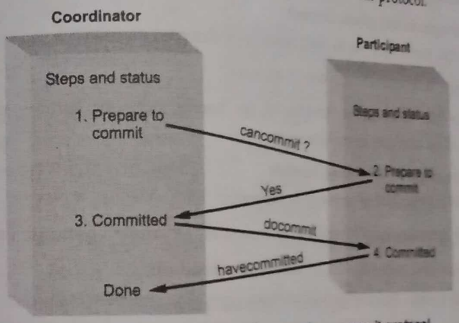


Fig. 5.10.1 Communication in two phase commit protocol

Performance of the two-phase commit protocol

- If there are no failures, the 2PC involving N participants requires *N can Commit ? messages and replies, followed by N doCommit messages.*
- The cost in messages is proportional to 3N, and the cost in time is three rounds of messages. The *haveCommitted messages are not counted.*
- There may be arbitrarily many server and communication failures. 2PC is guaranteed to complete eventually, but it is not possible to specify a time limit within which it will be completed. Two phase commit protocol can cause considerable delays to participants in uncertain state.
- These delays occur when the coordinator has failed and cannot reply to get decision requests from participants.
- Some 3PCs designed to alleviate such delays. They require more messages and more rounds for the normal case.

1. To deal with server crashes : Each participant saves tentative updates into permanent storage, right before replying yes/no in first phase. It is retrievable after crash recovery.
2. To deal with can commit ? loss : The participant may decide to abort unilaterally after a timeout.
3. To deal with Yes/No loss : The coordinator aborts the transaction after a timeout. It must announce *doAbort* to those who sent in their votes.
4. To deal with doCommit loss : The participant may wait for a timeout, send a *getDecision request cannot abort after having voted Yes but before receiving doCommit/doAbort.*

- **Advantages of two phase commit**

1. It ensures atomicity even in the presence of deferred constraints.
2. It ensures independent recovery of all sites.
3. Since it takes place in two-phases, it can handle network failures, disconnections and in their presence assure atomicity.

- **Disadvantages of two phase commit**

1. Involves a great deal of message complexity.
2. Greater communication overheads as compared to simple optimistic protocols.
3. Blocking of site nodes in case of failure of coordinator.
4. Multiple forced writes of log, which increase latency.
5. Its performance is again a trade off, especially for short lived transactions, like internet applications.

5.11 Recovery

- Recovery refers to restoring a system to its normal operational state. Once a failure has occurred, it is essential that the process where the failure happened can recover to a correct state. Fundamental to fault tolerance is the recovery from an error.
- Resources are allocated to executing processes in a computer. For example : A process has memory allocated to it and a process may have locked shared resources, such as files and memory.
- Following are some solution on process recovery :

1. Reclaim resources allocated to process
2. Undo modification made to databases and

3. Restart the process
4. Or restart process from point of failure and resume execution.

- In distributed process recovery, undo effect of interactions of failed process with other cooperating processes.
- System is combination of hardware and software components. These components provide a specified service. Failure of a system occurs when the system does not perform its service in the manner specified. An erroneous state of the system is a state which could lead to a system failure by a sequence of valid state transitions.
- A system is said to "fail" when it cannot meet its promises. A failure is brought about by the existence of "errors" in the system. A system is said to have a failure if the service it delivers to the user deviates from compliance with the system specification for a specified period of time. Fig. 5.11.1 shows concept of fault and recovery.

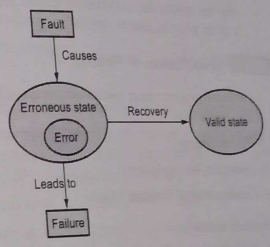


Fig. 5.11.1 Concept of recovery

a. **System failure :** System does not meet requirements, i.e. does not perform its services as specified.

b. **Erroneous system state :** State which could lead to a system failure by a sequence of valid state transitions.

c. **Error :** The part of the system state which differs from its intended value.

d. **Fault :** Anomalous physical condition, e.g. design errors, manufacturing problems, damage, external disturbances.

- A failure occurs when an actual running system deviates from this specified behavior. The cause of a failure is called an **error.** An error represents an invalid system state, one that is not allowed by the system behavior specification.
- The error itself is the result of a defect in the system or fault. In other words, a fault is the root cause of a failure. That means that an error is merely the symptom of a fault. A fault may not necessarily result in an error, but the same fault may result in multiple errors. Similarly, a single error may lead to multiple failures.

- To ensure correctness, recovery mechanisms must be in place to ensure transaction atomicity and durability even in the midst of failures.
- Distributed recovery is more complicated than centralized database recovery because failures can occur at the communication links or a remote site. Ideally, a recovery system should be simple, incur tolerable overhead, maintain system consistency, provide partial operability and avoid global rollback.
- **Reliability** refers to the probability that the system under consideration does not experience any failures in a given time period. Availability refers to the probability that the system can continue its normal execution according to the specification at a given point in time in spite of failures.

5.11.1 Classification of Failures

- Failures in a computer system can be classified as follows :

1. Process failure
2. System failure
3. Secondary storage failure
4. Communication medium failure.

1. Process failure :

- It means that it has halted and will not execute any further.
- In a process failure, the computation results in an incorrect outcome, the process causes the system state to deviate from specifications, the process may fail to progress.
- Behavior : Process causes system state to deviate from specification, for example : incorrect computation, process stop execution.
- Errors causing process failure : Protection violation, deadlocks, timeout, wrong user input, etc...
- Recovery : Abort process or Restart process from prior state.

2. System failure :

- Behavior : Processor fails to execute.
- A system failure occurs when the processor fails to execute. It is caused by software errors and hardware problems.
- Caused by software errors or hardware faults i.e. CPU/memory/bus failure.
- Recovery : System stopped and restarted in correct state.
- Assumption : Fail-stop processors, i.e. system stops execution, internal state is lost.

3. Secondary storage failure :

- A secondary storage failure is said to have occurred when the stored data cannot be accessed. This failure is usually caused by parity error, head crash or dust particles settled on the medium.
- Behavior : Stored data cannot be accessed.
- Errors causing failure : Parity error, head crash, etc.
- Recovery/Design strategies : Reconstruct content from archive plus log of activities and design mirrored disk system.
- A system failure can further be classified as follows.

1. An amnesia failure
2. A partial amnesia failure
3. A pause failure
4. A halting failure

4. Communication medium failure :

- **Behavior :** A site cannot communicate with another operational site.
- A communication medium failure occurs when a site cannot communicate with another operational site in the network. It is usually caused by the failure of the switching nodes and/or the links of the communicating system.
- **Errors/Faults :** Failure of switching nodes or communication links.
- **Recovery/Design strategies :** Reroute, error-resistant communication protocols.

5.11.2 Steps after Failure

- Distributed recovery manager take care of atomicity of global transactions. If distributed DBMS detects that a site failure has occurred, then the following steps are to be followed in recovery process :

1. Failure affected transactions must be aborted.
2. Site failure message is broadcasted to all sites.
3. Checking must be done periodically to see whether the failed site has recovered or not.
4. After restarting the failure site, site must initiate a recovery procedure to abort all partial transactions that were active at the time of failure.

5.11.3 Backward and Forward Recovery

- Once a failure has occurred, it is essential that the process where the failure happened recovers to a correct state.
- Recovery from an error is fundamental to fault tolerance.

- Two main forms of recovery :

1. Forward error recovery
2. Backward error recovery

- **Backward recovery**, by use of checkpointing (global snapshot of distributed system status) to record the system state but checkpointing is costly (performance degradation)

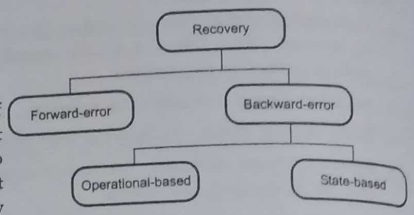


Fig. 5.11.2

Forward error recovery

- Forward recovery, attempt to bring system to a new stable state from which it is possible to proceed (applied in situations where the nature if errors are known and a reset can be applied).
- Forward error recovery continues from an erroneous state by making selective corrections to the system state.
- This includes making safe the controlled environment which may be hazardous or damaged because of the failure.
- It is system specific and depends on accurate predictions of the location and cause of errors (i.e, damage assessment.
- Examples : Redundant pointers in data structures and the use of self-correcting codes such as Hamming Codes.

Advantage Forward - error Recovery :

1. Less overhead.

Disadvantages of Forward Recovery :

1. In order to work, all potential errors need to be accounted for *up-front*.
2. Limited use, i.e. only when impact of faults understood.
3. Cannot be used as general mechanism for error recovery.
4. Design specifically for a particular system.

Backward Recovery

- Most extensively used in distributed systems and generally safest. It can be incorporated into middleware layers.
- Backward recovery is complicated in the case of process, machine or network failure but no guarantee that same fault may occur again.

- It can not be applied to irreversible (non-idempotent) operations, e.g. ATM withdrawall.

Advantages of Backward Recovery :

1. Simple to implement
2. Can be used as general recovery mechanism
3. Capable of providing recovery from arbitrary damage

Disadvantages of Backward Recovery :

1. Checkpointing can be very expensive - especially when errors are very rare
2. Performance penalty
3. No guarantee that fault does not occur again
4. Some components cannot be recovered.

5.11.4 Checkpoint

- **Checkpointing :** The process of writing the current committed values of a server's object to a new recovery file, together with transaction status entries and intentions lists of transactions that have not yet been fully resolved.
- **Checkpoint :** The information stored by the checkpointing process. It is a point of synchronization between database and log file. All buffers are force-written to secondary storage.
- Checkpoint record is created containing identifiers of all active transactions.
- When failure occurs, redo all transactions that committed since the checkpoint and undo all transactions active at time of crash.
- The purpose of make checkpoints reduces the number of transactions to be dealt with during recovery, to reclaim file space.
- When to make checkpoint : Immediately after recovery, or from time to time. After recovery from the checkpoint, discard old recovery file.
- Checkpointing in distributed systems requires that all processes (sites) that interact with one another establish periodic checkpoints.
- All the sites save their local states : **local checkpoints.** All the local checkpoints, one from each site, collectively form a **global checkpoint.**
- The domino effect is caused by orphan messages, which in turn are caused by rollbacks.

Strongly Consistent set of checkpoints

- The most recent distributed snapshot in a system is called the **recovery line.**

- Fig. 5.11.3 shows recovery line and checkpoint.

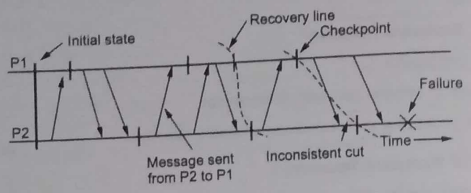


Fig. 5.11.3 Recovery line and checkpoint

- Establish a set of local checkpoints (one for each process in the set) such that no information flow takes place (i.e., no orphan messages) during the interval spanned by the checkpoints.
- A strongly consistent set of checkpoints (recovery line) corresponds to a strongly consistent global state.
- There is one recovery point for each process in the set during the interval spanned by the checkpoints; there is no information flow between any pair of processes in the set and process in the set and any process outside the set.
- A consistent set of checkpoints corresponds to a consistent global state.
- Fig. 5.11.4 shows consistent set of checkpoint.

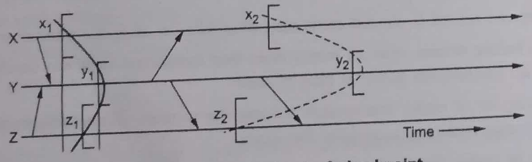


Fig. 5.11.4 Consistent set of checkpoint

- Set $\{x_1, y_1, z_1\}$ is a strongly consistent set of checkpoints.
- Set $\{x_2, y_2, z_2\}$ is a consistent set of checkpoints (need to handle lost messages).
- No local checkpoint includes an effect whose would be undone due to the rollback of another process.

Consistent set of checkpoints

- Similar to the consistent global state.
- Each message that is received in a checkpoint (state) should also be recorded as sent in another checkpoint (state).

- Suppose that Y fails after receiving message 'm'. If Y restarts from checkpoint, message 'm' is lost due to rollback.

Checkpoint notation :

- Each node maintains :

1. Monotonically increasing counter with which each message from that node is labelled.
2. Records of the last message from and the first message to all other nodes.

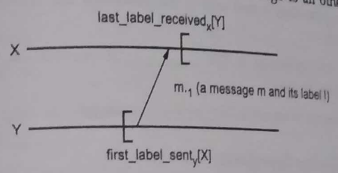


Fig. 5.11.5

Note : "sl" denotes a "smallest label" that is <any other label and "ll" denotes a "largest label" that is > any other label.

Simple method for taking consistent set of checkpoint :

- Every process takes a checkpoint after sending every message.
- The set of most recent checkpoints is always consistent.

5.12 Fill in the Blanks

Q.1 An error is a part of a system's state that may lead to a failure. The cause of an error is called a _____ .

Q.2 _____ model is a contract between a distributed data store and processes.

Q.3 Sequential consistency is an important _____ centric consistency model.

Q.4 _____ refers to the maintenance of copies at multiple sites.

Q.5 Failures are caused by _____ .

Q.6 _____ means that a system can provide its services even in the presence of faults.

Q.7 Failure masking is a fault tolerance technique that hides occurrence of failures from other processes. The most common approach to failure masking is _____ .

Q.8 Consistency protocol describes an implementation of a specific _____ model.

Q.9 _____ redundancy uses three identical modules performing identical operations, with a majority voter determining the output.

Q.10 Push - pull approach is also referred to as _____ protocols.

Q.11 Pull - based approach is efficient when the _____ ratio is relatively low.

Q.12 Coherence models describe what can be expected to only a _____ data item.

Q.13 _____ model describes what can be expected with respect to that set when multiple processes concurrently operate on that data.

Q.14 Reliability is defined in terms of a time interval instead of an _____ in time.

Q.15 A system is synchronous if and only if the processes are known to operate in a _____ mode.

Q.16 Goal of Byzantine agreement is that _____ is reached on the value for the non - faulty processes only.

Q.17 Two - phase commit is known as a _____ protocol.

5.13 Multiple Choice Questions

Q.1 _____ failure occurs when a server fails to respond to a request.

a Omission b Timing
c Response d Arbitrary

Q.2 A problem with the _____ commit protocol is that when the coordinator has crashed, participants may not be able to reach a final decision.

a three-phase b two-phase
c checkpoint d none

Q.3 _____ redundancy add extra bits to allow recovery from garbled bits.

a Physical b Time
c Information d All of these

Q.4 Which of the following is the client - centric consistency model ?

a Monotonic reads b Monotonic writes
c Eventual consistency d All of these

Q.5 TMR stands for _____ .

a Time Model Receiver
b Transmission Modular Redundancy
c Triple Modular Redundancy
d Two Modular Redundancy

Q.6 _____ redundancy adds extra equipment or processes so that the system can tolerate the loss or malfunctioning of some components.

a Physical b Time
c Information d Sequential

Q.7 If local states jointly do not form a distributed snapshot, further rollback is necessary. This process of a cascaded rollback may lead to what is called the _____ .

a checkpointing b recovery line
c domino effect d independent checkpointing

Q.8 As processes take local checkpoints independent of each other, this method is also referred to as _____ .

a coordinated checkpointing b independent checkpointing
c distributed snapshot d incremental snapshot

Q.9 List the name of techniques that avoid domino effect :

a Coordinated checkpointing rollback recover
b Log - based rollback recovery
c Communication - induced checkpointing rollback recovery
d All of these

Q.10 What is an orphan messages ?

a Messages that have been sent but not yet received
b Messages whose send is done but receive is undone due to rollback
c Messages with receive recorded but message send not recorded
d Arise due to message logging and replaying during process recovery

Q.11 _____ protocols assume that a failure can occur after any non - deterministic event in the computation.

a Optimistic logging b Pessimistic logging
c Causal logging d All of these

Q.12 In _____ checkpointing, each process takes its checkpoints independently.

a communication - induced b coordinated
c uncoordinated d none

Q.13 The most recent consistent global checkpoint is termed as the _____ .

a domino effect b recovery line

c coordinated checkpoint d all of these

Q.14 The checkpoints that a process takes independently are _____ checkpoints while those that a process is forced to take are called forced checkpoints.

a global b communication-induced

c local d uncoordinated

Answer Keys for Fill in the Blanks

Q.1	fault	Q.2	Consistency
Q.3	data	Q.4	Replication
Q.5	faults	Q.6	Fault tolerance
Q.7	redundancy	Q.8	consistency
Q.9	Triple modular	Q.10	server - based
Q.11	read-to-update	Q.12	single
Q.13	Consistency	Q.14	instant
Q.15	lock-step	Q.16	consensus
Q.17	blocking - commit		

Answer Keys for Multiple Choice Questions

Q.1	a	Q.2	b
Q.3	c	Q.4	d
Q.5	c	Q.6	a
Q.7	c	Q.8	b
Q.9	d	Q.10	c
Q.11	b	Q.12	c
Q.13	b	Q.14	c

❑❑❑

6 Security

Syllabus

Introduction to Security - Security Threats, Policies, and Mechanisms, Design Issues, Basics of Cryptography, Secure Channels- Authentication, Message Integrity and Confidentiality, Secure Group Communication; Access Control - General Issues in Access Control, Firewalls, Secure Mobile Code, Denial of Service; Security Management - Key Management, Secure Group Management, Authorization Management.

Contents

6.1 Introduction to Security

Security threats, Policies and Mechanism

- Now, with the information era at hand, the need is more pronounced than ever. As the world becomes more connected, the demand for information is growing and with increased demand comes increased dependency on electronic systems.
- Networks of computers became more common; so too did the need to interconnect networks. Internet became first manifestation of a global network of networks.
- The Internet brings millions of computer networks into communication with each other and many of them unsecured. Ability to secure a computer's data influenced by the security of every computer to which it is connected.
- A successful organization should have multiple layers of security in place :
  a. Physical security
  b. Personal security
  c. Operations security
  d. Communications security
  e. Network security
  f. Information security
- Security is required because the widespread use of data processing equipment, the security of information felt to be valuable to an organization was provided primarily by physical and administrative means. Network security measures are needed to protect data during their transmission.
- Data security is the science and study of methods of protecting data from unauthorized disclosure and modification.
- Information security, to protect the confidentiality, integrity and availability of information assets, whether in storage, processing, or transmission. It is achieved via the application of policy, education, training and awareness and technology.
- Fig. 6.1.1 shows components of information security.

