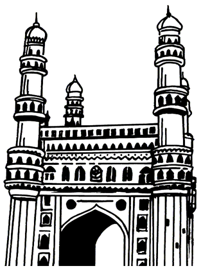
Rahul's ✔
Topper's Voice



MCA

II Year III Sem

(Osmania University)

Latest Edition

DISTRIBUTED SYSTEMS

- **Study Manual**
- **Important Questions**
- **Solved Model Papers**

Price
` . ~~299-00~~
` . 249-00

- by -

WELL EXPERIENCED LECTURER

Since 1986

Rahul Publications™

Hyderabad. Cell : 9391018098, 9505799122.

MCA

II Year III Sem

DISTRIBUTED SYSTEMS

Sole Distributors : **Cell : 9391018098, 9505799122**

VASU BOOK CENTRE

Shop No. 2, Beside Gokul Chat, Koti, Hyderabad.

Maternity Hospital Opp. Lane, Narayan Naik Complex, Koti, Hyderabad.
Near Andhra Bank, Subway, Sultan Bazar, Koti, Hyderabad -195.

CONTENTS

DISTRIBUTED SYSTEMS

SYLLABUS

UNIT – I

Introduction: definition of distributed system, goals, types of distributed systems.Architectures:architectural styles, system architectures, architectures versus middleware, self-management in distributed systems.Processes:threads, virtualization, clients, servers, code migration.

UNIT – II

Communication: Remote Procedure Call, Message-Oriented Communication, Stream-Oriented Communication, Multicast Communication.Naming:names, identifiers, and addresses, flat naming, structured naming, attribute based naming.Synchronization:clock synchronization, logical clocks, mutual exclusion, global positioning of nodes, election algorithms.

UNIT – III

Consistency and Replication:introduction, data-centric consistency models, client-centric consistency models, replica management, consistency protocols.Fault Tolerance:introduction, process resilience, reliable clientserver communication, reliable group communication, distributed commit, recovery.Security: introduction, secure channels, access control, security management.

UNIT – IV

Distributed Object–Based Systems: architecture, processes, communication, naming, synchronization, consistency and replication, fault tolerance, security.Distributed File Systems: architecture, process, communication, naming, synchronization, consistency and replication, fault tolerance, security.Distributed Webbased Systems:architecture, process, communication, naming, synchronization, consistency and replication, fault tolerance, security.

Contents

Topic **Page No.**

Important Questions

UNIT - I

Short Questions

1. What is distributed system?
2. What are the goals of a distributed systems?
3. What is architectural style?
4. What are called as super peers?
5. Write about middleware sytems.
6. Write a short note on Processes.
7. Write a note on threads.
8. What is virtualization ?
9. What is Thin- Client Network Computing?
10. Differentiate between (temporary) session state and permanent state.

Long Questions

1. Explain various scaling techniques.
2. What is architectural style? Explain different types of architectural styles.
3. Write about centralised anddecentralized architectures.
4. Write about various Hybrid Architectures.
5. Write about the Feed back control model for self- management in distributed systems
6. What are the uses of threads in distributed environment Write, how to implement the threads in distributed environment.
7. What is the role of virtualization to solve the problems in distributed systems :

8 Write about stateless and statefull servers.

9. What are the reasons for code migration. write about the code migration.
10. Explain the migration process in heterogeneous systems.

UNIT - II

Short Questions

1. Write about the communication in distributed systems
2. What is RPC?
3. What is Message Oriented Communication?
4. Write about Overlay Networks
5. Write about names in distributed environment
6. What are identifiers in distributed environment.
7. Write a note on addresses in distributed systems
8. Write a note on Flat Naming service.
9. What is clock synchronization in DS.
10. Write about Geometric Overlay Network

Long Questions

1. Explain about Replica Management in Distributed systems.
2. Explain about consistency protocols.
3. What is failure model? Explain different types of failures in distributed systems
4. Explain what is process resilience? and how fault tolerance can be achieved by using fault tolerance.
5. Explain point to point communication in distributed systems
6. Write about two phase commit protocol
7. Explain about three phase commit protocol
8. What is message logging ? When should we use message logging?
9. Explain various Cryptosystems.
10. What are secure channels? Write about them

UNIT - III

Short Questions

1. Write about the reasons for replication
2. What is dependability
3. What is failure model?
4. Write about Failure Masking by Redundancy
5. What is process resilience?
6. Write about failure detection process
7. Write about backward recovery.
8. Write about forward recovery.
9. Write about Security in distributed systems
10. Write about the design issues when considering the security in distributed systems

Long Questions

1. Explain about Replica Management in Distributed systems.
2. Explain about consistency protocols.
3. What is failure model? Explain different types of failures in distributed systems?
4. Explain what is process resilience? and how fault tolerance can be achieved by using fault tolerance.
5. Explain point to point communication in distributed systems.
6. Write about two phase commit protocol.
7. Explain about three phase commit protocol.
8. What is message logging ? When should we use message logging?
9. Explain various Cryptosystems.
10. What are secure channels? Write about them.

UNIT - IV

Short Questions

1. Write about processes or object Servers
2. Write a note on ORB
3. What is Synchronization in Distributed systems.
4. What are distributed file systems? Explain
5. What is the role of Processes in DFS
6. Write a note on Naming service in DFS
7. What are File Handles in DFS
8. What are Web Services
9. Write a note on Web Browser
10. Write a note on SOAP.

Long Questions

1. Explain RMI and its Architecture.
2. Explain Replication Framework.
3. Explain about Sun Network File Systmes.
4. Write about , how Synchronization takes place for the file systems
5. How caching and replication is useful in DFS, Explain.
6. Explain how to handle Byzantine Failures in DFS.
7. Explain about distributed web based systems and its architecture.
8. Explain about HTTP protocol.
9. Write about synchronization in distributed web based systems.
10. Write about Web Proxy Caching.

FACULTIES OF COMPUTER SCIENCE

M.Sc IV Semester Examination

Model Paper - I

DISTRIBUTED SYSTEMS

Time : 3 Hours] [Max. Marks : 80

PART - A (8 × 4 = 32 M)

Answer all the Questions

ANSWERS

1.	Write about middleware sytems.	**(Unit-I, Q.No. 12)**
2.	Write a note on threads.	**(Unit-I, Q.No. 17)**
3.	What is RPC?	**(Unit-II, Q.No. 4)**
4.	Write about Overlay Networks	**(Unit-II, Q.No. 8)**
5.	What is dependability	**(Unit-III, Q.No. 6)**
6.	Write about forward recovery.	**(Unit-III, Q.No. 16)**
7.	What is Synchronization in Distributed systems.	**(Unit-IV, Q.No. 9)**
8.	What are Web Services	**(Unit-IV, Q.No. 27)**

PART - B (4 × 12 = 48 M)

Answer any Four of the following questions

9a)	What are the uses of threads in distributed environment 19. Write, how to implement the threads in distributed environment	**(Unit-I, Q.No. 18 & 19)**
b)	Explain the migration process in heterogeneous systems.	**(Unit-I, Q.No. 36)**
10a)	Explain about Stream oriented Communciation.	**(Unit-II, Q.No. 8)**
b)	Explain about mutual exclusion. 24. Write about Token Ring Algorithm for mutual Exclusion	**(Unit-II, Q.No. 23, 24)**
11a)	Write about two phase commit protocol.	**(Unit-III, Q.No. 14)**
b)	Write about various methods of access control.	**(Unit-III, Q.No. 26)**
12a)	Explain how to handle Byzantine Failures in DFS.	**(Unit-IV, Q.No. 23)**
b)	Write about Web Proxy Caching.	**(Unit-IV, Q.No. 37)**

FACULTIES OF COMPUTER SCIENCE

M.Sc IV Semester Examination

Model Paper - II

DISTRIBUTED SYSTEMS

Time : 3 Hours] [Max. Marks : 80

PART- A (8 × 4 = 32 M)

Answer all the Questions

		ANSWERS
1.	What are called as super peers?	**(Unit-I, Q.No. 10)**
2.	What is virtualization ?	**(Unit-I, Q.No. 22)**
3.	What is Message Oriented Communication?	**(Unit-II, Q.No. 7)**
4.	What is clock synchronization in DS.	**(Unit-II, Q.No. 18)**
5.	What is failure model?	**(Unit-III, Q.No. 7)**
6.	What is process resilience?	**(Unit-III, Q.No. 9)**
7.	What is the role of Processes in DFS	**(Unit-IV, Q.No. 17)**
8.	Write a note on SOAP.	**(Unit-IV, Q.No. 31)**

PART - B (4 × 12 = 48 M)

Answer any Four of the following questions

9a)	Explain various scaling techniques	**(Unit-I, Q.No. 3)**
b)	Write about decentralized architectures.	**(Unit-I, Q.No. 9)**
10a)	Write about middleware protocols and types of communication supports by middleware protocols	**(Unit-II, Q.No. 3)**
b)	Write about physical clocks. Explain NTP algorithm in physical clocks.	**(Unit-II, Q.No. 18 & 19)**
11a)	What are data centric consistency models.	**(Unit-III, Q.No. 2)**
b)	Explain various Cryptosystems.	**(Unit-III, Q.No. 23)**
12a)	Explain RMI and its Architecture	**(Unit-IV, Q.No. 4)**
b)	Explain about distributed web based systems and its architecture.	**(Unit-IV, Q.No. 25)**

FACULTIES OF COMPUTER SCIENCE

M.Sc IV Semester Examination

Model Paper - III

DISTRIBUTED SYSTEMS

Time : 3 Hours] [Max. Marks : 80

PART- A (8 × 4 = 32 M)

Answer all the Questions

		ANSWERS
1.	Write a short note on Processes.	**(Unit-I, Q.No. 16)**
2.	What is Thin- Client Network Computing?	**(Unit-I, Q.No. 26)**
3.	Write about the communication in distributed systems.	**(Unit-II, Q.No. 1)**
4.	Write a note on Flat Naming service.	**(Unit-II, Q.No. 13)**
5.	Write about Failure Masking by Redundancy.	**(Unit-III, Q.No. 8)**
6.	Write about backward recovery.	**(Unit-III, Q.No. 16)**
7.	Write about processes or object Servers.	**(Unit-IV, Q.No. 2)**
9.	What are File Handles in DFS.	**(Unit-IV, Q.No. 20)**

PART - B (4 × 12 = 48 M)

Answer any Four of the following questions

9a)	Write about the Feed back control model for self- management in distributed systems.	**(Unit-I, Q.No. 15)**
b).	What is the role of virtualization to solve the problems in distributed systems.	**(Unit-I, Q.No. 23)**
10a)	Explain ISO OSI reference Model.	**(Unit-II, Q.No. 2)**
b)	What is RPC? Explain the working of RPC .what are the issues of RPC. And Mention its advantages.	**(Unit-II, Q.No. 4 & 5)**
11a)	Explain point to point communication in distributed systems.	**(Unit-III, Q.No. 12)**
b)	Explain various Security Mechanisms.	**(Unit-III, Q.No. 22)**
12a)	Explain Replication Framework.	**(Unit-IV, Q.No. 11)**
b)	Write about, how Synchronization takes place for the file systems.	**(Unit-IV, Q.No. 21)**

UNIT I

Introduction: Definition of distributed system, goals, types of distributed systems. Architectures: architectural styles, system architectures, architectures versus middleware, self-management in distributed systems. Processes: threads, virtualization, clients, servers, code migration.