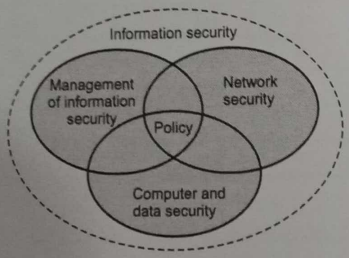


Fig. 6.1.1 Components of IS

- Vulnerability refers to the security flaws in a system that allows an attack to be successful.

- Threat refers to the source and means of a particular type of attack. A threat assessment is performed to determine the best approaches to securing a system against a particular threat, or class of threat.
- Risk is a function of threats exploiting vulnerabilities to obtain damage or destroy assets. Thus, threats may exist, but if there are no vulnerabilities then there is little/no risk.
- Control is used as proactive measure. Control is an action, device, procedure or technique that removes or reduces a vulnerability. A threat is blocked by control of vulnerability.
- The Committee on National Security Systems (CNSS) defines information security as the protection of information. The CNSS model of information security evolved from a concept developed by the computer security industry called the CIA (Confidentially, Integrity, Availability) triangle.
- The CIA triangle has been the industry standard for computer security since the development of the mainframe. Goals or principles of information security are as follows :
  1. Confidentially 2. Integrity 3. Availability
- Fig. 6.1.2 shows pillars of information security.

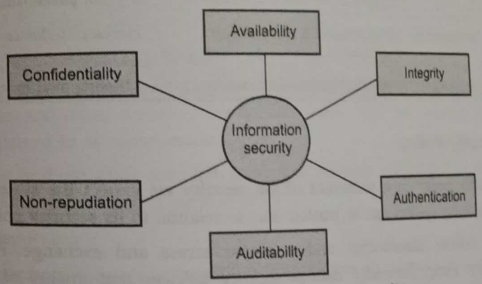


Fig. 6.1.2 Pillars of information security

- **Confidentiality** means prevention of unauthorized disclosure of information. Main mechanisms of protection of confidentiality in information systems are cryptography and access controls. Malware, intruders, social engineering, insecure networks are threats to confidentiality.
- **Integrity** is a prevention of unauthorized modification of information. Integrity in terms of information security refers not only to integrity of information itself but also to the origin integrity i.e. integrity of the source of information.

- **Availability** is a prevention of unauthorized withholding of information or resources. Attacks against availability are known as denial-of-service attacks. Unavailability of information is just as harmful to an organization as a lack of confidentiality or integrity.

Security Threats :

1. **Interception :** An unauthorized party has gained access to a service or data.
2. **Interruption :** Services or data become unavailable, unusable, destroyed and so on.
3. **Modification :** Unauthorized changing of data or tampering with a service so that it no longer adheres to its original specifications.
4. **Fabrication :** Refers to the situation in which additional data or activity are generated that would normally not exist.

Examples of Threats :

Sr. No.	Threat	Definition	Example
1.	Interruption	An asset becomes unusable, unavailable or lost.	Denial of service attack on a website.
2.	Interception	An unauthorized party gains access to an information asset.	Compromise of confidential data, e.g. but packet sniffing.
3.	Modification	An authorized party tampers with an asset.	Hacking to deface a website.
4.	Fabrication	An asset has been counterfeit.	Spoofing attacks in a network.

6.1.1 Security Policy

- A security policy is a statement of the security we expect the system to enforce. An operating system can be trusted only in relation to its security policy.
- Policy defines classification and rules for access and exchange. Policy defines criticality. Policy hierarchy defines security services and quality of mechanisms. Security policy is different from security procedure and processes.
- The purpose of the Information Security Policy is to provide management with direction and support regarding information security in accordance with business requirements and relevant laws and regulations and to outline the responsibilities of management and staff with respect to information security.
- A security policy is technology and vendor independent. Security policies are not easy to create. A security policy indicates company senior management's commitment to maintaining a secure network, which allows the staff to do a more

effective job of securing the company's information assets. Policy will reduce your risk of a damaging security incident.

- Information security policy provides management direction and support for information security across the organization, in both electronic and hard copy.
- Specific, subsidiary information security policies are considered part of this information security policy and have equal standing.

6.1.2 Security Mechanism

- Security mechanism is a mechanism that is designed to detect, prevent, or recover from a security attack. Security mechanisms are used to implement security services.
- Important security mechanism are as follows :
  1) Encryption : Transform the data into something an attacker cannot understand
  2) Authentication : Used to verify the claimed identity of a user.
  3) Authorization : Checking if the authenticated user has the right to perform the action requested.
  4) Auditing : Are set of tools used to trace which clients accessed what, and which way
- Matching security mechanisms to threats is only possible when a Policy on security and security issues exists.

Example : The Globus Security Architecture

- Globus is a system supporting large scale distributed computations in which many hosts, files and other resources are simultaneously used for doing a computation. It also referred to as computational grids.
- Resources in these grids are often located in different administrative domains that may be located in different parts of the world.

Globus Security Policy :

1. The environment consists of multiple administrative domains.
2. Local operations (i.e., operations that are carried out only within a single domain) are subject to a local domain security policy only.
3. Global operations require the initiator to be known in each domain where the operation is carried out.
4. Operations between entities in different domains require mutual authentication.
5. Global authentication replaces local authentication.
6. Controlling access to resources is subject to local security only.

7. Users can delegate rights to processes.
8. A group of processes in the same domain can share credentials.

- The Globus security architecture consists of entities such as : users, user proxies, resource proxies and general processes. Entities are located in domains and interact with each other.
- Fig. 6.1.3 shows Globus security architecture.

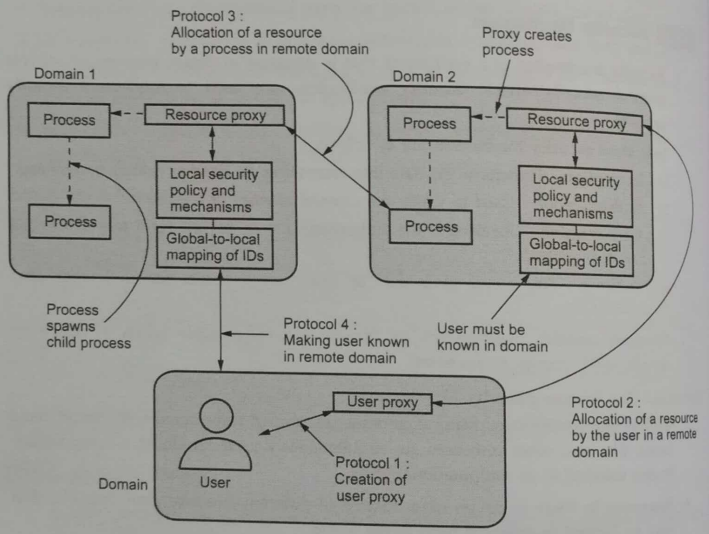


Fig. 6.1.3 : Globus security architecture

- Globus needs mechanism for cross domain authentication and making a user known in remote domains. **User proxy** is a process that is given permission to act on behalf of a user for a limited period of time.
- Resources are represented by resource proxies. **Resource proxy** is a process running within a specific domain that is used to translate global operation on a resource into local operations that comply with that particular domain security policies.
- User proxy typically communicates with a resource proxy when access to that resource is required.

6.1.3 Design Issues

- Three design issues when considering security :
  1. Focus of Control.
  2. Layering of Security Mechanisms.
  3. Simplicity.
- **Focus of Control :** Three approaches for protection against security threats.
  a) Protection against invalid operations
  b) Protection against unauthorized invocations.
  c) Protection against unauthorized users.
- **Layering of Security Mechanisms :** Decision required as to where the security mechanism is to be placed.
- A system is either secure or it is not. Whether a client considers a system to be secure is a matter of trust. Layer in which security mechanisms are placed depends on the trust a client has in how secure the services are in any particular layer.
- The Trusted Computing Base (TCB) is the set of services/mechanisms within a distributed system required to support a security policy.

Simplicity

- Designing a secure computer system is difficult, regardless of whether it is also a DS. Using a few simple mechanisms that are easily understood and trusted to work is the ideal situation.
- In real world, introducing security mechanisms to an already complex system can make matters worse. However, this is still a design goal to aim for.

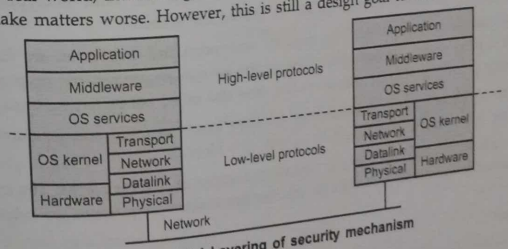


Fig. 6.1.4 Layering of security mechanism

6.2 Basics of Cryptography

- The basic terminology is that cryptography refers to the science and art of designing ciphers; cryptanalysis to the science and art of breaking them.

- The input to an encryption process is commonly called the plaintext and the output the ciphertext.
- The objective is to design an encryption technique so that it would be very difficult or impossible for an unauthorized party to understand the contents of the ciphertext.
- A user can recover the original message only by decrypting the ciphertext using the secret key.
- Fig. 6.2.1 shows cryptography.

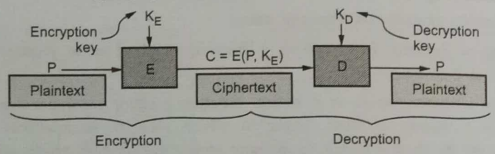


Fig. 6.2.1

- Depending upon the secret key used, the algorithm will produce a different output. If the secret key changes, the output of the algorithm also changes.

Characteristics of Cryptography :

1. The type of operations used for transforming plaintext to ciphertext.
2. The number of keys used.
3. The way in which the plaintext is processed.

- **Symmetric cryptosystem :** The same key is used for both encryption and decryption. It is also called secret kay or shared key system.
- **Asymmetric cryptosystem :** The keys for encryption and decryption are different, but together form a unique pair. One of the keys in an asymmetric cryptosystem is kept private, the other is made public. It is also called public key systems.
- **Hash function** takes a message m of arbitrary length as input and produces a bit string "h" having a fixed length as output : h = H(m).
- For a hash function, the input is viewed as a sequence of n-bit blocks. The input is processed one block at a time in an iterative fashion to produce an n-bit hash function.

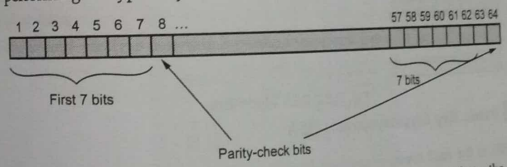
Properties :

1. H can be applied to a block of data of any size.
2. H produces a fixed length output.

3. H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical.
4. For any given value h, it is computationally infeasible to find x such that H(x) = h. This is called one-way property.
5. For any given block x, it is computationally infeasible to find $y \neq x$ such that H(y) = H(x). This is called as weak collision resistance.
6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y). This is called as strong collision resistance.

6.2.1 Symmetric Cryptosystems : DES

- Data Encryption Standard is a block cipher and operates on 64-bit blocks of data, using a 56-bit key. It is a 'private key' system. DES expects two inputs : the plaintext to be encrypted and the secret key.
- Once a plain-text message is received to be encrypted, it is arranged into 64 bit blocks required for input. If the number of bits in the message is not evenly divisible by 64, then the last block will be padded. Multiple permutations and substitutions are incorporated throughout in order to increase the difficulty of performing a cryptanalysis on the cipher.



- In the DES specification, the key length is 64 bit : 8 bytes; in each byte, the 8th bit is a parity-check bit.
- Each parity-check bit is the XOR of the previous 7 bits. Fig 6.2.2 shows DES algorithm.
- DES performs an initial permutation on the entire 64 bit block of data. It is then split into two, 32 bit sub-blocks, Li and Ri which are then passed into what is known as a round.
- At the end of the 16th round, the 32 bit Li and Ri output quantities are swapped to create what is known as the pre-output. This [R16, L16] concatenation is permuted using a function which is the exact inverse of the initial permutation. The output of this final permutation is the 64 bit ciphertext.

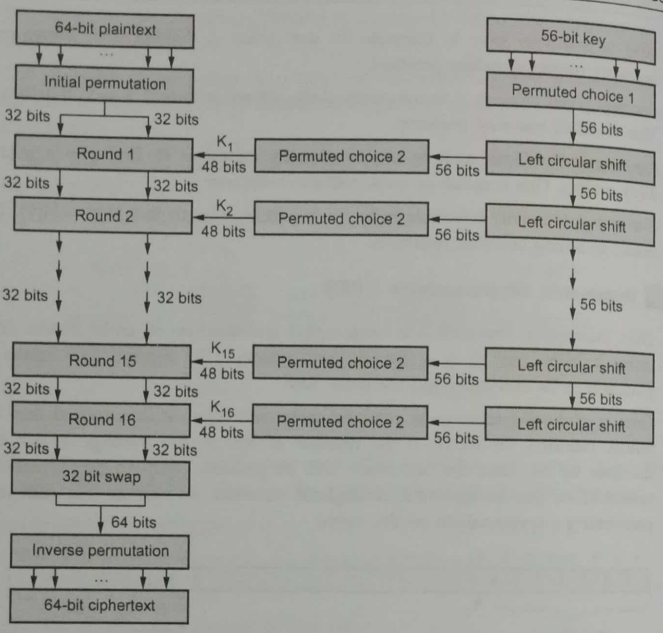


Fig. 6.2.2 DES algorithm

6.2.2 Public Key Cryptosystems : RSA

- RSA is the most common and well-known public key cryptosystem. It is invented by Rivest, Shamir & Adleman of MIT in 1977. RSA is a block cipher.
- RSA uses two key pair : public key (e) and private key (d).
- Key pair (e,d) contains two keys : e is the public key (used to encrypt documents) and d is the private key (used to decrypt documents)
  - Encryption and decryption process uses modular exponentiation. Modular exponentiation is a type of exponentiation performed over a modulus.
  - It is easy to multiple two prime number together but extremely difficult to factor them back from the result. Factoring a number means finding its prime factors.
  - Modular exponentiation : Given n, m, and e, it is easy to compute $c = m_e$ **mod n.**

- The value m_e **mod n** is formally the result of multiplying e copies of m, dividing by n, and keeping the remainder. The public key in this cryptosystem consists of the value n, which is called the modulus, and the value e, which is called the public exponent. The private key consists of the modulus n and the value d, which is called the private exponent.
- The relation between e, d and n is as follows :
  1. The pair of numbers (e, N) is known as the public key and can be published.
  2. The pair of numbers (d, N) is known as the private key and must be kept secret.
- Anybody knowing the public key can use it to create encrypted messages, but only the owner of the secret key can decrypt them.
- Security of RSA depends on the difficulty of factorizing N. Because factorization is believed to be a hard problem, the longer N is, the more secure the cryptosystem.
- The problem with choosing long keys is that RSA is very slow compared with a symmetric block cipher such as DES, and the longer the key the slower it is. The best solution is to use RSA for digital signatures and for protecting DES keys.

Working of RSA

- Sender and receiver generate a public and private key.

Symbols	Description
c	Cipher text
M or P	Message/Plain text
e	Public key
d	Private key
p and q	Prime number

1. Key generation

a. Select two distinct prime number : **p and q**

b. Find n such that **n = p x q.**
(The n will be used as the modulus for both the public and private keys)

c. Find the totient of n, ϕ **(n) = (p – 1) x (q – 1)**

d. Choose an e such that **1 < e < ϕ (n),** and such that e and ϕ (n) share no divisors other than 1 (e and φ (n) are relatively prime). Here e is kept as the public key exponent.

e. Determine d which satisfies the congruence relation $\mathbf{d \times e \equiv 1 \pmod{\phi(n)}}$

2. Encryption : Calculate cipher text (c)

$$c \equiv m^e \pmod{n}$$

3. Decryption $m \equiv c^d \pmod{n}$

6.2.3 Hash Function : MD5

- The MD5 algorithm (Message Digest 5) is a cryptographic message digest algorithm. Cryptographic hash functions take data input (or massage) and generate a fixed size result (or digest). It is a hashing function developed by Ron Rivest.
- MD5 creates a 128-bit message digest from the data input which is typically expressed in 32 digits hexadecimal number. MD5 hashes are unique for different inputs regardless of the size of the input.
- The MD5 algorithm is intended for digital signature applications, where a large file must be "compressed" in a secure manner before being encrypted with a private key under a public-key cryptosystem such as RSA or PGP.
- MD5 consists of two phases : **Padding phase and Compression phase.**
- In the padding phase, some extra bits (1 to 512 bits) are appended to the input message. The result bits are congruent to 448 mod 512. Then the length of the initial message is transformed to a 64-bit binary-string and 64 bits is added to the tail of the message too. So the padding phase ends with a bit stream that consists of one or more 512-bit blocks.
- In the compression phase, a compression function is used on each 512-bit block and generates a 128-bit output. The output is always involved in the calculation of next round.

6.3 Secure Channels

- Secure communication requires authentication of the communicating parties, but also ensuring message integrity and possibly confidentiality as well.
- A secure channel protects senders and receivers against interception, modification, and fabrication of messages. It does not necessarily protect against interruption.
- Protecting messages against interception are done by ensuring confidentiality. Protecting messages against modification and fabrication is done through protocols for mutual authentication and message integrity.

- Fig 6.3.1 shows secure channel.

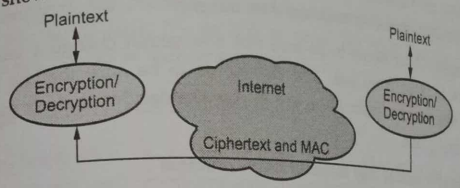
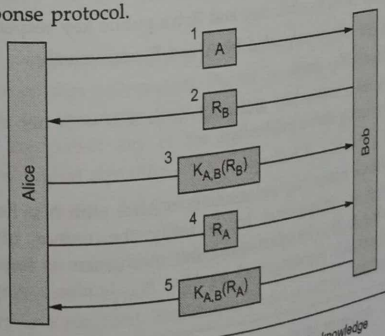


Fig. 6.3.1 Secure channel

- A stream with these security requirements :
  1. Authentication : Ensures sender and receiver are who they claim to be.
  2. Confidentiality : Ensures that data is read only by authorized users
  3. Data integrity : Ensures that data is not changed from source to destination
  4. Non-repudiation : Ensures that sender cannot deny message and receiver cannot deny message.

6.3.1 Authentication

- Authentication and message integrity cannot do without each other. The combination works as follows:
  a. Alice starts by sending a message to Bob to set up a channel.
  b. Once the channel has been set up, Alice knows for sure that she is talking to Bob and Bob knows for sure that he is talking to Alice, they can exchange messages.
  c. To subsequently ensure integrity it is common practice to use secret-key cryptography by means of session keys.
- Authentication based on a shared secret key. It is also known as challenge-response protocol.



- In addition to authentication, a secure channel also requires that messages are confidential, and that they maintain their integrity.
- **For example :** Alice needs to be sure that Bob cannot change a received message and claim it came from her. And Bob needs to be sure that he can prove the message was sent by/from Alice, just in case she decides to deny ever having sent it in the first place.

Needham Schroeder public key authentication protocol :

- The Needham Schroeder public key authentication protocol aims to provide a mutual authentication between two parties Alice (A) and Bob (B).
- Both parties want to insure each other identity before starting to communicate.
- Fig. 6.3.2 shows Needham Schroeder authentication protocol.

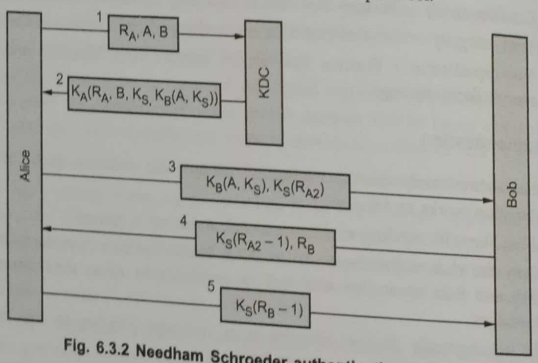


Fig. 6.3.2 Needham Schroeder authentication protocol

- The protocol is as follows :

a. K_A and K_B are Alice's public key and Bob's public key respectively,

b. N_A and N_B are nonce generated by A and B respectively.

1. $A \rightarrow B : \{N_A, A\}K_B$ (Init)

   Alice generates a nonce N_A and sends it to Bob with her identity. Everything is encrypted using Bob's public key.

2. $B \rightarrow A : \{N_A, N_B\}K_A$ (Challenge)

   Bob generates a nonce N_B and sends it to Alice with N_A he has just received. It is a way to prove that he is really the owner of the private key corresponding to K_B. In other word, this mechanism is implemented in order to authenticate Bob. Sending back to Alice N_A is also a way to avoid a reply of this message.

3. $A \rightarrow A : \{N_B\}K_B$ (Response)

   Alice decrypts the message and check if it contains the right value of N_A. Then, she sends back N_B to Bob to prove her ability to decrypt with her private key and so to authenticate herself.

6.3.2 Message Integrity and Confidentiality

- Besides authentication, a secure channel should also provide guarantees for message integrity and confidentiality. Confidentiality is easily established by simply encrypting a message before sending it.
- Protecting a message against modifications is somewhat more complicated.
- During the establishment of a secure channel, after the authentication phase has completed, the communicating parties generally use a unique shared session key for confidentiality. The session key is safely discarded when the channel is no longer used.
- Why not use the same keys for confidentiality as those that are used for setting up the secure channel ? Cryptographic keys are subject to wear and tear. They are just like ordinary keys. If a key is compromised, only a single session is affected.
- The combination of long-lasting keys with the much cheaper and more temporary session keys is often a good choice for implementing secure channels for exchanging data.

Digital Signature

- Digital signature same as a persons signature on a document. A digital signature on a message is required for the authentication and identification of the right sender. The digital signature is supposed to be unique to an individual and serves as a means of identifying the sender.
- Any public key cipher can be used for digital signature. Digital Signature Standard (DSS) is a digital signature format that has been standardized by NIST.
- DSS signature may use any one of three public key ciphers: RSA, ElGamal and Elliptic curve
- Hash value of a message when encrypted with the private key of a person is his digital signature on that e-Document. Digital Signature of a person therefore varies from document to document thus ensuring authenticity of each word of that document. As the public key of the signer is known, anybody can verify the message and the digital signature.
- Each individual generates his own key pair. Public key known to everyone & private key only to the owner. Private key is used for making digital signature. Public key is used to verify the digital signature. The originator of a message uses a signing key (Private Key) to sign the message and send the message and its digital signature to a recipient. The recipient uses a verification key (Public Key) to

verify the origin of the message and that it has not been tampered with while in transit.

- Digital signature uses three algorithms:
  1. Key generation : This algorithm selects a private key uniformly at random from a set of possible private keys. Output of this algorithm is private key and its corresponding public key.
  2. Signing Algorithm : It produce signature by using message and private key.
  3. Signature verifying algorithm : For a given message, signature and public key, either accepts or rejects the messages claim to authenticity.
- Fig 6.3.3 shows digital signature.

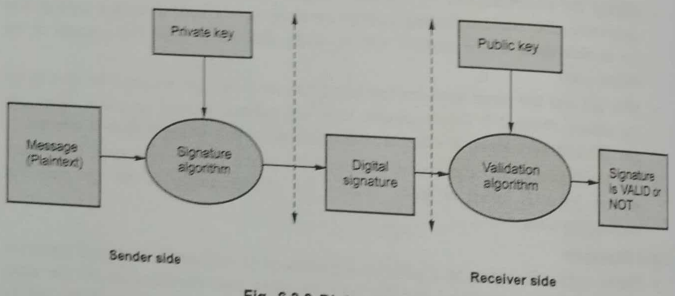


Fig. 6.3.3 Digital signature

- **Digital signature creation** uses a hash result derived from and unique to both the signed message and a given private key. For the hash result to be secure there must be only a negligible possibility that the same digital signature could be created by the combination of any other message or private key. A hash value consists of a small amount of binary data, typically around 160 bits.
- Digital signature verification is the process of checking the digital signature by reference to the original message and a given public key, thereby determining whether the digital signature was created for that same message using the private key that corresponds to the referenced public key.
- When an individual receives a signed document or transaction, he will initiate the verification process. The public key of the sender is used to decrypt the digital signature and retrieve the message digest. The hash algorithm is applied again to the digital contents to generate another message digest. These two message digests are compared and if they match verification is successful. If there were any changes in the digital contents the resultant message digest would differ from the original one and the verification would fail.

6.3.3 Secure Group Communication

- In distributed applications like groupware, group communication among multiple entities is required. The local area networks (LANs) and radio networks provide broadcast communication at the media access control (MAC) layer, that is every entity can receive every protocol data unit (PDU) transmitted in the network. Group communication among multiple entities can be easily realized by these networks. One problem in the broadcast network is how to provide secure communication for the group.
- The secure group communication abstraction provides both point-to-point communication and secure multipoint communication. Point-to-point messages are encrypted with a key shared by the two ends. Multicast messages are encrypted with the group key.
- One important property of secure group communications is the assurance that only the legitimate members (or users, receivers) can have access to the multi-cast or broadcast data.

1. Confidential group communication

- To ensure confidentiality, a simple scheme of letting all group members to share same secret key. This key is used to encrypt and decrypt all the messages transmitted by the members. All the members need to be trusted to keep the key a secret.
- This pre-requisite alone makes the use of single key more vulnerable to attacks.
- Another solution is to maintain separate shared secret key between each pair of group members. When one attack happens others can stop sending the messages but still use their secret keys.
- However, instead of maintaining one key, it is necessary to maintain $N(N-1)/2$ keys which is a difficult problem.
- Using public-key cryptosystem the situation can be improved. Each member has its own (private key, public key) pair, in which public key can be used by all members for sending confidential messages.
- The N key pairs are need for N members. Unfaithful members can be removed from group without compromising the other keys.

2. Secure Replicated Servers

- Given a securely replicated group of servers, each server accompanies its response with a digital signature.
- If "ri" is the response from server "Si" then md(ri) denote the message digest computed by server Si. This digest is signed with server Si's private key .

- It we want to protect the client against at most corrupted servers, the server group should be able to tolerate corruption by at most "c" servers, and still be capable of producing a response that the client can put its trust in.
- Let the replicated servers generate a secret valid signature with the property that c corrupted servers alone are not enough to produce that signature.
- Consider a group of five replicated servers that should be able to tolerate two corrupted servers, and still produce a response that a client can trust.
- Let N = 5, c = 2.
- There are 5! / (3!2!) = 10 possible combinations of three signatures that the client can use as input for decryption function (D). If one of these combinations produces a correct digest md (ri) for some response "ri" then the client can consider "ri" as being correct. It can trust that the response has been produced by at least three honest servers.

6.3.4 Kerberos

- Kerberos is a network authentication protocol. It is designed to provide strong authentication for client/server applications by using secret-key cryptography.
- Kerberos was developed at M.I.T. and is based on the Needham-Schroeder authentication protocol. Two different versions of Kerberos - version 4 (V4) and version 5 (V5).
- Kerberos V5 being more flexible and scalable.
- In the Kerberos system, a trusted third-party issues session keys for interactions between users and services.
- Kerberos client/server authentication requirements are :
  1. Security : That Kerberos is strong enough to stop potential eavesdroppers from finding it to be a weak link.
  2. Reliability : That Kerberos is highly reliable employing a distributed server architecture where one server is able to back up another. This means that Kerberos systems are fail safe, meaning graceful degradation, if it happens.
  3. Transparency : That user is not aware that authentication is taking place beyond providing passwords.
  4. Scalability : Kerberos systems accept and support new clients and servers.
- Kerberos uses two different components :
  1. **Authentication Server** (AS) is responsible for handling a login request from a user. AS authenticates a user and provides a key that can be used to set up secure channels with servers.

2. **Ticket Granting Service** (TGS) sets up secure channels. The TGS hands out special messages, known as tickets, that are used to convince a server that the client is really who he or she claims to be.

- Client machine asks for the password. When the correct password is entered, then the key is ready to use. A requests for the connection with client to TGS. A secure channel is established.

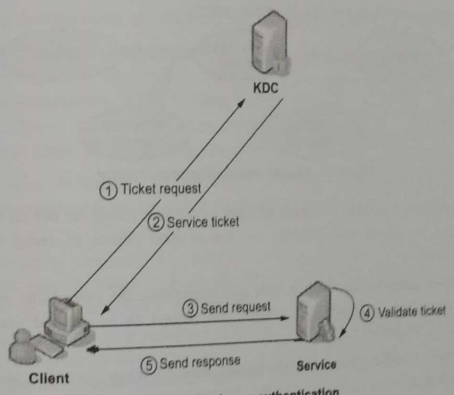


Fig. 6.3.4 Kerberos authentication

6.4 Access Control

- Access rights are referred to as access control, whereas authorization is about granting access rights.
- The access control mechanism refers to prevention of unauthorized use of a resource.
- Access control includes :
  1. Authentication of users
  2. Authorization of their privileges
  3. Auditing to monitor and record user actions

Authorization versus Authentication

- **Authentication :** Verify the claim that a subject says it is S : Verifying the identity of a subject. Authentication techniques are used to verify identity. The authentication of authorized users prevents unauthorized users from gaining access to corporate information systems.

- **Authorization :** Determining whether a subject is permitted certain services from an object. Authorization is a procedure of controlling the access of authenticated users to the system resources. An authorization system provides each user with exactly those rights granted to them by the administrator.

6.4.1 General Issues in Access Control

- Fig 6.4.1 shows simple model of access control.

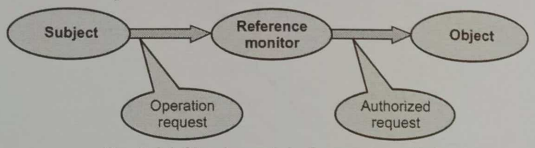


Fig. 6.4.1 Simple model of access control

- Subjects issue a request to access an object. Processes acting on behalf of users, but can also be objects that need the services of other objects in order to carry out their work.
- An object encapsulates its own state and implements the operations on that state. Operations of an object that subjects can request to be carried out are made available through interfaces.
- Reference monitor records which subject may do what, and decides whether a subject is allowed to have a specific operation carried out. This monitor is called each time an object is invoked.

6.4.1.1 Access Control Matrix

- Access control matrix is easy to implement. Access matrix contains row and columns. Row is represented by domains and columns represents by objects. Each entry in the matrix consists of a set of access rights.
- The intersection of the row and column contains the access rights for that subject to that objects. Following table shows access control matrix/ protection matrix.
- Each cell in the matrix specifies the actions that a subject can perform on an object.

	Sorting.c	Payroll.c	Abc.txt	Maths.xls	Sal.doc	Pay.xls	Printer
Domain1	Read, Write	Read, write, execute					Write
Domain2			Read	Read Write Execute			Write

	Read	write	Read Write Execute
Domain3			

- Access matrix works well only for systems with a few files and few users.

6.4.1.2 Protection Domains

- Protection mechanisms are implemented in operating system to support various security policies.
- Computer system consists of hardware and software resources. Hardware resources/components are CPU, main memory segment, keyboard, mouse, printer and disk drives. Software resources/components are processes, files, database and semaphore. These all components are called as object.
- Unique name is assigned to each object. This name is referenced whenever any operation performed on the object. Read and write operation is performed on the files.
- Objects must be protected from subjects. **Access rights** define how various subjects can access various objects. Subjects may be users, processes, programs etc. subjects are active entities and objects are passive.
- **Protection domain** is a collection of access rights. The protection state can be conceptualized as an access matrix. Each access rights in a protection domain are represented as an order pair with fields for the object name and its corresponding privileges.
- Protection domains are extensions of the hardware supervisor mode ability.
- Protection domain is unique to a subject. A domain is a set of (object, rights) pairs. Each pair specifies an object and some subset of the operations that can be performed on it.
- For example, if a user can read and write the file "sorting.c", the corresponding ordered pair for this user's access right can be represented by <sorting.c, {read, write}>.
- Fig. 6.4.2 shows three protection domains. The association between domain and process is static or dynamic. Static mean the set of available resource to process is fixed through process lifetime.

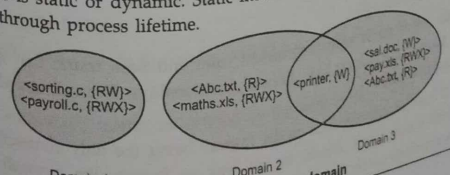


Fig. 6.4.2 Protection domain

- At every instant of time, each process runs in some protection domain. Processes can also move from one domain to other domain during execution. The rules for domain switching are highly system dependent.
- Access rights may be copied, transferred or propagated from one domain to another. Coping of access rights simply granting a right of one user to another user. When a subject no longer needs access to an object, access rights can be revoked.
- In UNIX operating system, process domain is defined by using UID and GID. Given any UID and GID combination, it is possible to make a complete list of all objects that can be accessed. Two subjects with the same UID and GID combination will have access to exactly same set of objects.
- Each file management system has its own method to control file access. The four most commonly used methods are as follows :

1. Access control matrix
2. Access control lists
3. Capability lists
4. Lockword control

6.4.2 Firewalls

- A firewall is a device designed to control the flow of traffic into and out-of a network. In general, firewalls are installed to prevent attacks. Firewall can be software program or a hardware device.
- Firewalls are software programs or hardware devices that filter the traffic that flows into user PC or user network through a internet connection. They shift through the data flow and block that which they seem harmful to user network or computer system.
- Firewalls filter based on IP, UDP and TCP information. Firewall is placed on the link between a network router and internet or between a user and router. For large organization with many small networks, the firewall is placed on every connection attached to the internet.
- Large organizations may use multiple levels of firewall or distributed firewalls, locating a firewall at a single access point to the network.
- Fig. 6.4.3 shows firewall.
- Firewalls test all traffic against consistent rules and pass traffic that meets those rules. Many router support basic firewall functionality. Firewall can also be used to control data traffic.
- Firewall based security depends on the firewall being the only connectivity to the size from outside; there should be no way to bypass the firewall via other gateways; wireless connections.

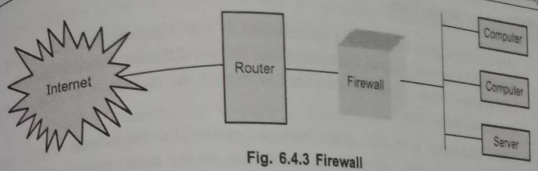


Fig. 6.4.3 Firewall

- Firewall filter out all incoming messages addressed to a particular IP address or a particular TCP port number. It divides a network into a more trusted zone internal to the firewall and a less trusted zone external to the firewall.
- Firewall may also impose restrictions on outgoing traffic, to prevent certain attacks and to limit losses if an attacker succeeds in getting access inside the firewall.
- Functions of firewall :
  1. **Access control :** Firewall filters incoming as well as outgoing packets.
  2. **Address/Port translation :** Using network address translation, internal machines, though not visible on the internet, can establish a connection with external machines on the internet. NATing is often done by firewall.
  3. **Logging :** Security architecture ensures that each incoming or outgoing packets encounters at least one firewall. The firewall can log all anomalous packets.
- Firewalls can protect the computer and user personal information from :
  1. Hackers who breaks your system security.
  2. Firewall prevents malware and other internet hacker attacks from reaching your computer in the first place.
  3. Outgoing traffic from your computer created by a virus infection.
- Firewalls cannot provide protection :
  1. Against phishing scams and other fraudulent activity.
  2. Viruses spread through e-mail.
  3. From physical access of your computer or network.
  4. For an unprotected wireless network.

Firewall Characteristics

1. All traffic from inside to outside, and vice versa, must pass through the firewall.
2. The firewall itself is resistant to penetration
3. Only authorized traffic, as defined by the local security policy, will be allowed to pass.

6.4.3 Secure Mobile Code

- Mobile code is great for balancing communication and computation, but is hard to implement a general-purpose mechanism that allows different security policies for local-resource access. Also, we may need to protect the mobile code (e.g., agents) against malicious hosts.
- When protecting a mobile code from a potentially malicious host, code mobility implies that the program will be run under total control of the host. This means the following threats :
  1. Spoofing through impersonation of code owner
  2. Theft and secrecy violation through unauthorized disclosure
  3. Integrity violation through subversion of code semantics
- To prevent all three cases, data segments as well as code semantics must be protected.