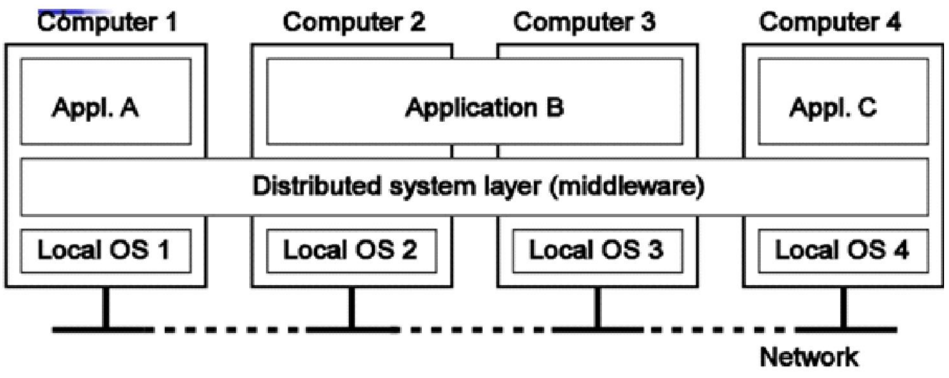
1.1 Introduction

1.1.1 Definition of Distributed System

Q1. What is distributed system?

Ans :

A distributed system is a network that consists of autonomous computers that are connected using a distribution middleware. They help in sharing different resources and capabilities to provide users with a single and integrated coherent network.



The key features of a distributed system are:

Components in the system are concurrent. A distributed system allows resource sharing, including software by systems connected to the network at the same time.

- The components could be multiple but will generally be autonomous in nature.
- A global clock is not required in a distributed system. The systems can be spread across different geographies.
- Compared to other network models, there is greater fault tolerance in a distributed model.
- Price/performance ratio is much better.

1.1.2 Goals

Q2. What are the goals of distributed systems?

Ans :

The four important goals that should be met for an efficient distributed system are as follows:

1. Connecting Users and Resources

- The main goal of a distributed system is to make it easy for users to acces remote resourses and to share them with others in a controlled way.
- It is cheaper to le a printer be shared by several users than buying and maintaining printers for each user.
- Collaborating and exchanging information can be made easier by connecting users and resource.

2. Transparency

- It is important for a distributed system to hide the location of its process and resource. A distributed system that can portray itself as a single system is said to be transparent.
- The various transparencies need to be considered are access, location, migration, relocation, replication, concurrency, failure and persistence.
- Aiming for distributed transparency should be considered along with performance issues.

3. Openness

- Openness is an important goal of distributed system in which it offers services according to standard rules that describe the syntax and semantics of those services.
- Open distributed system must be flexible making it easy to configure and add new components without affecting existing components.
- An open distributed system must also be extensible.

4. Scalable

- Scalability is one of the most important goals which are measured along three different dimensions.
- First, a system can be scalable with respect to its size which can add more user and resources to a system.
- Second, users and resources can be geographically apart.
- Third, it is possible to manage even if many administrative organizations are spanned.

Q3. Explain various scaling techniques.

Ans :

Scaling Techniques

Three techniques for scaling:

1. Hiding communication latencies
2. Distribution
3. Replication

Hiding communication latencies - important to achieving geographical scalability.

1. Try to avoid waiting for responses to remote service requests.

E.g.: when a service has been requested at a remote machine, an alternative to waiting for a reply from the server is to do other useful work at the requester's side.

Construct the requesting application in such a way that it uses only asynchronous communication.

2. Reduce the Overall Communication

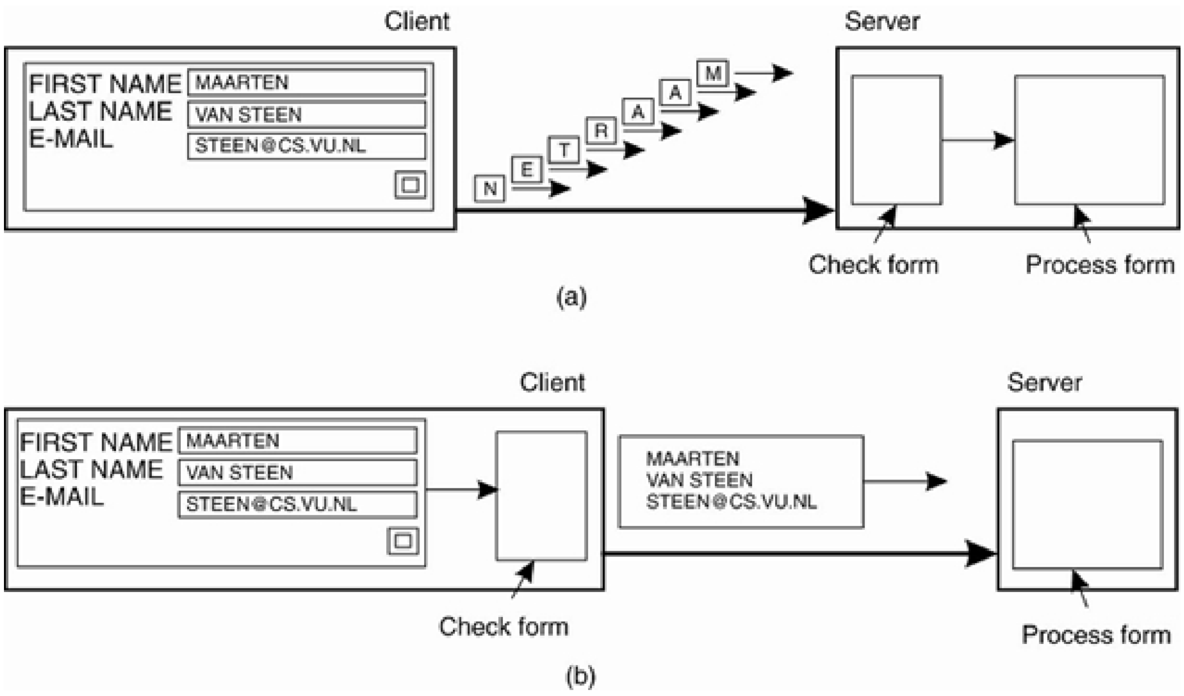
E.g.: in interactive applications when a user sends a request he will generally have nothing better to do than to wait for the answer.

Move part of the computation that is normally done at the server to the client process requesting the service.

- typical case - accessing databases using forms.

Ship the code for filling in the form, and possibly checking the entries, to the client, and have the client return a completed form - approach of shipping code is now widely supported by the Web in the form of Java applets and Javascript.

The difference between letting (a) a server or (b) a client check forms as they are being filled.



Distribution - splitting a component into smaller parts and spreading those parts across the system.

E.g.: Internet Domain Name System (DNS).

- ➢ The DNS name space is hierarchically organized into a tree of domains, which are divided into nonoverlapping zones
- ➢ Names in each zone are handled by a single name server.
- ➢ Resolving a name means returning the network address of the associated host.

e.g. the name nl.vu.cs.flits.

- ➢ Replication
  - ➢ Increases availability
  - ➢ Helps balance the load between components leading to better performance.

e.g. in geographically widely-dispersed systems - a copy nearby can hide much of the communication latency problems.

- ➢ Caching - special form of replication
  - ➢ Caching results in making a copy of a resource, generally in the proximity of the client accessing that resource.
  - ➢ Caching is a decision made by the client of a resource, and not by the owner of a resource.
  - ➢ Caching happens on demand whereas replication is often planned in advance.

Issues of caching and replication - multiple copies of a resource -> modifying one copy makes that copy different from the others -> leads to consistency problems.

- ➢ Weak consistency – e.g. a cached Web document of which the validity has not been checked for the last few minutes.
- ➢ Strong consistency – e.g. electronic stock exchanges and auctions.

1.1.3 Types of Distributed Systems

Q4. Explain about various types of distributed systems.

Ans :

According to the performance the distributed systems are categorised into three types:

- Distributed computing systems
- Distributed information systems
- Distributed pervasive systems
- Distributed computing systems

An important class of distributed systems is the one used for high-performance computing tasks.

It Is sub divided into two groups

- Cluster computing - the underlying hardware consists of a collection of similar workstations or PCs, closely connected by means of a high speed local-area network.
- Grid computing - consists of distributed systems that are often constructed as a federation of computer systems, where each system may fall under a different administrative domain,.

Cluster Computing Systems

Cluster computing is used for parallel programming in which a single program is run in parallel on multiple machines.

- Example of a cluster computer - Linux-based Beowulf clusters.
- Each cluster consists of a collection of compute nodes that are controlled and accessed by means of a single master node.