Protecting an Agent :

- Most agent systems refer to four elements for their security:
  1. A runtime environment (usually the Java Virtual Machine) for host protection;
  2. Code signing to prove that the agent has not been tampered with;
  3. Host authentication to prove that the agent is about to move to the intended host,
  4. Establishment of a secure (cryptographic protection) channel over which the agent can migrate.
- When an agent is loaded onto a host, this host gains full control over the agent. A malicious host may:
  1. Change the agent code, e.g. to launch attacks on other hosts;
  2. Alter, delete or read / copy (confidential) information that the agent has recorded from previous hosts.
- If an agent interacts with another agent , the other agent may attempt similar attacks. The other agent's possibilities will be limited by the host, but it is of course possible for the host to help on the attack instead.
- An agent will usually keep a trace of the hosts it visits, and relate this to information that it gathers along its route. It may also have other traceability requirements, to the extent that the hosts (or some authority) may need to sign information for non-repudiation **purposes**.

Protection of a host from a mobile code

Techniques for protection of hosts now evolve along two directions :

1. Mobile code infrastructure that is gradually enhanced with authentication, data integrity and access control mechanisms.
2. Verification of mobile code semantics.

Sandboxing

- Sandboxing consists in running a mobile code in a restricted environment called the "sandbox". An otherwise untrusted mobile code can be executed without worrying in the sandbox.
- Fig 6.4.4 shows sandbox.

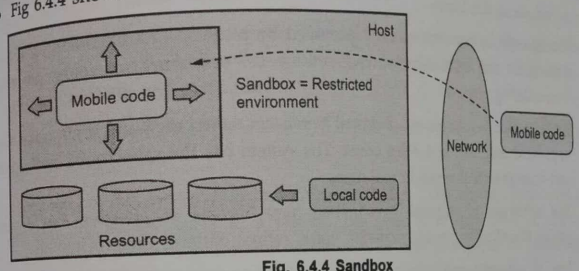


Fig. 6.4.4 Sandbox

- A sandbox can be characterized by two different mechanisms:
  a. It confines code, either through type checking, language properties, or the use of protection domains to prevent the subversion of trusted code and,
  b. It enforces a fixed policy for the execution of code.
- This approach is especially illustrated by the early Java JDK 1.0, where it was used in order to enable applets available anywhere on the internet to run within a browser.
- The major drawback of sandboxing is due to the fact that applications running in such a restrictive environment are themselves seldom useful.

6.4.4 Denial of Service

- Computer security addresses the issue of preventing unauthorized access to resources and information maintained by computers. Computer system must provide mechanism to manage security threats.
- Basic methods used to protect software and hardware is passwords, backups, maintenance of written security policies and user training.
- **Computer security** deals with the prevention and detection of unauthorized actions by users of a computer system.

Security Definition

- How to protect the valuable assets ? It is necessary to keep in safe place like a bank to protect the valuable assets. But bank is not a safe place now a day. There are so many example where bank robbery in our country.
- Bank robbery is the crime of stealing from a bank during opening hours. Protecting assets was difficult and not always effective.
- Now a day, protection is easier because many factors working against the potential criminal. Very sophisticated alarm and camera systems silently protect secure places like banks.
- Traditionally information security provided by physical i.e. rugged filing cabinets with locks and administrative mechanisms i.e. personnel screening procedures during hiring process.
- Asset protection systems are designed to recover stolen cash and high value assets, apprehend criminals and deter crime. The system has the capacity to track, protect and manage critical assets in real-time.
- The techniques of criminal investigation have become so effective that a person can be identified by genetic material, voice, retinal pattern, fingerprints etc.
- Use of networks and communications links requires measures to protect data during transmission.
- **Data security** is the science and study of methods of protecting data from unauthorized disclosure and modification.
- Data and information security is about enabling collaboration while managing risk with an approach that balances availability versus the confidentiality of data.
- **Computer security :** Generic name for the collection of tools designed to protect data and to hackers.
- **Network security :** Measures to protect data during their transmission.
- **Internet security :** Measures to protect data during their transmission over a collection of interconnected networks.

Protecting valuables

- Following are certain aspects for the need of security :
  1. Increasing threat of attacks.
  2. Fast growth of computer networking for information sharing.
  3. Availability of number of tools and resources on internet.
  4. Lack of specialized resources that may be allotted for securing system.

6.5 Security Management

- Management and handling of the pieces of secret information is generally referred to as key management. Activities of key management include selection, exchange, storage, certification, revocation, changing, expiration and transmission of the key.
- Key management is the set of processes and mechanisms which support key establishment and maintenance of ongoing keying relationship between parties, including replacing older key with new keys.

Key Establishment: Diffie-Hellman

- The Diffie-Hellman key agreement protocol was developed by Diffie and Hellman in 1976. This protocol allows two users to exchange a secret key over an insecure medium without any prior secrets.
- The Diffie-Hellman Key Exchange (DHKE) is a key exchange protocol and not used for encryption.
- The protocol has two system parameters p and g. They are both public and may be used by all the users in a system. The point is to agree on a key that two parties can use for a symmetric encryption, in such a way that an eavesdropper cannot obtain the key.
- Fig. 6.5.1 shows Diffie-Hellman key agreement protocol.

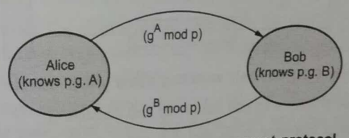


Fig. 6.5.1 Diffie-Hellman key agreement protocol

- Steps in the algorithm :
  1. Alice and Bob agree on a prime number p and a base g .
  2. Alice chooses a secret number a, and sends Bob $(g^a \bmod p)$.
  3. Bob chooses a secret number b, and sends Alice $(g^b \bmod p)$.
  4. Alice computes $((g^b \bmod p)^a \bmod p)$.
  5. Bob computes $((g^a \bmod p)^b \bmod p)$.
- Both Alice and Bob can use this number as their key. Notice that p and g need not be protected

Key Distribution

- Certificates are digital documents that are used for secure authentication of communicating parties. A certificate binds identity information about an entity to the entity's public key for a certain validity period.
- A certificate is digitally signed by a trusted third party who has verified that the key pair actually belongs to the entity.
- Authorities : The trusted party who issues certificates to the identified end entities is called a Certification Authority.
- Private-key cryptography requires shared, secret keys between each pair of communicating parties.

6.5.1 Secure Group Management

- Group uses a key pair (K^+_G, K^-_G) for communication with non-group members. There is a separate shared secret key CK_G for internal communication. Fig. 6.5.2 shows securely admitting a new group member.

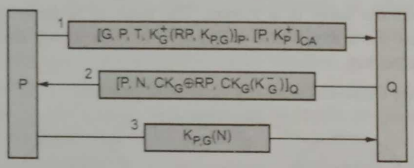


Fig. 6.5.2 Securely admitting a new group member

- Assume process P wants to join the group and contacts Q.
  1. P generates a one-time reply pad RP, and a secret key $K_{P,G}$. It sends a join request to Q, signed by itself along with a certificate containing its public key K^+_P.
  2. Q authenticates P, checks whether it can be allowed as member. It returns the group key CK_G, encrypted with the one-time pad, as well as the group's private key, encrypted as CK_G (K^-_G).
  3. Q authenticates P and sends back $K_{P,G}$ (N) letting Q know that it has all the necessary keys.

6.5.2 Authorization Management

- To avoid that each machine needs to know about all users, use capabilities and attribute certificates to express the access rights that the holder has.

- In Amoeba, restricted access rights are encoded in a capability, along with data for an integrity check to protect against tampering. A capability is a 128-bit identifier, internally organized as shown below.

Server Port (48 bits)	Object (24 bits)	Rights (8 bits)	Check (48 bits)

- First 48 bits are initialized by the object's server when the object is created and effectively form a machine-independent identifier of the object's server, referred to as the server port. Amoeba uses broadcasting to locate the machine where the server is currently located.
- Next 24 bits are used to identify the object at the given server. Next 8 bits are used to specify the access rights of the holder of the capability.
- 48-bits check field is used to make a capability unforgeable.
- When an object is created, its server picks a random check field and stores it both in the capability as well as internally in its own tables. All the right bits in a new capability are initially on, and it is this owner capability that is returned to the client.
- When the capability is sent back to the server in a request to perform an operation, the check field is verified. To create a restricted capability, a client can pass a capability back to the server, along with a bit mask for the new rights.
- The server takes the original check field from its tables, XORs it with the new rights (which must be a subset of the rights in the capability), and then runs the result through a one-way function.
- The server then creates a new capability, with the same value in the object field, but with the new rights bits in the rights field and the output of the one-way function in the check field. The new capability is then returned to the caller. The client may send this new capability to another process, if it wishes.

6.6 Fill in the Blanks

Q.1 _____ means prevention of unauthorized disclosure of information

Q.2 Integrity is a prevention of unauthorized _____ of information

Q.3 DES stands for _____.

Q.4 DES encrypts data in _____ blocks.

Q.5 DES uses _____ rounds of Feistel cipher.

Q.6 In _____ encryption, sender and receiver uses same key for encryption and decryption.

Q.7 The process of converting from _____ to ciphertext is known as enciphering or encryption.

Q.8 Secret key cryptography is also called as _____.

Q.9 DES uses 16 rounds. Each round of DES is a _____ cipher.

Q.10 DES uses _____ S-boxes, each with a 6-bit input and a 4-bit output.

Q.11 KDC means _____.

Q.12 The RSA scheme is a _____ cipher in which the plaintext and ciphertext are integers between 0 and n - 1 for some n.

Q.13 In RSA algorithm, p x q = _____.

Q.14 Asymmetric key cryptography is also called _____ key encryption.

6.7 Multiple Choice Questions

Q.1 Which one of the following is active attack ?

a Masquerade b Traffic analysis
c Eavesdropping d Shoulder surfing

Q.2 Which of the following is passive attack ?

a Relay attack b Masquerade
c Traffic analysis d Denial of Service

Q.3 In the RSA algorithm, we select 2 random large values 'p' and 'q'. Which of the following is the property of 'p' and 'q' ?

a p and q should be divisible by ϕ (n)
b p and q should be co-prime
c p and q should be prime
d p/q should give no remainder

Q.4 In RSA, ϕ (n) = _____ in terms of p and q.

a (p)/(q) b (p)(q)
c (p-1)(q-1) d (p+1)(q+1)

Q.5 RSA stands for _____.

a Rivest Shamir and Adleman
b Rock Shane and Amozen

c Rivest Shane and Amozen
d Rock Shamir and Adleman

Q.6 In an asymmetric-key cipher, the sender uses the _____ key.

a private b public
c either (a) or (b) d neither (a) nor (b)

Q.7 In public key cryptosystems , the private key is kept by

a sender b receiver
c sender and receiver d all the connected devices to the network

Q.8 User A, if wanting to send an authenticated message to user B, it would encrypt the message with A's _____ private key.

a public key b private key
c both key d third party key

Q.9 In RAS algorithm, the public key pair is _____.

a [d, n] b [p, q]
c [e, n] d [p, n]

Q.10 In RAS algorithm, the private key pair is _____.

a [d, n] b [p, q]
c [e, n] d [p, n]

Q.11 The original message is called as _____.

a ciphertext b plaintext
c cryptography d encryption

Q.12 The process of converting plaintext to ciphertext is called as _____.

a encryption b decryption
c substitution d transposition

Q.13 Interception, interruption, _____ and fabrication are the system security threats.

a traffic analysis b masquerade
c replay d modification

Q.14 Which of the following is NOT types of active attack ?

a Masquerade b Replay
c Traffic analysis d Modification of message

Q.15 _____ attacks are called as masquerade attacks.

a Interception b Interruption
c Modification d Fabrication

Q.16 _____ attacks are very difficult to detect because they do not involve any alternation of data.

a Active b Passive
c Active and passive d None of these

Q.17 The process of trying to break any cipher text message to obtain the original plain text message itself is called as _____.

a cryptanalyst b cryptography
c cryptology d cryptanalysis

Q.18 The process of converting the ciphertext into plaintext is called _____.

a encryption b decryption
c substitution d transposition

Q.19 A symmetric encryption model has _____ ingredients.

a four b three
c five d six

Answer Keys for Fill in the Blanks

1.	Confidentiality	2.	modification
3.	Data Encryption Standard	4.	64-bit
5.	16	6.	symmetric
7.	plaintext	8.	symmetric key cryptography
9.	Feistel	10.	8
11.	key distribution centre	12.	block
13.	n	14.	public

Answer Keys for Multiple Choice Questions

1.	a	2.	c
3.	c	4.	c
5.	a	6.	b
7.	b	8.	b
9.	c	10.	a
11.	b	12.	a
13.	d	14.	c
15.	b	16.	b
17.	d	18.	b
19.	c		

□□□

7 Categories of Distributed System

Syllabus

Architecture, Processes, Communication, Naming, Synchronization, Consistency and Replication, Fault Tolerance, Security : Distributed Object-based System; Distributed File System; Distributed Web-based System; Distributed Coordination based System

Contents

7.1 Distributed Object-based System

- Local objects are those whose methods can only be invoked by a local process, a process that runs on the same computer on which the object exists.
- A distributed object is one whose methods can be invoked by a remote process, a process running on a computer connected via a network to the computer on which the object exists.
- Distributed object is an object that can be accessed remotely. This means that a distributed object can be used like a regular object, but from anywhere on the network. An object is typically considered to encapsulate data and behaviour. The location of the distributed object is not critical to the user of the object.
- A distributed object might provide its user with a set of related capabilities. The application that provides a set of capabilities is often referred to as a service.
- A Business Object might be a local object or a distributed object. The term business object refers to an object that performs a set of tasks associated with a particular business process.
- Key feature of an object : it encapsulates data, the state, and the operations on those data, the methods.
- Methods are made available through an interface. The separation between interfaces and the objects implementing these interfaces is crucial for distributed systems
- Fig. 7.1.1 shows distributed object.
- A stub is defined on client side. When a client binds to a distributed object, an implementation of the object's interface, called a proxy, is then loaded into the client's address space

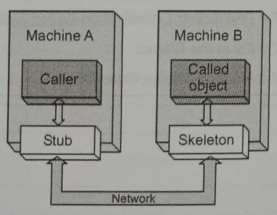


Fig. 7.1.1 Distributed object

- Then the stub passes caller data over the network to the server skeleton (machine B). The skeleton then passes received data to the called object. Skeleton waits for a response and returns the result to the client stub (machine A).
- Client stub responsible for :
  1. Initiate remote calls
  2. Marshal arguments to be sent
  3. Inform the remote reference layer to invoke the call
  4. Unmarshaling the return value
  5. Inform remote reference the call is complete

- Server skeleton responsible for :
  1. Unmarshaling incoming arguments from client
  2. Calling the actual remote object implementation
  3. Marshaling the return value for transport back to client

Compile-time vs Run-time Objects

- The most obvious form of objects, compile-time objects, are directly related to language-level objects supported by Java and C++.
- A class is a description of an abstract type in terms of a module with data elements and operations on that data.
- Using compile-time objects in distributed systems makes it easier to build distributed applications. An object can be fully defined by means of its class and the interfaces that the class implements.
- Compiling the class definition results in code that allows it to instantiate Java objects. The interfaces can be compiled into client-side and server-side stubs that permit Java objects to invoked by a remote machine.
- Compile-time objects are dependent on particular programming language. With run-time objects, the implementation is left "open" and this approach to object based distributed systems allows an application to be constructed from objects written in multiple languages. This scheme may use object adapters that act as wrappers that give implementations an object appearance.

Persistent and transient objects

- A **persistent object** is one that continues to exist even if it is currently not contained in the address space of any server process. In other words, a persistent object is not dependent on its current server.
- A **transient object** is an object that exists only as long as the server that is hosting the object. As soon as that server exits, the object ceases to exist as well.

7.1.1 Enterprise Java Beans

- Enterprise JavaBeans (EJB) is a specification of a server-side, managed component architecture and a major element of the Java Platform, Enterprise Edition (Java EE), a set of specifications for client-server programming.
- EJB is defined as a server-side component model because it supports the development of the classic style of application, where potentially large numbers of clients interact with a number of services realized through components or configuration of components.

- Fig. 7.1.2 shows architecture of EJB Server.

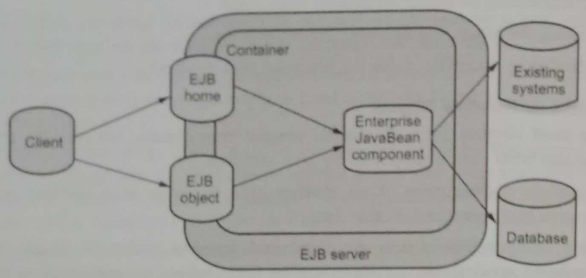


Fig. 7.1.2 Architecture of EJB Server

- The EJB server is a collection of services for supporting an EJB installation. These services include management of distributed transactions, management of distributed objects and distributed invocations on these objects, and low-level system services.
- An EJB container is a home for EJB components. A container is where a Bean lives. It provides a scalable, secure, transactional environment in which Beans can operate. It is the container that handles the object life cycle, including creating and destroying an object. The container also handles the state management of Beans.
- A container is transparent to the client. There is no client API to it.

Types of EJBs

- Four different types
  a) Stateless session bean : Transient object, called once, does its work and is done. Example : execute an SQL query and return result to caller.
  b) Stateful session bean : Transient object, but maintains client-related state until the end of a session. Example: shopping cart.
  c) Entity bean : Persistent, stateful object, can be invoked during different sessions. Example : object maintaining client info on last number of sessions.
  d) Message-driven bean: Reactive objects, often triggered by message types. Used to implement publish/subscribe forms of communication.

7.1.2 Globe Distributed Objects

- Globe objects are passive, with one or more interfaces. Multiple processes may simultaneously access same object. Changes to object's state made by one process are made visible to the others.
- Objects are physically distributed: state is partitioned and replicated across multiple machines. Processes are unaware of this: operations and state are encapsulated by object.
- Fig. 7.1.3 shows the organization of a Globe distributed shared object

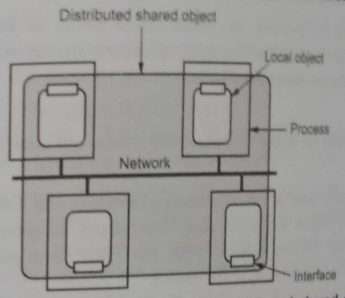


Fig. 7.1.3 Organization of a Globe distributed shared object

- Objects in Globe are referred to as distributed shared objects. Globe local objects are of two types :
  a) A primitive local object is a local object that does not contain any other local objects.
  b) Composite local object is an object that is composed of multiple local objects.
- Composition is used to construct a local object that is needed for implementing distributed shared objects.
- General organization of a local object for distributed shared objects in Globe are as follows :
  a) **Semantics sub-object** implements the functionality provided by a distributed shared object.
  b) The **communication sub-object** is used to provide a standard interface to the underlying network.
  c) **Replication sub-object** : This sub-object implements the actual distribution strategy for an object. The replication sub-object is responsible for deciding exactly when a method as provided by the semantics sub-object is to be carried out.

d) The **control sub-object** is used as an intermediate between the user-defined interfaces of the semantics sub-object and the standardized interfaces of the replication sub-object.

7.1.3 Processes

a) Object Server :

- An object server is a server tailored to support distributed objects. An object server thus acts as a place where objects live. An object consists of two parts : data representing its state and the code for executing its methods.
- For an object to be invoked, the object server needs to know which code to execute, on which data it should operate, whether it should start a separate thread to take care of the invocation, and so on.

b) Object Adapter :

- Decisions on how to invoke an object are commonly referred to as **activation policies**.
- The object adapter acts as an intermediary between the Object Request Broker (ORB) and the object implementation. The object adapter is responsible for associating object implementations with the ORB, activating/deactivating the objects and delivering requests to the objects.
- Object adapters separate object-specific behaviour from the Object Request Broker (ORB) kernel. This additional layer exists to allow for different object adapters to support the numerous functionality requirements that exist in a server.
- For example, the needs of a server representing an object database which may provide numerous individual objects that cannot all exist in memory at the same time, are very different from a server that provides a printing service.

7.1.4 Communication

a) Binding a Client to an Object

- When a process holds an object reference, it must first bind to the referenced object before invoking any of its methods. Binding results in a proxy being placed in the process's address space.
- With implicit binding, the client is offered a simple mechanism that allows it to directly invoke methods using only a reference to an object.
- With explicit binding, the client should first call a special function to bind to the object before it can actually invoke its methods. Explicit binding generally returns a pointer to a proxy that is then become locally available

b) Static versus Dynamic Remote Method Invocations

- By making use of an object-based language (Java), that will handle stub generation automatically. This approach of using predefined interface definitions is generally referred to as **static invocation**.
- Static invocations require that the interfaces of an object are known when the client application is being developed.
- It is sometimes convenient to be able to compose a method invocation at runtime, also referred to as a **dynamic invocation**.

c) Parameter Passing :

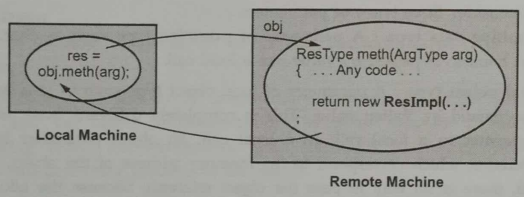
- When a client invokes a remote method with parameters, passing parameters are handled under the cover by the stub and the skeleton.
- Let us consider three types of parameters :
  1. **Primitive data type :** A parameter of primitive type such as char, int, double and boolean is passed by value like a local call.
  2. **Local object type :** A parameter of local object type such as java.lang. String is also passed by flatten value. This is completely different from passing object parameter in a local call. In a local call, an object parameter is passed by reference, which correspond to the memory address of the object. In a remote call, there is no way to pass the object reference because the address on one machine is meaningless to a different Java VM. Any object can be used as a parameter in a remote call as long as the object is serializable. The stub serializes the object parameter and sends it in a stream across the network. The skeleton deserializes stream into an object.
  3. **Remote object type :** Remote objects are passed differently from the local objects. When a client invokes a remote method with a parameter of some remote object type, the reference of the remote object is passed. The server receives the reference and manipulates the parameter through the reference.

d) Java RMI :

- Java RMI is a mechanism that allows a Java program running on one computer to apply a method to an object living on a different computer. RMI is an implementation of the of the Distributed Object programming model similar to CORBA, but simpler and specialized to the Java language.
- The syntax of the remote method invocation looks like an ordinary Java method invocation. The remote method call can be passed arguments computed in the context of the local machine. It can return arbitrary values computed in the context of the remote machine. The RMI runtime system transparently communicates all data required. In some ways Java RMI is more general than

CORBA, it can exploit Java features like object serialization and dynamic class loading to provide more complete object-oriented semantics.

- Java RMI provides the following elements :
  a. Remote object implementations.
  b. Client interfaces, or stubs, to remote objects.
  c. A remote object registry for finding objects on the network.
  d. A network protocol for communication between remote objects and their client (this protocol is JRMP, i.e. Java Remote Method Protocol).
  e. A facility for automatically creating (activating) remote objects on-demand.
- Code running in the local machine holds a remote reference to an object **obj** on a remote machine.



The Remote Interface

- In RMI, a common remote interface is the minimum amount of information that must be shared in advance between "client" and "server" machines. It defines a high-level "protocol" through which the machines will communicate.
- A remote interface is a normal Java interface, which must extent the marker interface java.rmi.Remote. All methods in a remote interface must be declared to throw the java.rmi.RemoteException exception.

A Simple Example :

- A file MessageWriter.java contains the interface definition :

```
import java.rmi.* ;
public interface MessageWriter extends Remote {
    void writeMessage(String s) throws RemoteException ;
}
```

- This interface defines a single remote method, writeMessage( ).
- The interface **java.rmi.Remote** is a marker interface. It declares no methods or fields; however, extending it tells the RMI system to treat the interface concerned as a remote interface.

e) Object-based Messaging

- CORBA, which stands for Common Object Request Broker Architecture, is an industry standard developed by the OMG to aid in distributed objects programming.
- CORBA is just a specification for creating and using distributed objects; CORBA is not a programming language.
- The CORBA architecture is based on the object model. This model is derived from the abstract core object model defined by the OMG.
- CORBA-based system is a collection of objects that isolates the requestors of services (clients) from the providers of services (servers) by a well-defined encapsulating interface.
- In CORBA's callback model, a client provides an object that implements an interface containing callback methods. These methods can be called by the underlying communication system to pass the result of an asynchronous invocation.
- An important design issue is that asynchronous method invocations do not affect the original implementation of an object. CORBA is a client-server system, in which client processes on client machines can invoke operations on objects located on server machines.

7.1.5 Naming : CORBA Object References

- In CORBA, every object has a unique Interoperable Object Reference (IOR), that can be used to locate the object. The IOR contains information such as the host on which the object resides, the TCP/IP port on which the object is listening for requests, and an object key that distinguishes the object from other objects listening on that port.
- Current CORBA systems all support the same language-independent representation of an object reference, which is called an Interoperable Object Reference.
- Fig. 7.1.4 shows organization of an IOR with specific information for **Internet InterOrb Protocol** (IIOP).
- Each IOR starts with a repository identifier. This identifier is assigned to an interface so that it can be stored and looked up in an interface repository.
- It is used to retrieve information on an interface at runtime, and can assist in, for example, type checking or dynamically constructing an invocation. CORBA used the Internet Inter-ORB Protocol (IIOP) for communication between nodes.

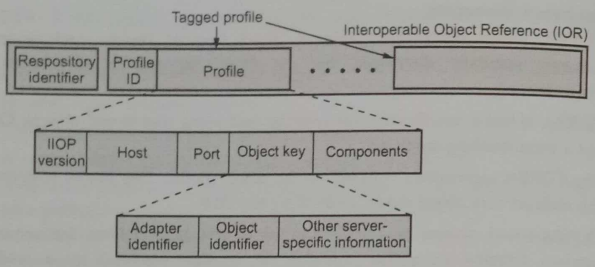


Fig. 7.1.4 Shows organization of an IOR with specific information for Internet InterOrb Protocol

7.1.6 Globe Object References

- In Globe, each distributed shared object is assigned a globally unique object identifier (OID), which is a 256-bit string.
- Globe OIDs can be used only for comparing object references. For example, suppose processes A and B are each bound to a distributed shared object.
- Each process can request the OID of the object they are bound to. If and only if the two OIDs are the same, then A and B are considered to be bound to the same object.
- Globe support two kinds of addresses: stacked address and instance address.
- A stacked address represents a layered protocol suite, where each layer is represented by the three-field record.

Field	Description
Protocol identifier	A constant representing a (known) protocol
Protocol address	A protocol-specific address
Implementation handle	Reference to a file in a class repository

Table 7.1.1 Stacked address

- Instance address, which consists of the two fields.

Field	Description
Implementation handle	Reference to a file in a class repository
Protocol address	String that is used to initialize an implementation

Table 7.1.2 : Instance address

7.2 Distributed File System

- File is a collection of data with a user view (file structure) and a physical view (blocks).
- A Dstributed File System (DFS) is a file system with distributed storage and users. DFS provides transparency of location, access, and migration of files. DFS systems use cache replicas for efficiency and fault tolerance.
- A distributed file system enables programs to store and access remote files exactly as they do local ones.
- Two distributed systems that have been in widespread used for a decade or more.
  a. Sun Network File System (NFS).
  b. Andrew File System (AFS).
- File systems are abstraction that enables users to read, manipulate and organize data. Typically the data is stored in units known as files in a hierarchical tree where the nodes are known as directories.
- The file system enables a uniform view, independent of the underlying storage devices which can range between anything from floppy drives to hard drives and flash memory cards. Since file systems evolved from stand-alone computers the connection between the logical file system and the storage device was typically a one-to-one mapping.
- Even software RAID that is used to distribute the data on multiple storage devices is typically implemented below the file system layer.
- Distributed file system is a resource management component of a distributed operating system. Distributed file system is a part of distributed system that provides a user with a unified view of the files on the network. A machine that holds the shared files is called a server, a machine that accesses the files is called a client.
- The file systems in the 1970s were developed for centralized computer systems, where the data was only accessed by one user at a time. When multiple users and processes were to access files at the same time a notion of locking was introduced. There are two kinds of locks, read and write.
- Goals of distributed file systems are as follows :
  1. **Network transparency :** Clients should be able to access remote files using the same operations that apply to local files.
  2. **High availability :** Users should have the same easy access to files, irrespective of their physical location.

7.2.1 Distributed File System Requirements

- A good distributed file system should have the following features :

1. Transparency :

Following are the desirable transparency :

a. **Structure transparency :** DFS normally uses multiple file servers for performance, scalability and reliability reasons. Each file server is normally user process or sometimes a kernel process that is responsible for controlling a set of secondary storage devices of the node on which it runs. In multiple file servers, the multiplicity of file servers should be transparent to the clients of a DFS. Client should not know the number or locations of the file server and storage devices.

b. **Access transparency :** Both local and remote files should be accessible in the same way.

c. **Naming transparency :** The name of file should give no lint as to where the file is located.

d. **Replication transparency :** The clients do not need to know the existence or locations of multiple file copies.

2. **User mobility :** User should not force to work on a specific node but should have the flexibility to work on different nodes at different times.
3. **Performance :** The performance of the file system is usually measured as the average amount of time needed to satisfy client requests.
4. **Scalability :** A good distributed file system should be designed to easily cope with the growth of nodes and users in the system.
5. **High availability :** DFS should continue to function even when partial failures occur due to the failure of one or more components, such as a communication link failure, a machine failure or a storage device crash.
6. **High reliability :** In a good distributed file system, the probability of loss of stored data should be minimized as far as practicable.
7. **Security :** DFS should be secure so that its users can be confident of the privacy of their data.

7.2.2 Sun Network File System

- It is developed by Sun Microsystems to provide a distributed file system independent of the hardware and operating system.

- NFS provides transparent access to remote files on a LAN, for clients running on UNIX and other operating systems.
- It is a client/server application that provides shared file storage for clients across a network.
- NFS is stateless. All client requests must be self-contained. Each procedure call contains all the information necessary to complete the call. Server maintains no "between call" information.
- It uses an External Data Representation (XDR) specification to describe protocols in a machine and system independent way.
- NFS is implemented on top of a Remote Procedure Call package (RPC) to help simplify protocol definition, implementation, and maintenance.
- NFS is not so much a true file system, as a collection of protocols that together provide clients with a model of a distributed file system.
- **Goals of NFS design :**
  1. Compatibility : NFS should provide the same semantics as a local Unix file system. Programs should not need or be able to tell whether a file is remote or local.
  2. Easy deployable : Implementation should be easily incorporated into existing systems remote files should be made available for local programs without these having to be modified or re-linked.
  3. Machine and OS independence : NFS Clients should run in non-Unix platforms.
  4. Efficiency : NFS should be good enough to satisfy users, but did not have to be as fast as local FS. Clients and Servers should be able to easily recover from machine crashes and network problems.

7.2.3 NFS Architecture

- Fig. 7.2.1 shows NFS architecture.
- The Virtual File System (VFS) interface is implemented using a structure that contains the operations that can be done on a file system.
- Likewise, the vnode interface is a structure that contains the operations that can be done on a node (file or directory) within a file system.

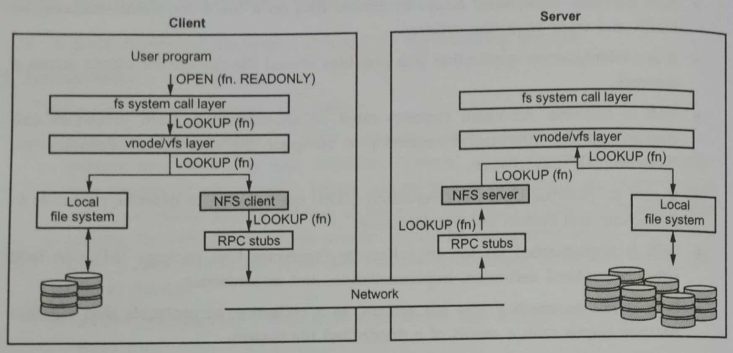


Fig. 7.2.1 NFS architecture

- There is one VFS structure per mounted file system in the kernel and one vnode structure for each active node. Using this abstract data type implementation allows the kernel to treat all file systems and nodes in the same way without knowing which underlying file system implementation it is using.
- Each vnode contains a pointer to its parent VFS and a pointer to a mounted-on VFS. This means that any node in a file system tree can be a mount point for another file system.
- A root operation is provided in the VFS to return the root vnode of a mounted file system. This is used by the pathname traversal routines in the kernel to bridge mount points.
- The root operation is used instead of keeping a pointer so that the root vnode for each mounted file system can be released.

Server Side

- Because the NFS server is stateless, when servicing an NFS request it must commit any modified data to stable storage before returning results.
- The implication for UNIX based servers is that requests which modify the file system must flush all modified data to disk before returning from the call.
- For example, on a write request, not only the data block, but also any modified indirect blocks and the block containing the inode must be flushed if they have been modified.

Client Side

- The Sun implementation of the client side provides an interface to NFS which is transparent to applications.
- To make transparent access to remote files work we had to use a method of locating remote files that does not change the structure of path names.
- Transparent access to different types of file systems mounted on a single machine is provided by a new file system interface in the kernel.
- Each "filesystem type" supports two sets of operations: the Virtual Filesystem (VFS) interface defines the procedures that operate on the filesystem as a whole; and the Virtual Node (vnode) interface defines the procedures that operate on an individual file within that filesystem type.
- The ability of the client to simply retry the request is due to an important property of most NFS requests: they are idempotent.
- An operation is called **idempotent** when the effect of performing the operation multiple times is equivalent to the effect of performing the operation a single time.

Working :

- When a user is accessing a file, the kernel determines whether the file is a local file or an NFS file. The kernel passes all references to local files to the local file access module and all references to the NFS files to the NFS client module.
- The NFS client sends RPC requests to the NFS server through its TCP/IP module. Normally, NFS is used with UDP, but newer implementations can use TCP. Then the NFS server receives the requests on port 2049.
- Next, the NFS server passes the request through its local file access routines, which access the file on server's local disk.
- After the server gets the results back from the local file access routines, the NFS server sends back the reply in the RPC reply format to the client.
- While the NFS server is handling the client's request, the local file system needs some amount of time to return the results to the server. During this time the server does not want to block other incoming client requests.
- To handle multiple client requests, NFS servers are multithreaded or there are multiple servers running at the same time.

7.2.4 Cluster-based Distributed File Systems

- Google File System (GFS) is a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance even with inexpensive commodity hardware, and delivers high average performance to a large number of clients.
- Fig. 7.2.2 shows organization of a Google cluster of servers.

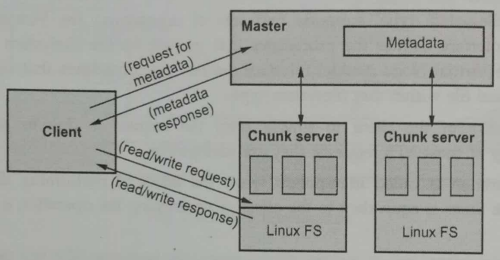


Fig. 7.2.2 Organization of a Google cluster of servers

- GFS consists of a single master and multiple chunk servers and is accessed by multiple clients. Files are divided into fixed-sized chunks of 64 MB.
- Each chunk has an immutable and globally unique chunk handler, which is assigned by the master at the time of chunk creation. By default, each file chunk is replicated on 3 different chunk servers.
- Single Master : The master maintains all the file system metadata. This includes mapping from file to chunks, chunk locations, etc. The master also periodically sends Heart Beat messages to the chunk server to give it instructions and collect its state

7.2.5 Communication

- In NFS, all communication between a client and server proceeds along the open network computing RPC protocol. ONC RPC is similar to other RPC systems.
- Every NFS operations can be implemented as a single remote procedure call to a file server. Up until NFS version 4, the client was made responsible for making the server life as easy as possible by keeping requests relatively simple.

- For example, in order to read data from a file for the first time, a client normally first has to look up the file handle using the lookup operation, after which it can issue a read request.
- This approach requires two successive RPCs. In a wide-area system the drawback is that the extra latency of a second RPC may lead to a performance degradation.
- NFS version 4 supports compound procedures by which several RPCs can be grouped into a single request. In the previous example, the client combines the lookup and read request into a single RPC.
- In the case of version 4, it is also necessary to open the file before reading can take place. There are no transactional semantics associated with compound procedures.
- The operations are simply handled in the order as requested. If there are concurrent operations from other clients then no measures are taken to avoid conflicts.