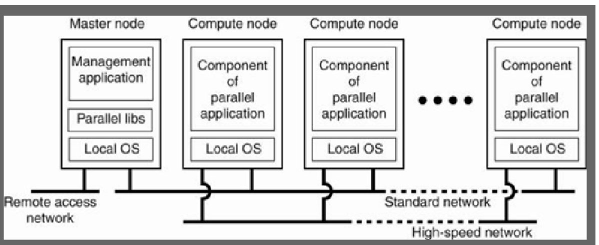
Master Node

- Handles the allocation of nodes to a particular parallel program.
- Maintains a batch queue of submitted jobs
- Provides an interface for the users of the system.
- The master runs the middleware needed for the execution of programs and management of the cluster.
- Middleware is formed by the libraries for executing parallel programs.

many of these libraries effectively provide only advanced message-based commu-nication facilities, but are not capable of handling faulty processes, security, etc.

o compute nodes - often need nothing else but a standard operating system.

An example of a cluster computing system.



Grid Computing Systems

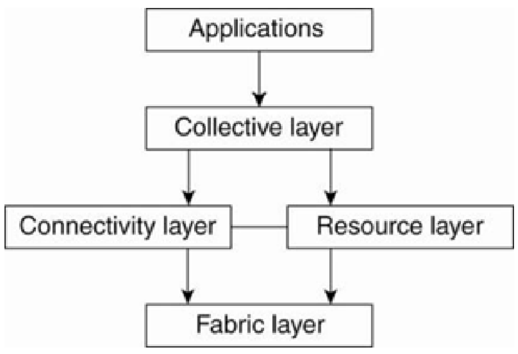
Grid computing systems have a high degree of heterogeneity: no assumptions are made concerning hardware, operating systems, networks, administrative domains, security policies, etc.

Key issue in a grid computing system is resources from different organizations are brought together to allow the collaboration of a group of people or institutions - forms a virtual organization.

- Members of the same virtual organization have access rights to the resources that are provided to that organization.
- Resources consist of compute servers (including supercomputers, possibly implemented as cluster computers), storage facilities, and databases. In addition, special networked devices such as telescopes, sensors, etc., can be provided as well.

Software required for grid computing evolves around providing access to resources from different administrative domains, and to only those users and applications that belong to a specific virtual organization.

A layered architecture for grid computing systems.



Four layer architecture:

1. fabric layer - provides interfaces to local resources at a specific site.

   ➢ Mterfaces are tailored to allow sharing of resources within a virtual organization.

   ➢ Provide functions for querying the state and capabilities of a resource, along with functions for actual resource management (e.g., locking resources).

2. (a) Connectivity layer - consists of communication protocols for supporting grid transactions that span the usage of multiple resources.

   ➢ Contains security protocols to authenticate users and resources.

   ➢ In many cases human users are not authenticated - programs acting on behalf of the users are authenticated.

2. (b) Resource layer - responsible for managing a single resource.

   ➢ Uses functions provided by the connectivity layer and calls directly the interfaces made available by the fabric layer.

   ➢ Responsible for access control, and hence will rely on the authentication performed as part of the connectivity layer.

3. Collective layer - handles access to multiple resources

   ➢ Consists of services for resource discovery, allocation and scheduling of tasks onto multiple resources, data replication, etc.

   ➢ May consist of many different protocols for many different purposes, reflecting the broad spectrum of services it may offer to a virtual organization.

4. Application layer - consists of the applications that operate within a virtual organization and which make use of the grid computing environment.

   Grid Middleware layer - collective, connectivity, and resource layers.

   ➢ Provide access to and management of resources that are potentially dispersed across multiple sites.

   ➢ Shift toward a service-oriented architecture in which sites offer access to the various layers through a collection of Web services

   ➢ Led to the definition of an alternative architecture known as the Open Grid Services Architecture (OGSA).

   ➢ Consists of various layers and many components, making it rather complex.

Distributed Information Systems

Many of the existing middleware solutions are the result of working with an infrastructure in which it was easier to integrate applications into an enterprise-wide information system.

Several levels at which integration took place:

1. A networked application simply consisted of a server running that application (often including a database) and making it available to remote programs, called clients.

   Such clients could send a request to the server for executing a specific operation, after which a response would be sent back. Integration at the lowest level would allow clients to wrap a number of requests, possibly for different servers, into a single larger request and have it executed as a distributed transaction.

The key idea was that all, or none of the requests would be executed.

2. As applications became more sophisticated and were gradually separated into independent components (notably distinguishing database components from processing components), it became clear that integration should also take place by letting applications communicate directly with each other.

Thre are two forms of distributed information systems available

- ➢ Transaction processing systems
- ➢ Enterprise application integration (EAI)

Transaction Processing Systems

- ➢ Focus on database applications - operations on a database are usually carried out in the form of transactions.
- ➢ Programming using transactions requires special primitives that must either be supplied by the underlying distributed system or by the language runtime system.

Example primitives for transactions **Pmitive scription**

BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

Properties of transactions (ACID):

1. Atomic: To the outside world, the transaction happens indivisibly.

- ➢ Ensures that each transaction either happens completely, or not at all
- ➢ If it happens, it happens in a single indivisible, instantaneous action.
- ➢ While a transaction is in progress, other processes (whether or not they are themselves involved in transactions) cannot see any of the intermediate states.

2. Consistent: The transaction does not violate system invariants.

If the system has certain invariants that must always hold, if they held before the transaction, they will hold afterward too e.g. a banking system - a key invariant is the law of conservation of money. After every internal transfer, the amount of money in the bank must be the same as it was before the transfer

3. Isolated: Concurrent transactions do not interfere with each other.

- ➢ Transactions are isolated or serializable
- ➢ If two or more transactions are running at the same time, to each of them and to other processes, the final result looks as though all transactions ran sequentially in some (system dependent) order.

4. **Durable:** Once a transaction commits, the changes are permanent.
   - Once a transaction commits, no matter what happens, the transaction goes forward and the results become permanent.
   - No failure after the commit can undo the results or cause them to be lost

   A nested transaction is constructed from a number of subtransactions,

   The top-level transaction may fork off children that run in parallel with one another, on different machines, to gain performance or simplify programming.

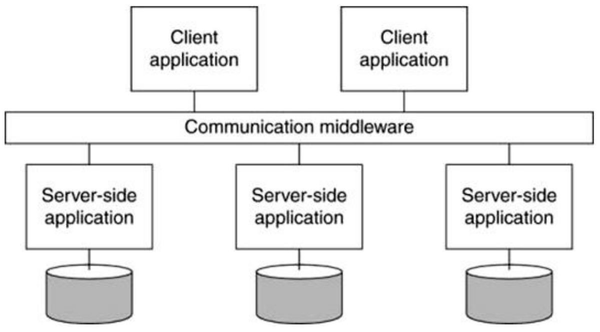
   Each of these children may also execute one or more subtransactions, or fork off its own children.

   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.

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.

   Problems with RPC and RMI
   - The caller and callee both need to be up and running at the time of communication.
   - They need to know exactly how to refer to each other.

   Solutions
   - Message-oriented middleware (MOM) - applications send messages to logical contact points, often described by means of a subject.
   - Publish/subscribe systems - applications can indicate their interest for a specific type of message, after which the communication middleware will take care that those messages are delivered to those applications.

Distributed Pervasive Systems

Above distributed systems characterized by their stability:

Nodes are fixed and have a more or less permanent and high-quality connection to a network.

Mobile and embedded computing devices

Instability is the default behavior

Distributed pervasive system - is part of the surroundings -> inherently distributed.

Features
- General lack of human administrative control
- Evices can be configured by their owners
- They need to automatically discover their environment and fit in as best as possible.

Three requirements for pervasive applications:

- ➢ Embrace contextual changes- a device must be continuously be aware of the fact that its environment may change all the time
- ➢ Encourage ad hoc composition-many devices in pervasive systems will be used in very different ways by different users – make it easy to configure the suite of applications running on a device
- ➢ Recognize sharing as the default.-devices generally join the system in order to access (and possibly provide) information

Examples

Home Systems

- ➢ Built around home networks
- ➢ Consist of one or more personal computers
- ➢ Integrate consumer electronics such as TVs, audio and video equipment, gaming devices, (smart) phones, PDAs, and other personal wearables into a single system.
- ➢ Now/Soon: all kinds of devices such as kitchen appliances, surveillance cameras, clocks, controllers for lighting, and so on, will all be hooked up into a single distributed system.

1.2 Architectures

1.2.1 Architectural Styles

Q5. What is architectural style? Explan different types of architectural styles.

Ans :

Architectural Styles refers the types of components, which are connected in the architecture. It refers the way that components are connected and the data exchanged between components, and how these elements are jointly configured into a system.

Software component - a modular unit with well-defined, required and provided interfaces that is **replaceable** within its environment.

Connector - a mechanism that mediates communication, coordination, or cooperation among components.

E.g.: a connector can be formed by the facilities for (remote) procedure calls, message passing, or streaming data.

Several styles have by now been identified, of which the most important ones for distributed systems are:

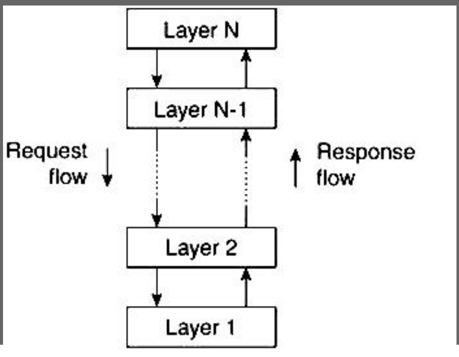
1. Layered architectures
2. Object-based architectures
3. Data-centered architectures
4. Event-based architectures

1. Layered Architectures

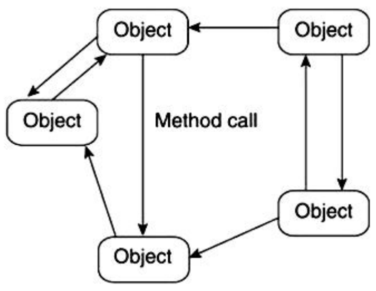
The basic idea for the layered style is simple: components are organized in a layered fashion where a component at layer Li is allowed to call components at the underlying layer Li-1, but not the other way around, as shown in Figure 2-1(a).

2. Object-based Architectures

In the object based architecture, each object corresponds a component and the components are connected through a (remote) procedure call mechanism.