7.2.6 Naming and Mounting

- Workstations are designated as clients or file servers. All workstations are treated as peers.
- Servers export whole file systems, but clients can mount any sub-directory of a remote file system on top of a local file system, or on top of another remote file system.
- In fact, a remote file system can be mounted more than once, and can even be mounted on another copy of itself. This means that clients can have different "names" for file systems by mounting them in different places.
- The NFS naming model provides complete transparent access to a remote file system as maintained by a server. This transparency is achieved by letting a client be able to mount a remote file system into its own local file system.
- Each client maintains a table which maps the remote file directories to servers.
- Instead of mounting an entire file system, NFs allows clients to part of a file system. A server is said to export directory when it makes that directory and its entries available to clients.
- The mount protocol is used to establish the initial logical connection between a server and a client. A mount operation includes the name of the remote directory to be mounted and the name of the server machine storing it.
- The server maintains an export list which specifies local file system that it exports for mounting along with the permitted machine names.

- An NFS server can itself mount directories that are exported by other servers. However, it is not allowed to export those directories to its own clients.
- Instead, a client will have to explicitly mound such a directory from the server that maintains it.
- There is a problem with this model that has to do with deciding when a remote file system should be mounted. To deal with the problem NFS implements on demand mounting of a remote file system that is handled by an automounter, which runs as a separate process on the client's machine.
- Fig. 7.2.3 shows simple automounter in NFS.

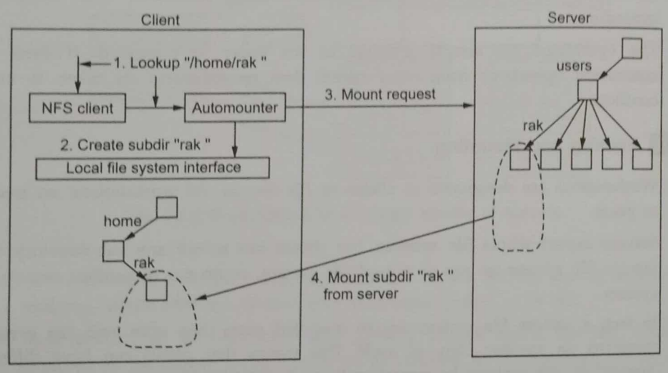


Fig. 7.2.3 : Automounter in NFS

- To access a file, a client must first look up its name in a naming service and obtain the associated file handle. A file handle is a reference to a file within a file system.
- It is independent of the name of the file it refers to. A file handle is created by the server that is hosting the file system and is unique with respect to all file systems exported by the server.
- It is created when the file is created. The client is kept ignorant of the content of a file handle. In version 4, file handles can have a variable length up to 128 bytes.
- The automounter was added to the UNIX implementation of NFS in order to mount a remote directory dynamically whenever an 'empty' mount point is referenced by a client.

- Automounter has a table of mount points with a reference to one or more NFS servers listed against each. It sends a probe message to each candidate server and then uses the mount service to mount the file system at the first server to respond.
- Automounter keeps the mount table small. Automounter provides a simple form of replication for read-only file systems.
- An NFS file has a number of associated attributes. With NFS version 4, the set of file attributes has been split into a set of mandatory attributes that every implementation must support (type, size, change, FSID), a set of recommended attributes that should be preferably supported, and an additional set of named attributes.
- Named attributes are actually not part of the NFS protocol, but are encoded as an array of (attribute, value)-pairs in which an attribute is represented as a string, and its value as an un-interpreted sequence of bytes. They are stored along with the file (or directory) and NFS provides operations to read and write attribute values.
- The mount protocol is used to establish the initial logical connection between a server and a client. A mount operation includes the name of the remote directory to be mounted and the name of the server machine storing it.
- The server maintains an export list which specifies local file system that it exports for mounting along with the permitted machine names.
- UNIX uses /etc./exports for this purpose. The list has a maximum length, NFS is limited in scalability. Any directory within an exported file system can be mounted remotely on a machine. When the server receives a mount request, it returns a file handle to the client.
- File handle is basically a data-structure of length 32 bytes. It serves as the key for further access to files within the mounted system.
- In UNIX term, the file handle consists of a file system identifier that is stored in super block and an inode number to identify the exact mounted directory within the exported file system.
- In NFS, one new field is added in inode that is called the generic number. Mount can be is of three types -
  1. Soft mount : A time bound is there.
  2. Hard mount : No time bound.
  3. Automount : Mount operation done on demand.

7.2.7 Caching and Replication

- The caching NFS client has been designed to address several performance issues relating to the implementation of NFS. It also provides minor support for file replication.
- Each client can have a memory cache that contains data previously read from the server. In addition, there may also be a disk cache that is added as an extension to the memory cache, using the same consistency parameters.
- Typically, clients cache file data, attributes, file handles, and directories. Different strategies exist to handle consistency of the cached data, cached attributes, and so on.
- NFS version 4 supports two approaches for caching file data. The simplest approach is when a client opens a file and caches the data it obtains from the server as the result of various read operations.
- Fig. 7.2.4 shows client side caching in NFS.

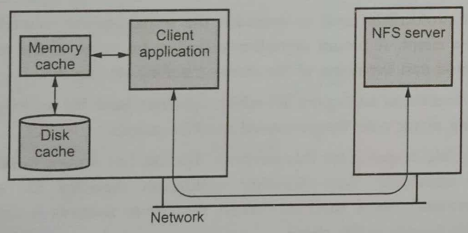


Fig. 7.2.4 : client side caching in NFS

- In addition, write operations can be carried out in the cache as well. When the client closes the file, NFS requires that if modifications have taken place, the cached data must be flushed back to the server. This approach corresponds to implementing session semantic.
- Once a file has been cached, a client can keep its data in the cache even after closing the file. Also, several clients on the same machine can share a single cache.
- Blocks that are read from a NFS server are kept in a disk cache. As blocks of a file are read they are added to the cache for this file. Once the file is complete it is marked as being persistent and can survive client crashes.
- This is possible because once the whole file is cached kernel data structures are no longer necessary to gain information about which blocks of the file are present.

- If the cache becomes full a process runs through the cache and removes persistent objects with preference for least recently used objects.
- Partially cached files are not eligible for cleaning as the additional complexity of updating the kernel data structures associated with these files during cache cleaning is tedious at best.
- As this is mechanism favors files that are complete in the cache a background process runs to collect uncached blocks of partially cached files.
- The client also supports an RPC lookup cache that holds recently requested information about file and directory attributes. These attribute requests actually contribute a large amount of the RPC traffic associated with NFS.
- This cache is however limited in its usefulness as the cache must expire after a time in the order of 100 ms to maintain NFS semantics.
- If the cache is held for longer then the client may no longer hold an accurate view of the attributes held on the server and there are no conflict resolution procedures in the NFS protocol to handle such a situation.
- Finally the client supports asynchronous writing of files. Buffering of writes of a file to a server avoids traffic in the situation where a file is modified many times in succession.
- This is of limited importance in the HTTP server application but becomes much more poignant when serving files to a group of workstations.

7.2.8 Advantages and Disadvantages of NFS

- Advantages of network file systems
  1. Easy to share if files available on multiple machines
  2. Easier to administer server than clients
  3. Access way more data that fits on your local disk
- Disadvantages
  1. Network slower than local disk
  2. Network or server may fail even when client OK
  3. Complexity, security issues

7.3 Distributed Web-based System

- The World Wide Web is a large distributed system. The WWW is an evolving system for publishing and accessing resources and services across the Internet.
- Web is an open system. Its operations are based on freely published communication standards and documents standards. The Web is one with respect to the types of 'resource' that can be published and shared on.
- World Wide Web is collection of millions of files stored on thousands of servers all over the world. These files represent documents, pictures, video, sounds, programs, interactive environments.
- The World Wide Web (WWW) can be viewed as a huge distributed system with millions of clients and servers for accessing linked documents. Servers maintain collections of documents while clients provide users an easy-to-use interface for presenting and accessing those documents.
- A document is fetched from a server, transferred to a client, and presented on the screen. To a user there is conceptually no difference between a documents stored locally or in another part of the world.

7.3.1 Architecture : Traditional Web-based Systems

- Fig. 7.3.1 shows traditional web based systems.

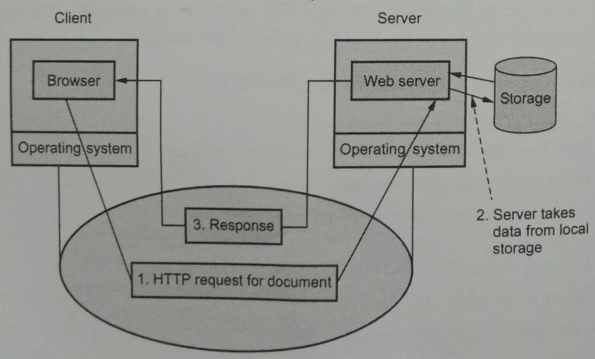


Fig. 7.3.1 Traditional web based systems

- Clients use browser application to send URLs via HTTP to servers requesting a Web page. Web pages constructed using HTML or other markup language and consist of text, graphics, and sounds plus embedded files.

- Servers respond with requested Web page. Client's browser renders Web page returned by server. The entire system runs over standard networking protocols.
- Web documents are known as `pages', each of which can be addressed by an identifier called a uniform resource locator. Web pages are usually grouped into `sites', sets of pages published together.
- The standard web transfer protocol is Hyper Text Transfer Protocol (HTTP). The HTTP protocol consists of two fairly distinct items : The set of requests from browsers to servers and the set of responses going back the other way.
- HTTP uses the services of TCP. HTTP is a stateless protocol. The client initializes the transaction by sending a request message. The server replies by sending a response.
- A web browser is a program. Web browser is used to communicate with web servers on the Internet, which enables it to download and display the web pages. Netscape Navigator and Microsoft Internet Explorer are the most popular browser software available in market. Browser interacts with web as well as computer operating system and with other programs.
- A web server is a computer connected to the Internet that runs a program that takes responsibility for storing, retrieving and distributing some of the web files. A web client is a computer that requests files from the web.
- Well-defined set of languages and protocols that are independent of the hardware or operating system are required to run on the computers. HTTP defines a standard rule by which browsers and any other types of client interact with web servers. Main features are
  a. Request-reply interaction
  b. Content types may or may not be handled by browser - using plug-in or external helper.
  c. One resource per request so several requests can be made concurrently.
  d. Simple access control : Any user with network connectivity to a web server can access any of its published resources.

Web Document

- Popular formats for web documents are HTML, followed by GIF and JPG, ASCII text, and Postscript in that order. The most compression tools used are GNU zip, Zip and Compress.
- A web page is an HTML document that is stored on a web server. A web site is a collection of web pages belonging to a particular organization.

- URL of these pages shares a common prefix, which is the address of the home page of the size. Search engines are a bottom-up approach for finding your way around the web. Some search engines search only the titles of web pages. While other search every word. Keywords can be combined with Boolean operations, such as AND, OR and NOT, to produce rather complicated queries.
- Home page is the front door of a web site. When a person or organization says "My web site is at www.sangeeta.com", the URL to which they refer is the URL of the site's home page. The home page introduces the rest of the web site and provides links that leads to other pages on the site.
- HTML documents are in plain text format that contain embedded HTML tags. Documents can be created in any text editor. There are also many other tools, including editors, designed specifically to assist in creating HTML documents. To view an HTML document, the user needs a browser.
- HTML defines the structural elements in a document such as headers, and addresses, layout information and the use of inline graphics together with the ability to provide hyper text links.
- HTML does not provide any structure to web pages. HTML mixes the content with the formatting with web pages in HTML, it is very difficult for a program to search particular word.
- To overcome this problem, two new language Extensible Markup Language (XML) and Extensible Style Language (XSL) are used. The XML and XSL specifications are much stricter than HTML specification.
- HTML and XML can also include all kinds of tags that refer to embedded documents. An embedded document is when one document is embedded within another.
- HTML tags are just like web pages. It can include text formatting, numbering, bullets, horizontal lines, backgrounds, hyperlinks, and HTML styles. It uses MIME protocol for sending. Rich text can be read by most word processing applications. MIME formatting is created just for e-mail. MIME formatting can include text formatting, pictures, video, sound, and other information.
- A content-type declaration must contain two identifiers, a content-type and a subtype, separated by a slash. In the example, image is the content type and gif is the subtype. The standard defines seven basic content types.

Sr. No.	Content type	Use
1.	Text	Textual (document)
2.	Image	Photograph
3.	Audio	A sound recording
4.	Video	A video recording including motion
5.	Appication	Raw data for program
6.	Multipart	Multiple messages
7.	Message	An entire e-mail message

- There are seven standardized content-types that can appear in a MIME content-type declaration.

Type	Subtype	Description
Text	Plain	Uniformatted text.
Multipart	Mixed	Body contain ordered parts of different data types.
	Parallel	Same as above, but no order.
	Digest	Similar to mixed, but the default is message.
	Alternative	Parts are different versions of the same message.
Video	MPEG	Video is in MPEG format.
Audio	Basic	Single channel encoding of voice at 8 kHZ. (Sound file)
Message	JPEG	Image is in JPEG format.
	GIF	Image is in GIF.
	Partial and external body	An entire e-mail message or an external reference to a message.
Application	Postscript	Adobe postscript.
	Octet stream	General binary data.

Multi-tiered Architecture

- WWW uses client-server interaction. The browser program acts as a client that uses the Internet to contact a remote server for a copy of the requested page. The server on the remote system returns a copy of the page along with the additional information.
- Common Gateway Interface makes dynamic computation of web pages possible. It allows a web server to associate some URLs with computer program instead of static documents on disk.

- When a browser request one of the special URLs the server runs the associated computer program and sends the output from the program back to the user. A server can have an arbitrary number of CGI programs that perform different computations. The server uses the URL in the incoming request to determine which CGI program to run.
- CGI makes dynamic computation of web pages possible. It allows a web server to associate some URLs with computer program instead of static documents on disk. Fig. 7.3.2 shows working of CGI.

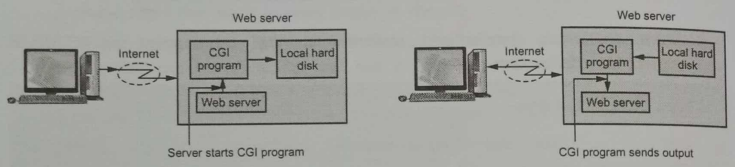


Fig. 7.3.2 Working of CGI

- When a browser request one of the special URLs the server runs the associated computer program and sends the output from the program back to the user. A server can have an arbitrary number of CGI programs that perform different computations.
- The server uses the URL in the incoming request to determine which CGI program to run. CGI working is as follows :
  1. The browser sends a request to the server. If the requested URL corresponds to a CGI program, the server starts the appropriate program and passes to the program a copy of the request.
  2. The server then sends the output from the CGI program back to the browser in the form of reply.
  3. From a browser's point of view, there is no difference between a URL that corresponds to a static document and one that corresponds to a CGI program.
  4. Requests for both static documents and CGI output have the same syntactic form.
- Web documents can be built in two ways :
  1. Static : locates and returns the object identified in the request. Static objects include predefined HTML pages and JPEG or GIF files. It does not require web servers to communication with any server-side application.

2. Dynamic : the request is forwarded to an application system where the reply is generated dynamically, i.e. data is generated through a server-side program execution.

7.3.1.1 Web Services

- Web services describes a standardized way of integrating Web-based applications using the XML, SOAP, WSDL and UDDI open standards over an Internet protocol backbone.
- Web Services components are as follows :
  a. A standard way for communication (SOAP)
  b. A uniform data representation and exchange mechanism (XML)
  c. A standard meta language to describe the services offered (WSDL)
  d. A mechanism to register and locate WS-based applications (UDDI)
- XML is used to tag the data, SOAP is used to transfer the data, WSDL is used for describing the services available and UDDI is used for listing what services are available. Fig. 7.3.3 shows web service.

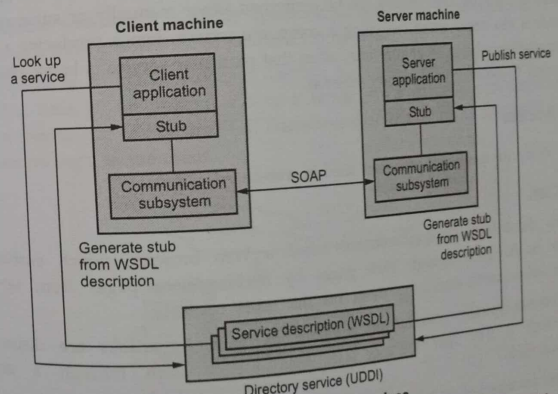


Fig. 7.3.3 Web services

- SOAP was originally part of the specification that included the Web Services Description Language (WSDL) and Universal Description, Discovery, and Integration (UDDI). It is used now without WSDL and UDDI. Instead of the discovery process described in the History of the Web Services Specification section below, SOAP messages are hard-coded or generated without the use of a repository.

- SOAP generally uses HTTP.
- WSDL is a formal language. It is same as the Interface Definition Languages (IDL) used to support RPC-based communication. A WSDL description contains the precise definitions of the interfaces provided by a service, that is, procedure specification, data types, the (logical) location of services, etc. An important issue of a WSDL description is that can be automatically translated to client side and server-side stubs, again, analogous to the generation of stubs in ordinary RPC-based systems.

Web Service Composition and Coordination

- Web Service Composition provides an open, standards-based approach for connecting web services together to create higher-level business processes. Standards are designed to reduce the complexity required to compose web services, hence reducing time and costs, and increase overall efficiency in businesses.
- Web service composition involves combining and coordinating a set of services with the purpose of achieving functionality that cannot be realized through existing ser-vices. This process can be performed either manually or automatically, while it can occur when designing a composite service, hence producing a static composition schema or at run-time, when that particular service is being executed, leading to dynamic composition schemas.

7.3.2 Process

Here we discuss important processes used in web-based system.

7.3.2.1 Client

- Web client is a piece of software called a Web browser, which enables a user to navigate through Web pages by fetching those pages from servers and subsequently displaying them on the users' screen.
- A browser typically provides an interface by which hyperlinks are displayed in such a way that the user can easily select them through a single mouse click.
- Important aspect of Web browsers :
  a. Platform independent.
  b. Should be easily extensible so that it, can support any type of document that is returned by a server.
  c. Another client-side process that is often used is a Web proxy.