(a)



(b)

3. Data-centered architectures

Processes communicate through a common (passive or active) repository

E.g.:

- Wealth of networked applications have been developed that rely on a shared distributed file system in which virtually all communication takes place through files.
- Web-based distributed systems are largely data centric: processes communicate through the use of shared Web-based data services.

4. Event-based architectures

- Processes communicate through the propagation of events

  E.g.: publish/subscribe systems

processes publish events after which the middleware ensures that only those processes that subscribed to those events will receive them.

advantage - processes are loosely coupled. In principle, they need not explicitly refer to each other. This is also referred to as being decoupled in space, or referentially decoupled.

1.2.2 System Architectures

Q6. Write about centralised architectures.

Ans :

Aim of Architectures: achieving distribution transparency.

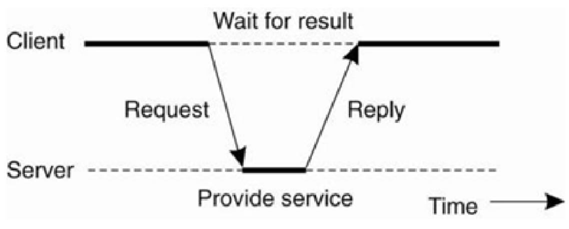
Centralized Architectures

Manage distributed system complexity - think in terms of clients that request services from servers.

Basic Client-server Model

Processes are divided into two groups:

1. A server is a process implementing a specific service, for example, a file system service or a database service.
2. A client is a process that requests a service from a server by sending it a request and subsequently waiting for the server's reply.
   - General interaction between a client and a server.



Communication

Implemented using a connectionless protocol when the network is reliable.

E.g.: Local area networks.

1. Client requests a service – packages and sends a message for the server, identifying the service it wants, along with the necessary input data.
2. The Server will always wait for an incoming request, process it, and package the results in a reply message that is then sent to the client.

Connectionless Protocol

- Describes communication between two network end points in which a message can be sent from one end point to another without prior arrangement.
- Device at one end of the communication transmits data to the other, without first ensuring that the recipient is available and ready to receive the data.
- The device sending a message sends it addressed to the intended recipient.
- More frequent problems with transmission than with connection-orientated protocols and it may be necessary to resend the data several times.
- Making the protocol resistant to occasional transmission failures is not trivial.
- The client cannot detect whether the original request message was lost, or that transmission of the reply failed.

If the reply was lost, then resending a request may result in performing the operation twice e.g. If operation was "transfer $10,000 from my bank account," then clearly, it would have been better that we simply reported an error instead.

When an operation can be repeated multiple times without harm, it is said to be idempotent

Often disfavored by network administrators because it is much harder to filter malicious packets from a connectionless protocol using a firewall.

E.g.: Connectionless protocols -The Internet Protocol (IP) and User Datagram Protocol (UDP) are connectionless protocols. Connection-oriented protocol (TCP/IP - the most common use of IP)

- Not appropriate in a local-area network due to relatively low performance
- Works fine in wide-area systems in which communication is inherently unreliable.
- E.g. virtually all Internet application protocols are based on reliable TCP/IP connections.

whenever a client requests a service, it first sets up a connection to the server before sending the request.

The server uses that same connection to send the reply message, after which the connection is torn down.

Problem: setting up and tearing down a connection is relatively costly, especially when the request and reply messages are small.

Q7. Write about application layering.

Ans :

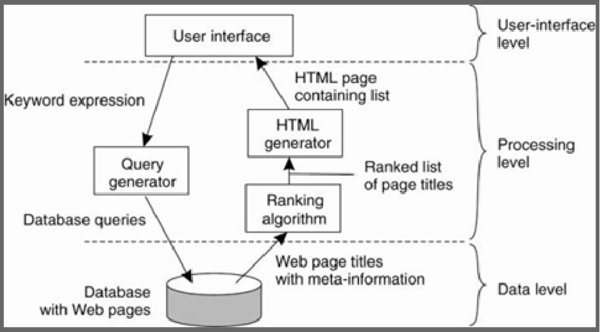
Application Layering

Since many client-server applications are targeted toward supporting user access to databases, **distinctions may be analyzed in a layered architectural style**:

1. The user-interface level - contains all that is necessary to directly interface with the user, such as display management.
   - Clients typically implement the user-interface level
   - Simplest user-interface program - character-based screen-the user's terminal does some local processing such as echoing typed keystrokes, or supporting form-like interfaces in which a complete entry is to be edited before sending it to the main computer
   - Simple GUI - pop-up or pull-down menus are used with many screen controls handled through a mouse instead of the keyboard.
   - Modern user interfaces offer considerably more functionality by allowing applications to share a single graphical window, and to use that window to exchange data through user actions.
2. The processing level - contains the applications.
   - Middle part of hierarchy logically placed at the processing level
3. The data level - manages the actual data that is being acted on

   Example: Internet search engine

Example : Internet Search Engine



Client-Server Model – Data Level

- Contains the programs that maintain the actual data on which the applications operate.
- Data are often persistent - even if no application is running, data will be stored somewhere for next use.
- Data level consists of a file system, but it is more common to use a full-fledged database.
- Data level is typically implemented at the server side.
- Responsible for keeping data consistent across different applications.
  - With databases - metadata such as table descriptions, entry constraints and application-specific metadata are also stored at this level.

➢ **Relational database** organize most business-oriented data.

➢ Data independence is crucial - data are organized independent of the applications in such a way that changes in that organization do not affect applications, and neither do the applications affect the data organization.

➢ Using relational databases in the client-server model helps separate the processing level from the data level, as processing and data are considered independent.

Q8. Write a note on multilayerd architec-tures.

Ans :

Multitiered Architectures

Possibilities for physically distributing a client-server application across several machines.

Simplest organization - two types of machines:

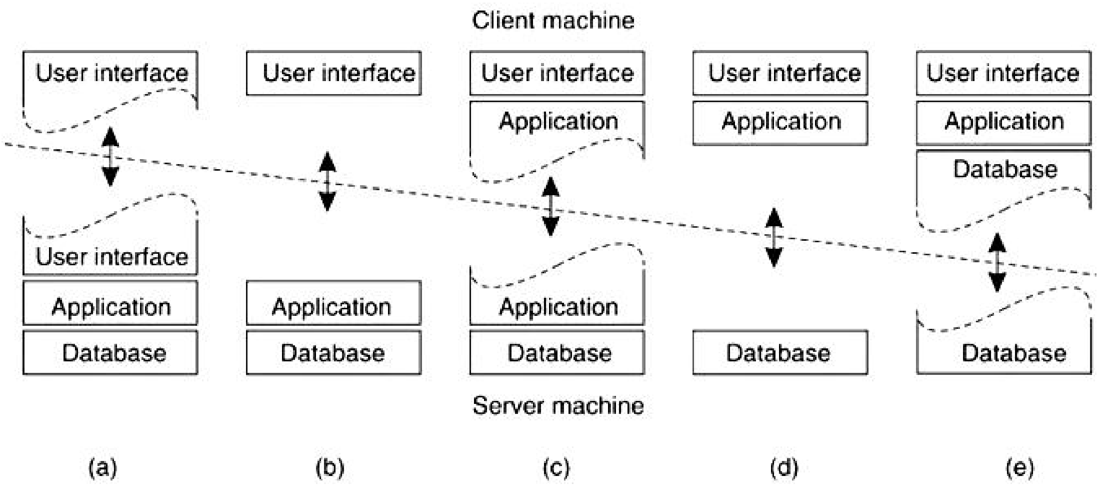
1. A client machine containing only the programs implementing (part of) the user-interface level.
2. A server machine containing the rest, that is the programs implementing the processing and data level.

➢ Everything is handled by the server while the client is essentially no more than a dumb terminal, possibly with a pretty graphical interface.

Distribute the programs in the application layers across different machines.

➢ Two-tiered architecture: client machines and server machines.

➢ Alternative client-server organizations (a)–(e).



Cases

A: only the terminal-dependent part of the user interface on the client machine

B: place the entire user-interface software on the client side

Divide the application into a graphical front end, which communicates with the rest of the application (residing at the server) through an application-specific protocol.

The front end (the client software) does no processing other than necessary for presenting the application's interface

Move part of the application to the front end

- e.g. the application makes use of a form that needs to be filled in entirely before it can be processed
- front end can then check the correctness and consistency of the form, and where necessary interact with the user

D: used where the client machine is a PC or workstation, connected through a network to a distributed file system or database

- Most of the application is running on the client machine, but all operations on files or database entries go to the server
- e.g. many banking applications run on an end-user's machine where the user prepares transactions and such

  Once finished, the application contacts the database on the bank's server and uploads the transactions for further processing

E: used where the client machine is a PC or workstation, connected through a network to a distributed file system or database

- The situation where the client's local disk contains part of the data

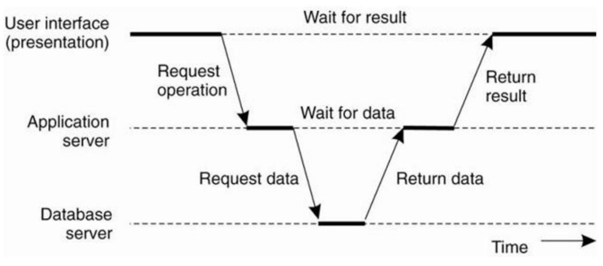
Issues

Trend to move away from the configurations D and E.

- Although client machines do a lot, they are also more problematic to manage
- Having more functionality on the client machine makes client-side software more prone to errors and more dependent on the client's underlying platform (i.e., operating system and resources).
- From a system's management perspective, having fat clients is not optimal.
- Thin clients in A – C are much easier

  Server-side solutions are becoming increasingly more distributed as a single server is being replaced by multiple servers running on different machines.A server may sometimes need to act as a client leading to a (physically) three-tiered architecture.

- Example of a server acting as client.



Programs that form part of the processing level reside on a separate server, but may additionally be partly distributed across the client and server machines.

E.g.: three-tiered architecture - organization of Web sites.

Q9. Write about decentralized architectures.

Ans :

Decentralized Architectures

Vertical Distribution Architecture

Vertical Distribution Architecture is Achieved by placing logically different components on different machines. This term is related to the concept of vertical fragmentation as used in distributed relational databases, where it means that tables are split column-wise, and subsequently distributed across multiple machines.

- Multitiered client-server architectures are a direct consequence of dividing applications into a user-interface, processing components, and a data level. The different tiers correspond directly with the logical organization of applications.
- Vertical Distribution - organizing a client-server application as a multitieredrchitecture.
- Can help manage distributed systems by logically and physically splitting functions across multiple machines, where each machine is tailored to a specific group of functions.

Horizontal Distribution Architecture

In Horrizontal Distribution Client or server may be physically split up into logically equivalent parts, but each part is operating on its own share of the complete data set, thus balancing the load.

e.g peer-to-peer systems.

- Processes that constitute a peer-to-peer system are all equal.
- Functions that need to be carried out are represented by every process that constitutes the distributed system.
- Much of the interaction between processes is symmetric:
- Each process will act as a client and a server at the same time (which is also referred to as acting as a servent).
- Peer-to-peer architectures - how to organize the processes in an overlay network in which the nodes are formed by the processes and the links represent the possible communication channels (which are usually realized as TCP connections).
- A Process cannot communicate directly with an arbitrary other process, but is required to send messages through the available communication channels.
- Two types of overlay networks exist: those that are structured and those that are not.

Structured Peer-to-Peer Architectures

- The P2P overlay network consists of all the participating peers as network nodes.
- There are links between any two nodes that know each other: i.e. if a participating peer knows the location of another peer in the P2P network, then there is a directed edge from the former node to the latter in the overlay network.
- Based on how the nodes in the overlay network are linked to each other, we can classify the P2P networks as unstructured or structured.
- Some well known structured P2P networks are Chord, Pastry, Tapestry, CAN, and Tulip.
- A structured Peer-to-Peer overlay network is constructed using a deterministic procedure.
- Most-used procedure - organize the processes through a distributed hash table (DHT).
- In DHT-based system –
- Data items are assigned a random key from a large identifier space, such as a 128-bit or 160-bit identifier.
- nodes are assigned a random number from the same identifier space.
- DHT-based system implements an efficient and deterministic scheme that uniquely maps the key of a data item to the identifier of a node based on some distance metric .
- When looking up a data item, the network address of the node responsible for that data item is returned.
- This is accomplished by routing a request for a data item to the responsible node.

Unstructured Peer-to-Peer Architectures.

An unstructured P2P network is formed when the overlay links are established arbitrarily.

- Such networks can be easily constructed as a new peer that wants to join the network can copy existing links of another node and then form its own links over time.
- In an unstructured P2P network, if a peer wants to find a desired piece of data in the network, the query has to be flooded through the network in order to find as many peers as possible that share the data.
- Main disadvantage - queries may not always be resolved.
- Popular content is likely to be available at several peers and any peer searching for it is likely to find the same thing, but, if a peer is looking for a rare or not-so-popular data shared by only a few other peers, then it is highly unlikely that search will be successful.
- Since there is no correlation between a peer and the content managed by it, there is no guarantee that flooding will find a peer that has the desired data.

- Flooding also causes a high amount of signaling traffic in the network and hence such networks typically have very poor search efficiency.
- Most of the popular P2P networks such as Napster, Gnutella and KaZaA are unstructured.
- Rely on randomized algorithms for constructing an overlay network.
- Each node maintains a list of neighbors constructed in a more or less random way.
- Data items are assumed to be randomly placed on nodes.

Q10. What are called as super peers?

Ans :

Superpeers

Network nodes that maintaining an index of node or acting as a broker for nodes are generally referred to as superpeers.

Unstructured peer-to-peer systems - locating relevant data items can become problematic as the network grows.

- No deterministic way of routing a lookup request to a specific data item -> only technique a node can resort to is flooding the request.
- Flooding can be dammed-> alternative -> use special nodes that maintain an index of data items.

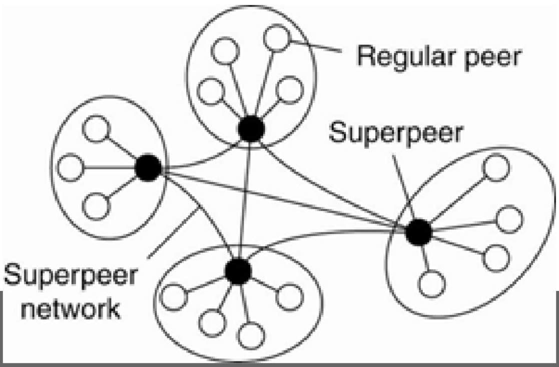
Other situations in which abandoning the symmetric nature of peer-to-peer systems is sensible.

Example:

Collaboration of nodes that offer resources to each other.

- In a collaborative content delivery network (CDN), nodes may offer storage for hosting copies of Web pages allowing Web clients to access pages nearby, and thus to access them quickly.
- A node P may need to seek for resources in a specific part of the network.
- Making use of a broker that collects resource usage for a number of nodes that are in each other's proximity will allow to quickly select a node with sufficient resources.

A hierarchical organization of nodes into a superpeer network.



- The client-superpeer relation is fixed n many cases: whenever a regular peer joins the network, it attaches to one of the superpeers and remains attached until it leaves the network.
- Expected that superpeers are long-lived processes with a high availability.
- To compensate for potential unstable behavior of a superpeer, backup schemes can be deployed, such as pairing every superpeer with another one and requiring clients to attach to both.

Q11. Write about various Hybrid Architectures

Ans :

Hybrid Architectures

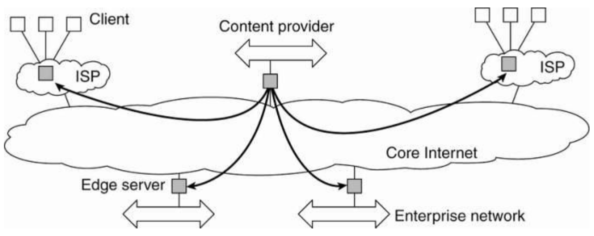
Edge-Server Systems

- Deployed on the Internet where servers are placed "at the edge" of the network.
- Purpose is to serve content, possibly after applying filtering and transcoding functions
- A collection of edge servers can be used to optimize content and application distribution

This edge is formed by the boundary between enterprise networks and the actual Internet e.g, an Internet Service Provider (ISP).

E.g. end users at home connect to the Internet through their ISP, the ISP can be considered as residing at the edge of the Internet.

Viewing the Internet as consisting of a collection of edge servers.



Basic model-one edge server acts as an origin server from which all content originates.

Collaborative Distributed Systems

Hybrid architectures are deployed in collaborative distributed systems.

Two step process:

1. Join system using a traditional client-server scheme.
2. Once a node has joined the system - use a fully decentralized scheme for collaboration.

Example

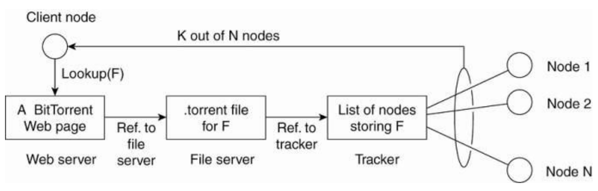
The BitTorrent file-sharing system (Cohen, 2003).

➢ BitTorrent is a peer-to-peer file downloading system.

➢ An end user downloads chunks of a file from other users until the downloaded chunks can be assembled together yielding the complete file.

➢ BitTorrent combines centralized with decentralized solutions.

➢ The principal working of BitTorrent



➢ Design goal - ensure collaboration.

1.2.3 Architectures Versus Middleware

Q12. Write about middleware sytems.

Ans :

➢ Middleware forms a layer between applications and distributed platforms

➢ Provide a degree of distribution transparency, hiding the distribution of data, processing, and control from applications.

➢ Where middleware fits in?

Middleware systems follow a specific architectural style

➢ Object-based architectural style - CORBA

➢ Event-based architectural style - TIB/ Rendezvous

Problems

➢ Molding middleware molded to a specific architectural style makes designing applications simpler BUT the middleware may no longer be optimal for what an application developer had in mind.

➢ Middleware is meant to provide distribution transparency, BUT specific solutions should be adaptable to application requirements.

Solutions :

Good

Make several versions of a middleware system, where each version is tailored to a specific class of applications.

Better

Make middleware systems that are easy to configure, adapt, and customize as needed by an application.

Q13. What are Interceptors?

Ans :

Interceptors

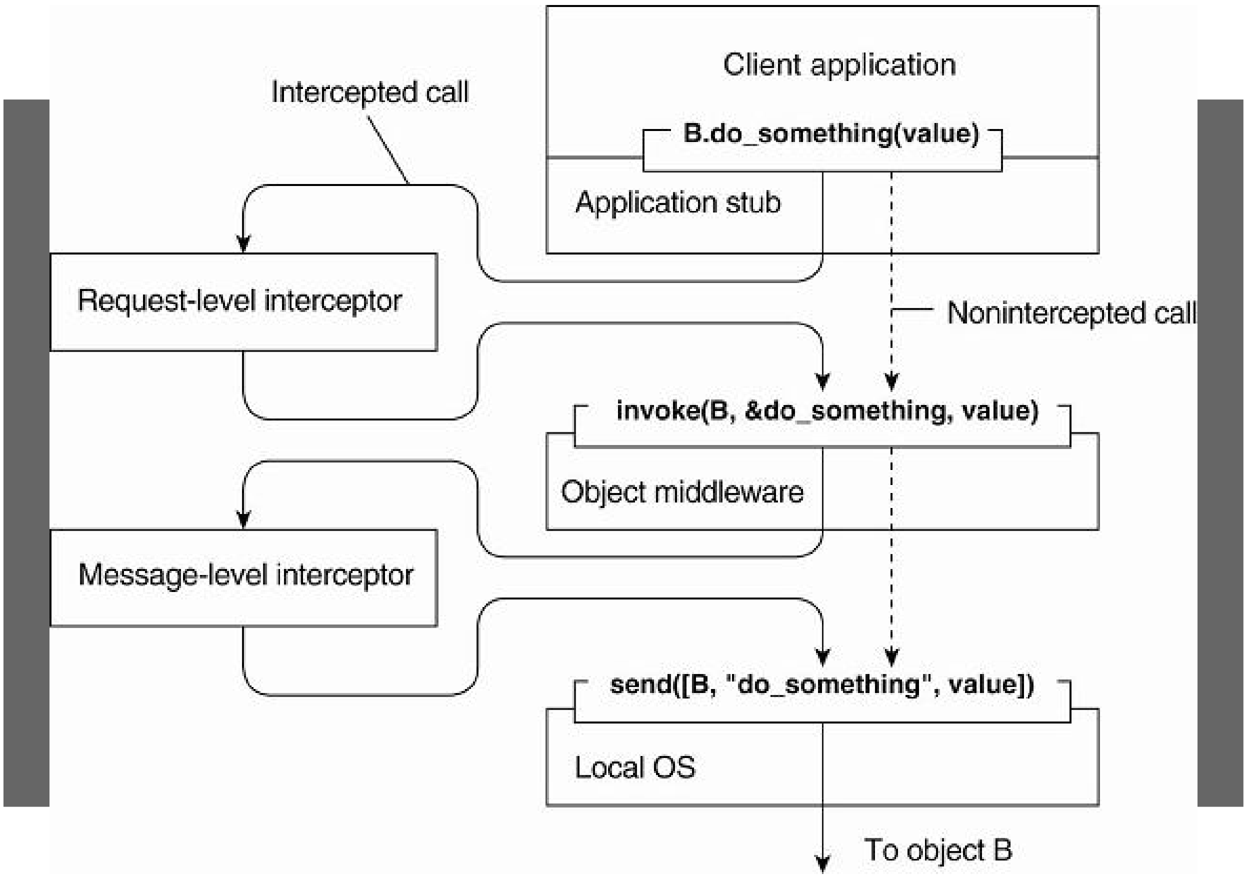
An interceptor is a software construct that will break the usual flow of control and allow other (application specific) code to be executed.