- Fig. 7.3.4 shows logical component of web browser.

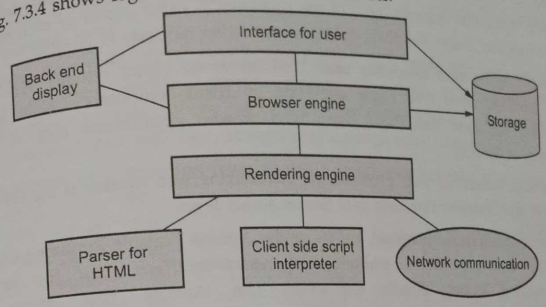


Fig. 7.3.4 Web browser logical component

7.3.2.2 Apache Web Server

- The Apache Web server is a free HTTP (Web) server developed by the Apache Server Project. It is a reliable, efficient, and easily extensible Web Server.
- Apache web server is highly optimized for serving up static resources. It is open source software with cross platform functionality.
- Apache is an HTTP server or daemon. A daemon is a UNIX term for a process that runs in the background of a server which normally provides continuous service to clients. Secure Shell Daemon is another example of daemon. It provides secure login to the client.

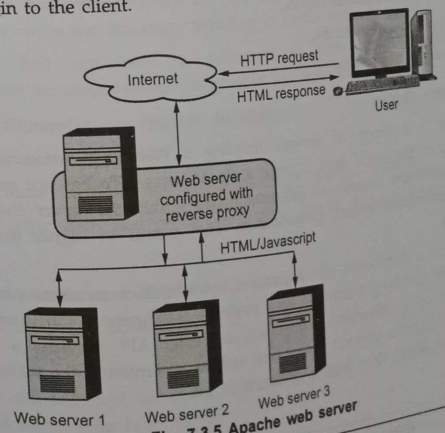


Fig. 7.3.5 Apache web server

- Apache creates a number of processes that run in the background waiting for client connection requests. Each of the processes is exact copies of each other and provides that ability to answer multiple connection requests in parallel.
- Apache is the most commonly used Web Server on Linux systems. Web Servers are used to serve Web Pages requested by client computers. Clients typically request and view Web Pages using Web Browser applications such as Firefox, Opera, or Mozilla.
- The most commonly used protocols for transferring web content is the HTTP and Hyper Text Transfer Protocols over Secure socket layer.
- Apache's runtime environment, known as the Apache Portable Runtime (APR), is a library that provides a platform-independent interface for file handling, networking, locking, threads, and so on. The Apache core makes few assumptions on how incoming requests should be handled.
- **For example :**
  a. Hook is used for translating URL to a local file name.
  b. Hook also performs writing information to log.
  c. It checks clients identification and access rights.

Advantages of Apache Web server

1. Apache is industry standard for most web servers.
2. It is Open source.
3. It supports remote administration.
4. It also support multi-platform.
5. The software is free of cost.

7.3.2.3 Web Server Clusters

- A cluster is a group of servers running a Web application simultaneously, appearing to the world as if it were a single server. To balance server load, the system distributes requests to different nodes within the server cluster, with the goal of optimizing system performance. This results in higher availability and scalability.
- A cluster is a distributed, not a parallel, system. Each machine runs a separate copy of the operating system on each node. A management subsystem creates the abstraction of an integrated entity and a cluster API provides a collection of system interfaces to perform operations such as determining the set of nodes on a cluster, monitor all state, launch applications, etc.

- There is a difference between front ends operating as transport layer switches and application layer.
- When client send HTTP request and setup a TCP connection with the server. A transport-layer switch simply passes the data sent along the TCP connection to one of the servers. The response from that server is returned to the switch, which will then forward it to the requesting client.
- **Drawback of a transport-layer switch :** switch cannot take into account the content of the HTTP request that is sent along the TCP connection. It can only base its redirection decisions on server loads.

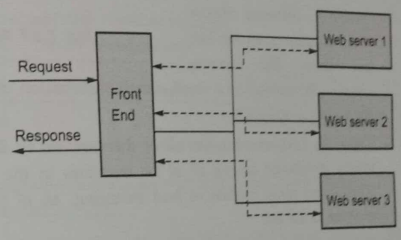


Fig. 7.3.6 Front end for web server

- Fig. 7.3.6 shows the principle of using a server cluster in combination with a front end to implement a Web service.
- **Content-aware request distribution :** it first inspects the HTTP request and decides which server it should forward that request to. For example, if the front end always forwards requests for the same document to the same server, the server may cache the document resulting in better response times.
- Approach that combines the efficiency of transport-layer switch and the functionality of content-aware distribution has been developed.

Advantages of Content-aware request distribution

1. Higher response times.
2. It makes more efficient use of the available storage capacity.
3. It allows using dedicated servers to handle special documents such as audio or video.

Problem with content-aware distribution

1. Front end are over loaded.

Round Robin DNS

- Another alternative to set up a Web server cluster is to use round-robin DNS. Fig. 7.3.7 shows round robin DNS.

- With round-robin DNS a single domain name is associated with multiple IP addresses. When resolving a host name, a browser would receive a list of multiple addresses, each address corresponding to a server.

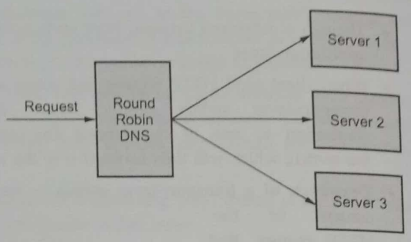


Fig. 7.3.7 Round Robin DNS

- Normally, browsers choose the first address on the list, but most DNS servers circulate the entries. As a result, simple distribution of requests over the servers in the cluster is achieved.
- Using the DNS round robin, all of the requests to the particular site have been evenly distributed among all of the machines in the cluster. Therefore, with the DNS round robin method of load balancing, all of the nodes in the cluster are exposed to the net.
- It is very effective for small to medium size business or organizations. It is extremely popular among ISPs, e-commerce sites, universities, and other cost sensitive sites.

Advantages of DNS Round Robin

1. Inexpensive and easy to set up
2. Simplicity. It does not require any networking experts to set up.

Disadvantages of DNS Round Robin

1. No support for high availability
2. No support for server affinity. Server affinity is a load-balancing system's ability to manage a user's requests

7.3.3 Communication

- In Web-based distributed systems, following communication protocols are used.
  1. HTTP for traditional web systems
  2. SOAP for web services

7.3.3.1 Hypertext Transfer Protocol

- HTTP is an application layer protocol. The Web client and the Web server are application programs. Application layer programs do useful work like retrieving

Web pages, sending and receiving email or transferring files. Lower layers take care of the communication details

- The client and server send messages and data without knowing anything about the communication network.
- HTTP is an asymmetric request-response client-server protocol. An HTTP client sends a request message to an HTTP server. The server, in turn, returns a response message. In other words, HTTP is a pull protocol; the client pulls information from the server.
- Fig. 7.3.8 shows HTTP request and response messages.

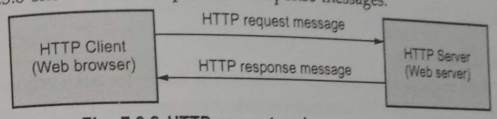


Fig. 7.3.8 HTTP request and response messages

- HTTP is core request-response protocol for web. It consists of four phases :
  1. Open connection : Based on URL
  2. Request : Client opens connection to server and sends request method, URL, HTTP version number, header information and terminated with blank line.
  3. Response : Server processes request and sends HTTP protocol version and status code, header information, terminated by blank line and text (data).
  4. Close connection
- All communication between clients and servers is based on HTTP. Servers listen on port 80. HTTP is a simple protocol; a client sends a request to a server and waits for a response.

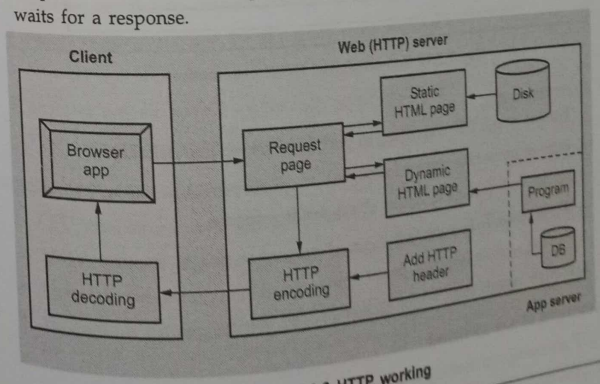


Fig. 7.3.9 HTTP working

- HTTP is stateless; it does not have any concept of open connection and does not require a server to maintain information on its clients.
- HTTP is based on TCP; whenever a client issues a request to a server, it first sets up a TCP connection and sends the message on that connection. The same connection is used for receiving the response. Fig. 7.3.9 shows working of HTTP server.
- HTTP connections are of two types
  1. Non-Persistent HTTP
  2. Persistent HTTP

Non-Persistent Connection

- In this type of connection, one TCP connection is made for each request/response. The initial design HTTP 1.0 uses non-persistent connections. The TCP connection is closed after each request/response interaction.
- Each subsequent request from the same client to the same server involves the setting up and tearing down of an additional TCP connection. Fig. 7.3.10 shows non-persistent HTTP connection.

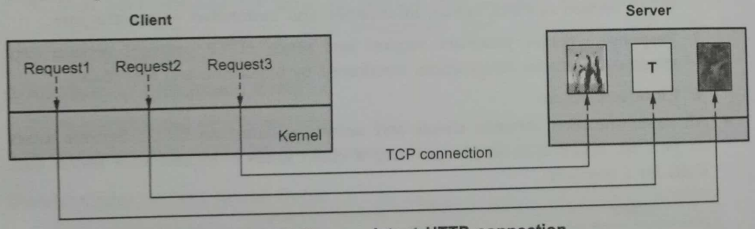


Fig. 7.3.10 Non-persistent HTTP connection

- Connection steps :
  1. Browser opens TCP connection to port 80 of server (handshake)
  2. Browser sends http request message
  3. Server receives request, locates object, sends response
  4. Server closes TCP connection
  5. Browser receives response, parses object
  6. Browser repeats steps 1-5 for each embedded object

Disadvantages of non-persistent

1. TCP processing and memory resource wasted in the server and the client.

2. It requires delay of 2 RTT associated with the transfer of each object.
3. Each TCP connection setup involves the exchange of three segments between client and server machines.

Persistent Connection

- HTTP 1.1 made persistent connections the default mode. The server now keeps the TCP connection open for a certain period of time after sending a response.
- This enables the client to make multiple requests over the same TCP connection and hence avoid the inefficiency and delay of the non-persistent mode. Fig. 7.3.11 shows persistent HTTP connection.

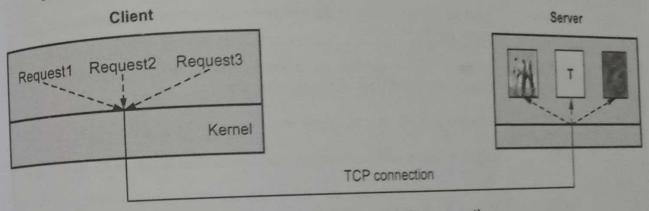


Fig. 7.3.11 Persistent HTTP connection

- There are two versions of persistent connections : Without pipelining and with pipelining.
- Steps for persistent connection :
  1. Browser opens TCP connection to port 80 of server (handshake)
  2. Browser sends http request message
  3. Server receives request, locates object, sends response
  4. Browser receives response, parses object
  5. Browser repeats steps 2 - 4 for each embedded object
  6. TCP connection closes on demand or timeout

Advantages of Persistent Connection

1. CPU time saved in routers and hosts
2. HTTP requests and responses can be pipelined on a connection
3. Network congestion is reduced
4. Latency on subsequent requests is reduced

7.3.3.2 HTTP Methods

- Request message defines several kinds of messages referred to as methods.

Sr. No.	Method	Purposes
1.	GET	Used when the client wants to reterieve a document from the server. Server responds with the contents of the document.
2.	HEAD	Used when client wants some information about a document but not the document itself.
3.	POST	Used by the client to provide some information to the server i.e. input to the server.
4.	PUT	Used by the client to provide a new or replacement document to be stored on the server.
5.	PATCH	Similar to PUT except that the request contains a list of differences that should be implemented in the existing file.
6.	DELETE	Removes a document on the server.
7.	COPY	Copies a files to another location. URL gives the location of the source file.
8.	MOVE	Move a file to another location.
9.	LINK	Creates a link or links from a document to another location.
10.	UNLINK	UNLINK method deleted links created by the LINK method.
11.	OPTION	This method is used by the client to ask the server about available options.

- The GET method requests the server to send the page suitably encoded in MIME. The HEAD method just asks for the message header, without the actual page.
- The PUT method is the reverse of GET : Instead of reading the page, it writes the page. This method makes it possible to build a collection of web pages on a remote server.
- The LINK and UNLINK methods allow connections to be established between existing pages or other resources.

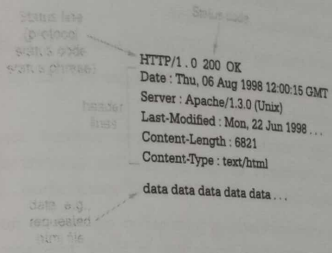
HTTP Messages

HTTP messages are two types.

1. Request 2. Response

- Both message type use same format. Request message consists of a request line, headers and sometimes a body. An HTTP request must specify the host name (and possibly port) for which the request is intended.

- Response message format is shown below :



Status Code :

- The status code is a three-digit integer, and the first digit identifies the general category of response:
  a) **1xx** indicates an informational message
  b) **2xx** indicates success of some kind
  c) **3xx** redirects the client to another URL
  d) **4xx** indicates an error on the client's part
  e) **5xx** indicates an error on the server's part

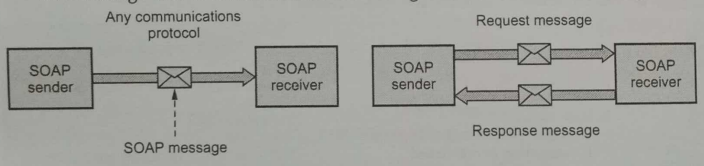
7.3.3.3 Difference between Persistent and Non-persistent HTTP

Sr. No.	Persistent HTTP	Non-persistent HTTP
1.	Peristent version in 1.1.	Non-persistent HTTP version is 1.0.
2.	IT uses on RTT.	It uses two RTT.
3.	TCP connection is not closed	TCP connection is closed after every request-response.
4.	Client make multiple request over the same TCP connection.	Client make multiple request over the multiple tCP connection.
5.	It is default mode.	It is not default mode.
6.	Request methods are GET, HEAD, POST, DELETE, TRACE and OPTIONS.	Request methods used are GET, POST and HEAD.

7.3.3.4 Simple Object Access Protocol

- SOAP is the standard messaging protocol used by web services. SOAP codifies the use of XML as an encoding scheme for request and response parameters using HTTP as a means for transport.

- SOAP is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment.
- SOAP can be used over any transport protocol such as TCP, HTTP, and SMTP.
- SOAP is based on message exchanges. Messages are seen as envelopes where the application encloses the data to be sent.
- A SOAP message consists of an <Envelope> element containing an optional <Header> and a mandatory <Body> element. The contents of these elements are application defined and not a part of the SOAP specification.
- SOAP <Header> contains blocks of information relevant to how the message is to be processed. This helps pass information in SOAP messages that is not for the application but for the SOAP engine
- The SOAP <Body> is where the main end-to-end information conveyed in a SOAP message must be carried.
- A SOAP message travels along the message path from a sender to a receiver. All SOAP messages start with an initial sender, which creates the SOAP message, and end with an ultimate receiver. SOAP message consists of three parts: SOAP Envelope, SOAP Header (optional) and SOAP Body.
- The SOAP Envelope construct defines an overall framework for expressing **what** is in a message and **who** should deal with it. Fig. 7.3.12 shows SOAP messages.



(a) One way message (b) Request and response message

Fig. 7.3.12 SOAP Message

- SOAP supports two possible communication styles :
  a) Remote procedure call (RPC) and
  b) Document (or message).
- A remote procedure call (RPC)-style Web service appears as a remote object to a client application. The interaction between a client and an RPC-style Web service centers around a service-specific interface. Clients express their request as a method call with a set of arguments, which returns a response containing a return value. Fig. 7.3.13 shows RPC communication styles.

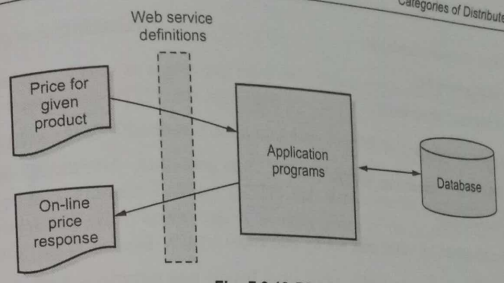


Fig. 7.3.13 RPC styles

- In the document-style of messaging, the SOAP <Body> contains an XML document fragment. The <Body> element reflects no explicit XML structure. The SOAP run-time environment accepts the SOAP <Body> element as it stands and hands it over to the application it is destined for unchanged. There may or may not be a response associated with this message. Fig. 7.3.14 shows document style of messaging.

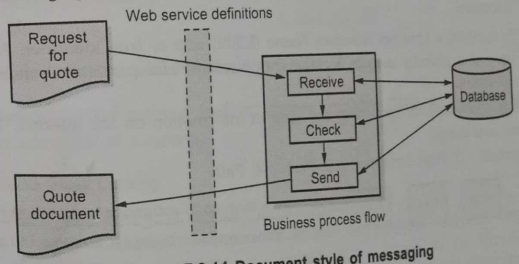


Fig. 7.3.14 Document style of messaging

SOAP and HTTP

- The typical binding for SOAP is HTTP. SOAP can use GET or POST.
- With GET, the request is not a SOAP message but the response is a SOAP message, with POST both request and response are SOAP messages.
- SOAP uses the same error and status codes as those used in HTTP so that HTTP responses can be directly interpreted by a SOAP module.

Advantages of SOAP

a) Simplicity
b) Portability
c) Firewall friendliness

d) Use of open standards

e) Interoperability

f) Universal acceptance.

Disadvantages of SOAP

a) Too much reliance on HTTP

b) Statelessness

c) Serialization by value and not by reference

7.3.4 Naming

- The Web uses a single naming system to refer to documents. The names used are called Uniform Resource Identifiers. Uniform Resource Identifier (URI) come in two forms :
  1. A Uniform Resource Locator (URL) is a URI that identifies a document by including information on how and where to access the document. In other words, a URL is a location-dependent reference to a document.
  2. In contrast, a Uniform Resource Name (URN) acts as true identifier. A URN is used as a globally unique, location-independent, and persistent reference to a document.
- URL is a standard for specifying any kind of information on the internet. The URL define four things.
  1. Method 2. Host computer 3. Port 4. Path

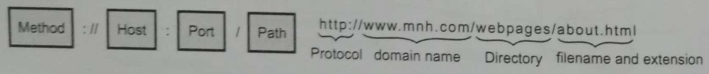


Fig. 7.3.15 URL

- The term 'URL' or "Universal Resource Locator" is not used in standards anymore. It generally means a URI that contains a domain-name but it is historical only. A *URI* identifies a Resource. A URI only comes into existence when it is bound to a Resource. A Resource is defined as anything that is identified by a *URI*.
- A URI can be further classified as a locator, a name, or both. The term "Uniform Resource Locator" (URL) refers to the subset of URI that identify resources via a representation of their primary access mechanism (e.g., their network "location"), rather than identifying the resource by name or by some other attribute(s) of that resource.
- The term "Uniform Resource Name" (URN) refers to the subset of URI that are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable.

7.3.5 Synchronization

- **Problem :** There is a growing need for collaborative auditing of Web documents, but bare-bones HTTP can't help here.
- **Solution :** Web Distributed Authoring and Versioning.
- Web Distributed Authoring and Versioning (WebDAV) is an extension of the Hypertext Transfer Protocol (HTTP) that allows clients to perform remote Web content authoring operations. A working group of the Internet Engineering Task Force (IETF) defined WebDAV.
- It supports exclusive and shared write locks, which operate on entire documents. A lock is passed by means of a lock token; the server registers the client(s) holding the lock. Clients modify the document locally and *post* it back to the server along with the lock token
- List of some of the commercial applications that currently support the WebDAV protocol :
  1. Apache Tomcat Server
  2. Microsoft Windows Server
  3. MAC OS X Server
  4. Groupware
  5. Linux
  6. IBM AS/400

7.3.6 Consistency and Replication

- Replication and caching used for system scalability. Data are replicated to increase the reliability of a system.

7.3.6.1 Web Proxy Caching

- Client stores the cache memory in two places.
  1. **Simple caching method :** Most of the browser supports this method. Whenever a document is fetched it is stored in the browser's cache from where it is loaded the next time. Clients can generally configure caching by indicating when consistency checking should take place.
  2. **Web proxy method :** Web proxy accepts requests from local clients and passes these to Web servers. When a response comes in, the result is passed to the client. The advantage of this approach is that the proxy can cache the result and return that result to another client, if necessary. Web proxy can implement a shared cache.
- Proxy caching differs substantially from the traditional ones used in processors. Two main features of the WWW applications that differ from the traditional caching are non-uniformity of the object sizes and non-uniformity of the cost of cache misses.

- Proxy cache works on the principle of localities of reference. Fig. 7.3.16 shows web proxy cache.

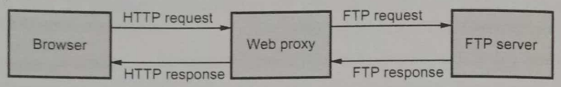


Fig. 7.3.16 Web proxy cache

Co-operative caching

- In co-operative caching mechanisms, a group of caches work together by collectively pooling their memory resources to provide a larger proxy cache. These co-operating caches can also be centrally managed by a server.
- To search in the cooperative cache, the proxy forwards the requested URL to a mapping server. The use of a central mapping service distinguishes the CRISP cache from other cooperative Internet caches.

7.3.6.2 Replication for Web Hosting Systems

- Replication is a technique that allows improving the quality of distributed services. Replication can lead to reduced client latency and network traffic by redirecting client requests to a replica closest to that client. It can also improve the availability of the system, as the failure of one replica does not result in entire service outage.
- Replication is a well-known technique to improve the accessibility of Web sites. It generally offers reduced client latencies and increases a site's availability. However, applying replication techniques is not trivial, and various Content Delivery Networks (CDNs) have been created to facilitate replication for digital content providers.
- These are systems that host the documents of a website and manage replication automatically.

Content Distribution Networks

- CDN is a collaborative collection of network elements spanning the Internet, where content is replicated over several mirrored Web servers in order to perform transparent and effective delivery of content to the end users. Collaboration among distributed CDN components can occur over nodes in both homogeneous and heterogeneous environments.
- CDN functions :
  1. Request redirection and content delivery services, to direct a request to the closest suitable CDN cache server using mechanisms to bypass congestion.
  2. Content outsourcing and distribution services, to replicate and/or cache content from the origin server to distributed Web servers.

  3. Content negotiation services, to meet specific needs of each individual user.
  4. Management services, to manage the network components, to handle accounting, and to monitor and report on content usage
- The three main entities in a CDN system are content provider, CDN provider, and end users.
- Content provider or customer is one who delegates the URL name space of the Web objects to be distributed. The origin server of the content provider holds those objects.
- CDN provider is a proprietary organization or company that provides infrastructure facilities to content providers in order to deliver content in a timely and reliable manner.
- End users or clients are the entities who access content from the content provider's Web site. Fig. 7.3.17 shows feedback control loop for a replica hosting system.

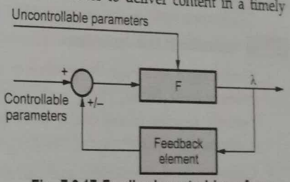


Fig. 7.3.17 Feedback control loop for a replica hosting system

- CDN providers use caching and replica servers located in different geographical locations to replicate content. CDN cache servers are also called edge servers or surrogates. The edge servers of a CDN are called Web cluster as a whole.
- CDNs distribute content to the edge servers in such a way that all of them share the same content and URL. Client requests are redirected to the nearby optimal edge server and it delivers requested content to the end users. Thus, transparency for users is achieved.
- **Akamai** is one of the largest CDNs currently deployed, with tens of thousands of replica servers placed all over the Internet. To a large extent, Akamai uses well-known technology to replicate content, notably a combination of DNS-based redirection and proxy-caching techniques.
- There are essentially three different kinds of aspects related to replication in Web hosting systems :
  1) Metric estimation
  2) Adaptation triggering
  3) Taking appropriate measures :
     A. Replica placement decisions
     B. Consistency enforcement
     C. Client-request routing.

Metric estimation :

1. Latency : Time is measured for an action. Fetching a document is an example of latency.
2. Spatial metrics : it consists of measuring the distance between nodes in terms of network level routing hops.
3. Consistency metrics : tell user to what extent a replica is deviating from its master copy.
4. Financial metrics are closely related to the actual infrastructure of the Internet. For example, most commercial CDNs place servers at the edge of the Internet, meaning that they hire capacity from ISPs directly servicing end users.

7.3.6.3 Replication of Web Applications

- Generating a Web page in response to every request takes more time than simply fetching static HTML pages from a server. Dynamic generation of a Web page typically requires issuing one or more queries to a database, so access times to the database can easily get out of hand when the request load is high.
- Content-delivery networks such as Akamai do this by deploying edge servers around the Internet to locally cache Web pages and then deliver them to clients. By delivering pages from edge servers located close to the clients, CDNs reduce each request's network latency. Page-caching techniques work well if the same cached HTML page can answer many requests to a particular Web site.

Content-Aware Data Caching (CAC)

- Instead of maintaining full copies of the database at each edge server, content-aware caching systems cache database query results as the application code issues them.
- Each edge server maintains a partial copy of the database, and each time the application running at the edge issues a query, the edge-server database checks if it contains enough data locally to answer the query correctly. This process is called a query **containment check**.
- If the containment check result is positive, the edge server can execute the query locally; otherwise, it must be sent to the central database.
- Examples of CAC systems include DBCache and DBProxy.

Content-Blind Data Caching (CBC)

- In Content-Blind Data Caching, edge servers don't need to run a database at all. They store the results of remote database queries independently.

7.4 Distributed Coordination-based System

- Separation between coordination and computation is possible by using coordination-based systems.
- Couplings make a distinction between
  a) Temporal coupling : Are cooperating/communicating processes alive at the same time ?
- Referential coupling : Do cooperating/communicating processes know each other explicitly ?
- We are trying to separate computation from coordination; coordination deals with all aspects of communication between processes, as well as their cooperation.
- Fig. 7.4.1 shows taxonomy of coordination models.

Referential \ Temporal	Coupled	Decoupled
Coupled	Direct	Mailbox
Decoupled	Meeting oriented	Generative communication

Fig. 7.4.1

- When processes are temporally and referentially coupled, coordination takes place in a direct way, referred to as direct coordination.
- The combination of referentially decoupled and temporally coupled systems form the group of models for meeting-oriented coordination.
- Meeting-based systems are often implemented by means of events, like the ones supported by object-based distributed systems.
- In publish/subscribe systems, processes can subscribe to messages containing information on specific subjects, while other processes produce (i.e., publish) such messages. Most publish/subscribe systems require that communicating processes are active at the same time; hence, there is a temporal coupling.
- However, the communicating processes may otherwise remain anonymous.
- Fig. 7.4.2 shows the principle of exchanging data items between publishers and subscribers.
- A data item is described by means of attributes. When made available, it is said to be published.
- A process interested in reading an item, must provide a subscription : A description of the items it wants. Middleware must match published items and subscriptions.

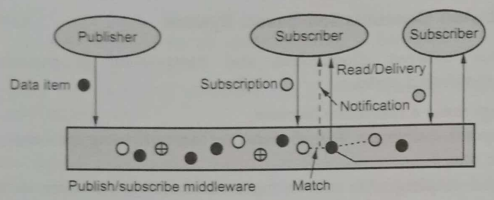


Fig. 7.4.2

Jini and Java Space

- Jini is a distributed system that consists of a mixture of different but related elements. It is strongly related to the Java programming language.
- Jini provides temporal and referential uncoupling of processes through a Linda-like coordination system called JavaSpaces. A JavaSpace is a shared dataspace that stores tuples representing a typed set of references to Java objects.
- In a distributed application, JavaSpaces technology acts as a virtual space between providers and requestors of network resources or objects. This allows participants in a distributed solution to exchange tasks, requests, and information in the form of Java technology-based objects.
- JavaSpaces technology provides developers with the ability to create and store objects with persistence, which allows for process integrity
- Multiple JavaSpaces may coexist in a single Jini system. Tuples are stored in serialized form. Fig. 7.4.3 shows general organization of a JavaSpace in Jini.

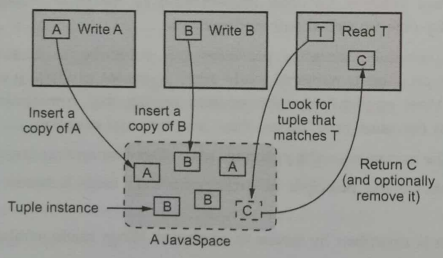


Fig. 7.4.3

- A tuple is put into a JavaSpace by means of a write operation, which first marshals the tuple before storing it. Each time the write operation is called on a tuple, another marshaled copy of that tuple is stored in the JavaSpace.

- To read a tuple instance, a process provides another tuple that it uses as a template for matching tuple instances as stored in a JavaSpace.
- Write : A copy of a tuple (tuple instance) is stored in a JavaSpace
- Read : A template is compared to tuple instances; the first match returns a tuple instance
- Take : A template is compared to tuple instances; the first match returns a tuple instance and removes the matching instance from the JavaSpace

7.4.1 Communication : Content-based Routing

- When a coordination-based system is built across a wide-area network, we need an efficient routing mechanism.
- Content-based routing is used to examine messages and route them to the correct channel or destination depending on a message's content. Content-based routing systems are typically built around two types of entities : routers and services.
- Routers, as their name suggests, route messages. They examine the content of the messages they receive, apply rules to that content, and forward the messages as the rules dictate. In content-based routing, the system is assumed to be built on top of a point-to-point network in which messages are explicitly routed between nodes.

7.5 Fill in the Blanks

Q.1 In Globe, each distributed shared object is assigned a globally unique object identifier, which is a _____ string.

Q.2 The combination of referentially decoupled and temporally coupled systems form the group of models for _____ coordination.

Q.3 When processes are temporally and referentially coupled, coordination takes place in a direct way, referred to as _____.

Q.4 In GFS, Files are divided into fixed-sized chunks of _____.

Q.5 A _____ is a typed set of references to objects.

Q.6 Coordination model makes use of subject-based addressing, leading to what is known as a _____ architecture.

Q.7 The combination of referentially decoupled and temporally coupled systems form the group of models for _____ coordination.

Q.8 Tuples are stored in _____ form.

Q.9 _____ coding is a well-known technique by which a _____ is partitioned into n fragments which are subsequently recoded into n > m fragments.

Q.10 GSpace is a distributed _____ system that is built on top of Java Spaces.

Q.11 A client interacts with web servers through a special application known as a ---------- (browser).

Q.12 The runtime environment, known as the _____ is a library that provides a platform independent interface for file handling, networking, threads.

Q.13 HTTP stands for _____.

Q.14 HTTP is a _____ protocol

Q.15 HTTP use port number _____ connection.

Q.16 The NFS client implements the NFS file system operation as _____ to the server.

Q.17 NFS also supports hard links as well as _____, like any UNIX file system

7.6 Multiple Choice Questions

Q.1 SOAP stands for _____.

a Simple object access protocol b Simple object access platform
c System object access protocol d Simple object access port

Q.2 _____ consists of a single master and multiple chunk servers and is accessed by multiple clients.

a Network file system b Andrew file system
c Google file system d Coda file system

Q.3 Erasure coding is a well-known technique by which a _____ is partitioned into n fragments which are subsequently recoded into n > m fragments.

a file b directory
c file and directory d none

Q.4 DNS is a _____ application that identifies each host on the Internet with a unique user friendly name.

a peer to peer b client/server
c web d internet

Q.5 FTP uses two different port connection _____.

a port 25 and port 53 b port 20 and port 25
c port 21 and port 25 d port 20 and port 21

Q.6 XML stands for eXtensible Markup Language.

a eXtensible Modern Language b eXtensible Markup List
c eXtensible Markov Language d None of these

Q.7 Web browser is a software program that interprets and displays the contents of _____ web pages.

a XML b HTML
c static d dynamic

Q.8 Most SOAP messages are sent over _____.

a UDP b TCP
c SNMP d HTTP

Q.9 Jini provides temporal and referential decoupling of processes through a coordination system called _____, derived from Linda.

a JavaSpaces b Java
c Tuple d None

Q.10 In content-based routing, the system is assumed to be built on top of a _____ network in which messages are explicitly routed between nodes.

a Multipoint b Single point
c Point to point d All of these

Q.11 Every GSpace kernel has a local dataspace, called a _____, which is implemented as a full-fledged, non-distributed version of JavaSpaces.

a Fragment b Slice
c Tuple d Space

Q.12 Availability is computed in terms of the mean time to failure and the _____.

a mean time to repair b mean time to recovery
c mean time to fragment d None

Answer Keys for Fill in the Blanks

Q.1	256-bit	Q.2	meeting-oriented	Q.3	direct coordination
Q.4	64 MB	Q.5	tuple	Q.6	publish-subscribe
Q.7	meeting oriented	Q.8	serialized	Q.9	Erasure
Q.10	coordination-based	Q.11	browser	Q.12	Apache Portable Runtime

Q.13	Hyper Text Trasnfer Protocol	Q.14	stateless	Q.15	80
Q.16	RPC	Q.17	symbolic links		

Answer Keys for Multiple Choice Quesions

Q.1	a	Q.2	c	Q.3	a
Q.4	b	Q.5	d	Q.6	a
Q.7	b	Q.8	d	Q.9	a
Q.10	c	Q.11	b	Q.12	a

❑❑❑

Solved Model Question Paper

(As per New Syllabus)

Distributed System

Semester - VII (CE / CSE) Professional Elective - V

Time : 2 Hours] [Total Marks : 56

Instructions :

1. **Attempt any FOUR questions out of Eight questions.**
2. **Make suitable assumptions wherever necessary.**
3. **Figures to the right indicate full Marks.**

Q.1 a) *What is distributed system ? List its advantages.* **[Refer Section 1.1]** [3]

b) *Draw and explain process state diagram.* **[Refer Section 1.4.2]** [4]

c) *What is transparency ? Explain various transparency.* **[Refer Section 1.2.2]** [7]

Q.2 a) *Define RPC ? How stud is generated in RPC ?* **[Refer Section 2.8]** [3]

b) *Explain stream-oriented communication.* **[Refer Section 2.10]** [4]

c) *What is code migration ? Explain various reasons of code migration.* **[Refer Section 2.6]** [7]

Q.3 a) *What is LDAP ?* **[Refer Section 3.4.2]** [3]

b) *Explain desirable features of good naming system.* **[Refer Section 3.1.1.1]** [4]

c) *Explain name space, name resolution and name server.* **[Refer Section 3.3]** [7]

Q.4 a) *What is event ordering ?* **[Refer Section 4.3.1]** [3]

b) *What is physical clock ? Explain.* **[Refer section 4.1.3]** [4]

c) *Explain the bully algorithm.* **[Refer section 4.6.1]** [7]

Q.5 a) *What is replica management ?* **[Refer section 5.4]** [3]

b) *Explain two phase commit protocol.* **[Refer section 5.10.2]** [4]

c) *Explain the following two consistency model.*
1) Sequential consistency model 2) Week consistency model.
[Refer section 5.2] [7]

Q.6 a) *Explain security policies.* **[Refer section 6.1.1]** [3]

b) *What is Kerberos ? Explain.* **[Refer section 6.3.4]** [4]

c) *What is DES ? Explain in details.* **[Refer section 6.2.1]** [7]

Q.7 **a)** *What is Apache web server ?* **[Refer section 7.3.2]** [3]

b) *Explain enterprise java bean.* **[Refer section 7.1.1]** [4]

c) *Draw and explain NFS architecture.* **[Refer section 7.2.3]** [7]

Q.8 **a)** *What is checkpoint ?* **[Refer section 5.11.4]** [3]

b) *Explain message oriented communication.* **[Refer section 2.9]** [4]

c) *Explain ring based algorithm for mutual exclusion.* **[Refer section 4.4]** [7]

❑❑❑