Example: consider interception as supported in many object-based distributed systems.

- An object A can call a method that belongs to an object B, while the latter resides on a different machine than A.
- This remote-object invocation is carried out in 3 steps:

1. Object A is offered a local interface that is exactly the same as the interface offered by object B. A simply calls the method available in that interface.
2. The call by A is transformed into a generic object invocation, made possible through a general object-invocation interface offered by the middleware at the machine where A resides.
3. Finally, the generic object invocation is transformed into a message that is sent through the transport-level network interface as offered by A's local operating system.

   Using interceptors to handle remote-object invocations.



1. After the first step, the call B.do_something(value) is transformed into a generic call such as invoke(B, &do_something, value) with a reference to B's method and the parameters that go along with the call.
2. Assume that object B is replicated.
   - Here, each replica should be invoked.
3. Interception helps here the request-level interceptor will call invoke(B, &do_something, value) for each of the replicas.
   - Object A need not be aware of the replication of B
   - The object middleware need not have special components that deal with this replicated call.

- Only the request-level interceptor, which may be added to the middleware needs to know about B's replication.
- A call to a remote object will have to be sent over the network.
- The messaging interface as offered by the local operating system will need to be invoked.
- At that level, a message-level interceptor may assist in transferring the invocation to the target object.

Q14. What is adaptive software ? what are the basic techniques used in adaptive software for middleware.

Ans :

Adaptive Software for Middleware

Adaptive software is specialized software designed for physically challenged users. This software usually runs on specialized hardware.

Three basic techniques to come to software adaptation and open research area .

1. **Separation of Concerns**
   - Separate the parts that implement functionality from those that take care of other things (known as extra functionalities) such as reliability, performance, security, etc
   - Cannot easily separate these extra functionalities by means of modularization
   - Aspect-oriented software development used to address separation of concerns
2. **Computational Reflection**

   The ability of a program to inspect itself and, if necessary, adapt its behavior.
3. **Component-based Design**
   - Supports adaptation through composition.
   - A system may either be configured statically at design time, or dynamically at runtime.

The latter requires support for late binding, a technique that has been successfully applied in programming language environments, but also for operating systems where modules can be loaded and unloaded at will.

1.2.4 Self-management In Distributed Systems

Q15. Write about the Feed back control model for self-management in distributed sytems.

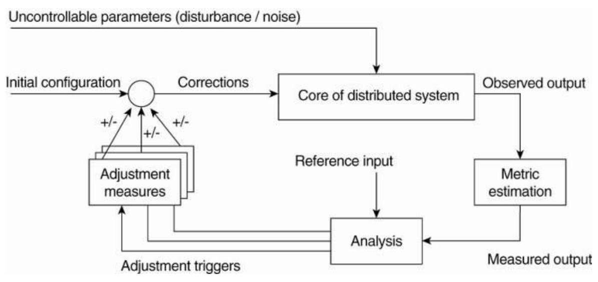
Ans :

Self management system will have the following characteristics:

- Must organize the components of a distributed system such that monitoring and adjustments can be done
- Organize distributed systems as high-level feedback-control systems to allow automatic adaptations to changes:
- Autonomic computing
- Self-star systems - indicates the variety by which automatic adaptations are being captured: self-managing, self-healing, self-configuring, self-optimizing, etc.

  The Feedback Control Model
- Adaptations take place by means of one or more feedback control loops.
- Systems that are organized by means of such loops are referred to as feedback control systems.
- Feedback control has since long been applied in various engineering fields, and its mathematical foundations are gradually also finding their way in computing systems
- For self-managing systems, the architectural issues are initially the most interesting.

  The basic idea behind this organization is :



Three elements that form the feedback control loop:

1. The system itself needs to be monitored, which requires that various aspects of the system need to be measured.
2. Another part of the feedback control loop analyzes the measurements and compares these to reference values. This feedback analysis component forms the heart of the control loop, as it will contain the algorithms that decide on possible adaptations.
3. The last group of components consist of various mechanisms to directly influence the behaviour of the system.
   - ➢ There can be many different mechanisms: placing replicas, changing scheduling priorities, switching services, moving data for reasons of availability, redirecting requests to different servers, etc.
   - ➢ The analysis component will need to be aware of these mechanisms and their (expected) effect on system behavior.

1.3 Processes

Q16. Write a short note on Processes.

Ans :

Processes

- ➢ An operating system creates a number of virtual processors, each one for running a different program.
- ➢ The operating system has a process table to keep track of these virtual processors.
- ➢ The process table contains entries to store CPU register values, memory maps, open files, accounting information, privileges, etc.
- ➢ A process is a running instance of a program, including all variables and other state attributes on one of the operating system's virtual processors.
- ➢ The operating system ensures that independent processes cannot affect each other's behavior.
- ➢ Sharing the same CPU and other hardware resources is made transparent with hardware support to enforce this separation.
- ➢ Each time a process is created, the operating system must create a complete independent address space.

  E.g.: zeroing a data segment, copying the associated program into a text segment, and setting up a stack for temporary data.
- ➢ Switching the CPU between two processes requires:
- ➢ Saving the CPU context (which consists of register values, program counter, stack pointer, etc.),
- ➢ Modifying registers of the memory management unit (MMU)
- ➢ Invalidate address translation caches such as in the translation lookaside buffer (TLB) a cache in a CPU that is used to improve the speed of virtual address translation.
- ➢ If the operating system supports more processes than it can simultaneously hold in main memory, it may have to swap processes between main memory and disk before the actual switch can take place.

1.3.1 Threads

Q17. Write a note on threads.

Ans :

- ➢ A traditional process has a single thread of control and a single program counter.
- ➢ Modern operating systems provide multiple threads of control within a process at the kernel level these are called *Threads* or lightweight processes. (A heavyweight process is an instance of a program).
- ➢ All threads within a process share the same address space.
- ➢ A thread shares its code and data section with other threads.
- ➢ Each thread has its own program counter, stack and register set.

- The *program counter* determines which instruction the thread is currently executing.
- The *register set* saves the thread's state when it is suspended and reloaded into the machine registers upon resumption.

Thread Types

User Threads

- Threads are implemented at the user level by a thread library
  - Library provides support for thread creation, scheduling and
  - Management.
  - User threads are fast to create and manage.

Kernel Threads

- Supported and managed directly by the OS.
  - Thread creation, scheduling and management take place in kernel space.
  - Slower to create and manage.

Q18. What are the uses of threads in distributed environment.

Ans :

Increased Responsiveness to User

A program continues running with other threads even if part of it is blocked or performing a lengthy operation in one thread.

Resource Sharing

Threads share memory and resources of their process.

Economy

Less time consuming to create and manage threads than processes as threads share resources,

e.g., thread creating is 30 times faster than process creating in Solaris.

Utilization of Multiprocessor Architectures

Increases concurrency because each thread can run in parallel on a different processor.

- Many applications are easier to structure as a collection of cooperating threads.
  - e.g., word processor - separate threads can be used for handling user input, spelling and grammar checking, document layout, index generation, etc.

Q19. Write, how to implement the threads in distributed environment.

Ans :

Thread Implementation

Threads are provided in the form of a thread package. The package contains operations to create and destroy threads as well as operations on synchronization variables such as mutexes and condition variables.

Two approaches to implement a thread package.

1. Construct a thread library that is executed entirely in user mode.

   Advantages

   - It is cheap to create and destroy threads
   - All thread administration is kept in the user's address space, the price of creating a thread is primarily determined by the cost for allocating memory to set up a thread stack
   - Destroying a thread mainly involves freeing memory for the stack, which is no longer used.
   - Switching thread context can be done in just a few instructions

   Disadvantage

   - A blocking system call will immediately block the entire process to which the thread belongs, and thus also all the other threads in that process

2. Have the kernel be aware of threads and schedule them.

   Advantages

   - Eliminates blocking problem.

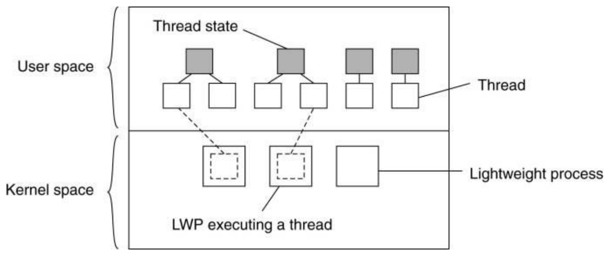
   Disadvantage

   - Every thread operation (creation, deletion, synchronization, etc.), will have to be carried out by the kernel, requiring a system call.

Solution :

Hybrid of user-level and kernel-level threads, generally referred to as lightweight processes (LWP).

- An LWP runs in the context of a single (heavyweight) process
- There can be several LWPs per process.
- User-level thread packages can be used as well.
- The package provides facilities for thread synchronization, such as mutexes and condition variable
  - Mutex - **mu**tual **ex**clusion object.
  - A mutex is a program object that allows multiple program threads to share the same resource, such as file access, but not simultaneously.When a program is started, a mutex is created with a unique name.
  - After this stage, any thread that needs the resource must lock the mutex from other threads while it is using the resource.
  - The mutex is set to unlock when the data is no longer needed or the routine is finished.
  - The thread package is implemented entirely in user space in which all operations on threads are carried out without intervention of the kernel.
  - The thread package can be shared by multiple LWPs
    - Each LWP can run its own (user-level) thread.
    - Multithreaded applications are constructed by creating threads, and subsequently assigning each thread to an LWP
    - Assigning a thread to an LWP is normally implicit and hidden from the programmer.
  - Combining kernel-level lightweight processes and user-level threads.



Multithreading Models

Q20. Write about various multithreading models.

Ans:

Three common ways of establishing a relationship between user level threads and kernel-level threads

1. Many-to-One

Many user-level threads mapped to single kernel thread.

- Easier thread management.
- Blocking-problem.
- No concurrency.
- **Examples:** Green threads for Solaris

2. One-to-One

Each user-level thread maps to a kernel thread.

- Overhead of creating kernel threads, one for each user thread.
- No blocking problem
- Provides concurrency.
- Examples: Linux, family of Windows

3. Many-to-Many

Allows many user level threads to be mapped to many kernel threads.

- Allows the OS to create a sufficient number of kernel threads.
- Users can create as many as user threads as necessary.
- No blocking and concurrency problems.
- Two-level model.

Q21. Write about some general concepts in threads.

Ans :

fork and exec :

these are two basic operations on therads

fork system call :

Change in semantics of **fork()** and **exec()** system calls.

Two versions of **fork** system call are the following :

- One duplicates only the thread that invokes the call.
- Another duplicates all the threads, i.e., duplicates an entire process.

exec system call: Program specified in the parameters to **exec** will replace the entire process – including all threads. If **exec** is called immediately after forking, duplicating all threads is not required.

Cancellation

Task of terminating a thread before it has completed.Cancelling one thread in a multithreaded searching through a database.It Stopsa web page from loading.

Two ways of termination or cancellation threads

- **Asynchronous cancellation:** One thread immediately terminates the target thread.
- **Deferred cancellation :** The target thread can periodically check if it should terminate, allowing a normal exit.

Signal Handling

Signal handling is the Signal to notify a process that a particular event has occurred. The signal can be Default or user defined signal handler.

There are two types of signals

- Synchronous signal is related to the operation performed by a running process.
- Illegal memory access or division by zero.
- Asynchronous signal is caused by an event external to a running process.
- Terminating a process (<control><C>) or a timer expires.

Options for delivering signals in a multithreaded process:

- Signal to the thread to which the signal applies.
- Signal to all threads.
- Signal to certain threads.
- Signal to a specific thread.

Thread Pools

Thread Pools are used to Create a number of threads at process startup and place them into a pool where they sit and wait for work.

- e.g. for multithreading a web server.

A thread from the pool is activated on the request, and it returns to the pool on completion.Thread-pool-architecture allows dynamic adjustment of pool size.

Benefits of thread pool

- Faster service
- Limitation on the number of threads, according to the need.

Scheduler Activations

Scheduler Activations are used for Communication between the user-thread library and the kernel threads.It uses an intermediate data structure - LWP.

- User thread runs on a virtual processor (LWP)
- Corresponding kernel thread runs on a physical processor

  Each application gets a set of virtual processors from OS
- Application schedules threads on these processors
- Kernel informs an application about certain events issuing upcalls which are handled by thread library.

Advantages of LWPs in combination with a user-level thread package:

1. Creating, destroying, and synchronizing threads is relatively cheap and involves no kernel intervention at all.

2. Provided that a process has enough LWPs, a blocking system call will not suspend the entire process.
3. There is no need for an application to know about the LWPs. All it sees are user-level threads.
4. LWPs can be easily used in multiprocessing environments, by executing different LWPs on different CPUs.

1.3.2 Virtualization

Q22. What is virtualization?

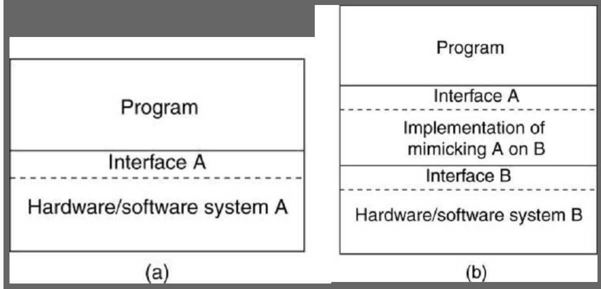
Ans :

Virtualization is a broad term that refers to the abstraction of computer resources.It creates an external interface that hides an underlying implementation

The following figures shows the virtualization:

(a) General organization between a program, interface, and system.

(b) General organization of virtualizing system A on top of system B.


Program
Interface A
Hardware/software system A
(a)
Program
Interface A
Implementation of mimicking A on B
Interface B
Hardware/software system B
(b)

Virtualization divided into two main categories:

1. **Platform virtualization :** It involves the simulation of virtual machines.
   - ➢ Platform virtualization is performed on a given hardware platform by "host" software (a control program), which creates a simulated computer environment (a virtual machine) for its "guest" software.
   - ➢ The "guest" software, which is often itself a complete operating system, runs just as if it were installed on a stand-alone hardware platform.
2. **Resource virtualization:** It involves the simulation of combined, fragmented, or simplified resources. Virtualization of specific system resources, such as storage volumes, name spaces, and network resources.

Q23. What is the role of virtualization to solve the problems in distributed systems.

Ans :

Role of Virtualization in Distributed Systems

Problem

While hardware and low-level systems software change reasonably fast, software at higher levels of abstraction (e.g., middleware and applications), are much more stable - legacy software cannot be maintained in the same pace as the platforms it relies on.

Solution

Virtualization can help here by porting the legacy interfaces to the new platforms and thus immediately opening up the latter for large classes of existing programs.

Problem

Networking has become completely pervasive. Connectivity requires that system administrators maintain a large and heterogeneous collection of server computers, each one running very different applications, which can be accessed by clients.

Solution

The diversity of platforms and machines can be reduced by letting each application run on its own virtual machine, possibly including the related libraries and operating system, which, in turn, run on a common platform.

Problem

Management of content delivery networks that support replication of dynamic content becomes easier if edge servers supported virtualization, allowing a complete site, including its environment to be dynamically copied.

Solution

Virtualization provides a high degree of portability and flexibility making it an important mechanism for distributed systems.

Q24. Where can we use virtual machines. Mention.

Ans :

Four distinct levels of interfaces to computers the behavior of which that virtualization can use:

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). In many cases, the aforementioned system calls are hidden by an API.

Virtualization can take place in two different ways:

1. Process virtual machine

- Build a runtime system that provides an abstract instruction set that is to be used for executing applications.
- Instructions can be interpreted (as is the case for the Java runtime environment), but could also be emulated as is done for running Windows applications on UNIX platforms.

2. Virtual machine monitor (VMM)

- Implemented as a layer completely shielding the original hardware, but offering the complete instruction set of that same (or other hardware) as an interface.
- This interface can be offered simultaneously to different programs.
- Possible to have multiple, and different operating systems run independently and concurrently on the same platform.
- **Example:** VMwareand Xen.

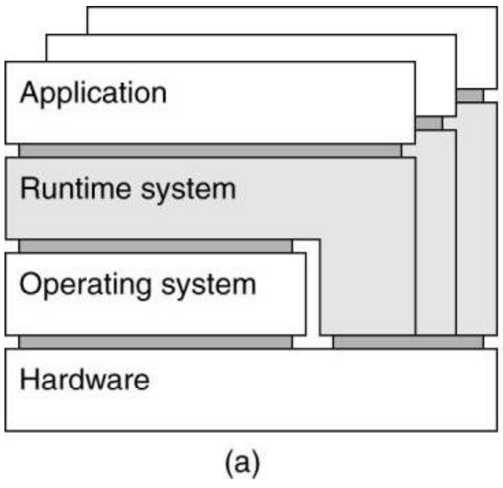


Fig:. (a) A process virtual machine, with multiple instances of (application, runtime) combinations.

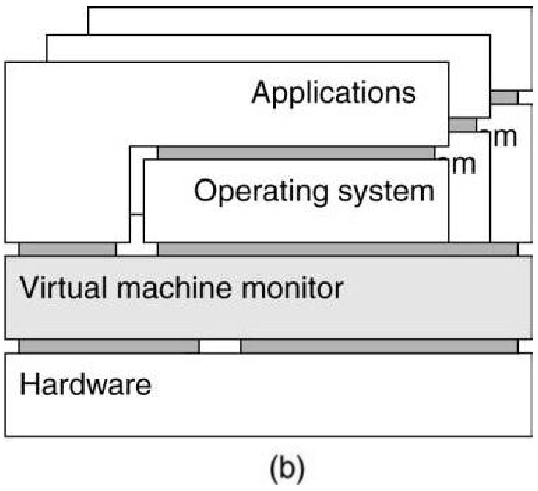


Fig.: (b) A virtual machine monitor, with multiple instances of (applications, operating system) combinations.

VMMs will become increasingly important in the context of reliability and security for (distributed) systems. Since they allow for the isolation of a complete application and its environment, a failure caused by an error or security attack need no longer affect a complete machine. Portability is greatly improved as VMMs provide a further decoupling between hardware and software, allowing a complete environment to be moved from one machine to another.

1.3.3 Clients

Q25. Write about networked user interfaces which support client–server interaction.

Ans :

Networked User Interfaces

Two ways to support client-server interaction:

1. For Each Remote Service - The client machine will have a separate counterpart that can contact the service over the network.

   Example: an agenda running on a user's PDA that needs to synchronize with a remote, possibly shared agenda.

   In this case, an application-level protocol will handle the synchronization, as shown in Figure(a).

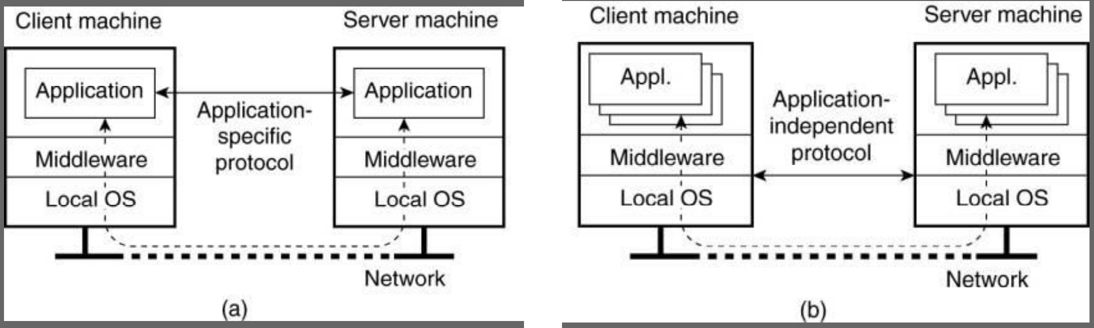
2. Provide direct access to remote services by only offering a convenient user interface.

   The client machine is used only as a terminal with no need for local storage, leading to an application neutral solution as shown in Figure(b).

   Thin-client approach - everything is processed and stored at the server

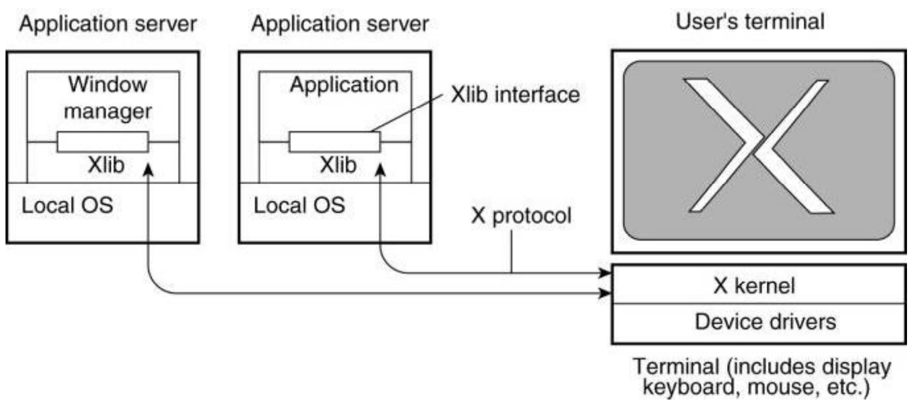
   (a) A networked application with its own protocol.

   (b) A general solution to allow access to remote applications.



Example: The X Window System (X)

- Used to control bit-mapped terminals, which include a monitor, keyboard, and a pointing device such as a mouse.
- Viewed as that part of an operating system that controls the terminal.
- X kernel is heart of the system.
- Contains all the terminal-specific device drivers - highly hardware dependent.
- X kernel offers a low-level interface for controlling the screen and for capturing events from the keyboard and mouse.
- This interface is made available to applications as a library called Xlib.
- The basic organization of the X Window System.



- X kernel and the X applications need not necessarily reside on the same machine.
- X provides the X protocol, which is an application-level communication protocol by which an instance of Xlib can exchange data and events with the X kernel.

Q26. What is Thin- Client Network Computing?

Ans :

Thin-Client Network Computing

- Applications manipulate a display using the specific display commands as offered by X.
- These commands are sent over the network where they are executed by the X kernel on the server.

Problem

- Applications written for X should separate application logic from user-interface commands
- This is often not the case - much of the application logic and user interaction are tightly coupled, meaning that an application will send many requests to the X kernel for which it will expect a response before being able to make a next step.
- Synchronous behavioradverselys affects performance when operating over a wide-area network with long latencies.

Solutions

1. Re-engineer the implementation of the X protocol, as is done with NX.

Q27. What are different Client-Side Software for Distribution Transparency?

Ans :

Distribution Transparency

A client should not be aware that it is communicating with remote processes.

Access Transparency

- Handled through the generation of a client stub from an interface definition of what the server has to offer.
- The stub provides the same interface as available at the server, but hides the possible differences in machine architectures, as well as the actual communication.

Location, Migration, and Relocation transparency

- A naming system is crucial
- Cooperation with client-side software is important

Example

When a client is already bound to a server, the client can be directly informed when the server changes location.

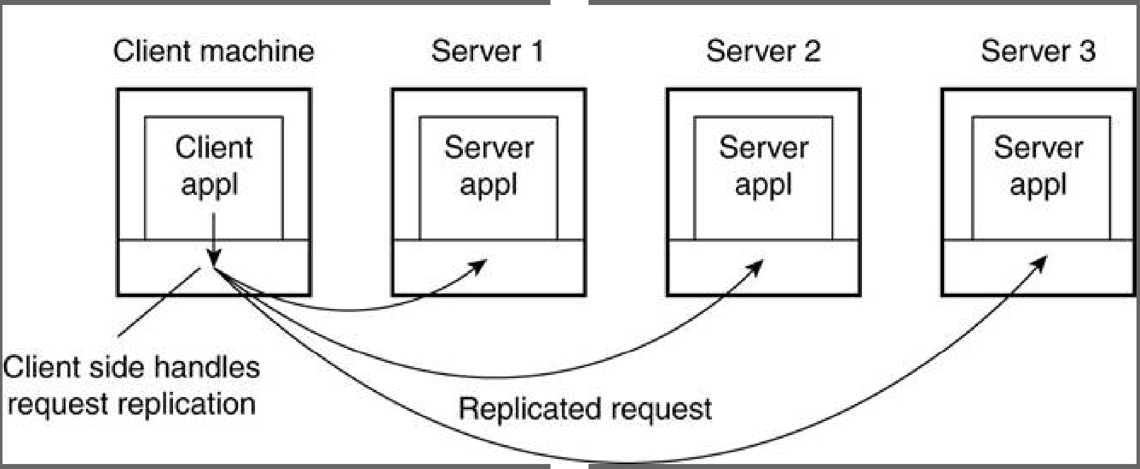
The client's middleware can hide the server's current geographical location from the user, and also transparently rebind to the server if necessary.

Replication Transparency

Implemented as client-side solutions.

Example: A distributed system with replica-ted servers.

- Replication can be achieved by forwarding a request to each replica (figure).
- Client-side software can transparently collect all responses and pass a single response to the client application.
- Transparent replication of a server using a client-side solution.



Failure Transparency

Masking communication failures with a server is typically done through client middleware.

Example: client middle-ware can be configured to repeatedly attempt to connect to a server. Client middleware can return data it had cached during a previous session, as is sometimes done by Web browsers that fail to connect to a server.

Concurrency Transparency

Handled through special intermediate servers, notably transaction monitors, and requires less support from client software.

Persistence transparency

Completely handled at the server.

1.3.4 Servers

Q28. What is server? Mention different types of servers.

Ans :

A server is a process implementing a specific service on behalf of a collection of clients.

Each server is organized in the same way:

- It waits for an incoming request from a client
- Ensures that the request is fulfilled
- It waits for the next incoming request.

Several ways to organize servers:

Iterative Server

Iterative server handles request, then returns results to the client; any new client requests must wait for previous request to complete (also useful to think of this type of server as sequential).

Concurrent server

Concurrent: server does not handle the request itself; a separate thread or sub-process handles the request and returns any results to the client; the server is then free to immediately service the next client.

- A multithreaded server is an example of a concurrent server.
- An alternative implementation of a concurrent server is to fork a new process for each new incoming request.
  - This approach is followed in many UNIX systems.
  - The thread or process that handles the request is responsible for returning a response to the requesting client.

Q29. Explain, how the clients contact a server.

Ans :

Clients send requests to an end point, also called a port, at the machine where the server is running. Each server listens to a specific end point.

How do clients know the end point of a service

Globally assign end points for well-known services.

Examples

1. Servers that handle Internet FTP requests always listen to TCP port 21.
2. An HTTP server for the World Wide Web will always listen to TCP port 80.

These end points have been assigned by the Internet Assigned Numbers Authority (IANA).With assigned end points, the client only needs to find the network address of the machine where the server is running.

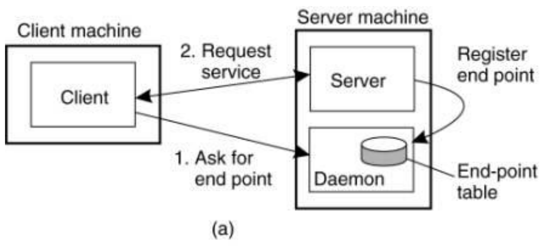
Many services that do not require a pre-assigned end point

Example

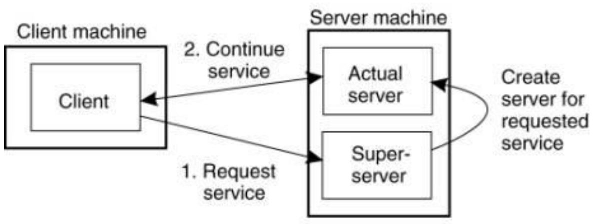
A time-of-day server may use an end point that is dynamically assigned to it by its local operating system. A client will look need to up the end point.

Solution

- A daemon running on each machine that runs servers.
- The daemon keeps track of the current end point of each service implemented by a co-located server.
- The daemon itself listens to a well-known end point.
- A client will first contant the daemon, request the end point, and then contact the specific server (Figure (a))



(a) Client-to-server binding using a daemon.



(b)

(b) Client-to-server binding using a superserver.

It is common to associate an end point with a specific service. Implementing each service by means of a separate server may be a waste of resources.

Example: UNIX system

- Many servers run simultaneously, with most of them passively waiting for a client request.
- Instead of having to keep track of so many passive processes, it is often more efficient to have a single superserver listening to each end point associated with a specific service, (Figure (b)).
- This is the approach taken, with the inetd daemon in UNIX.
  - Inetd manages Internet services by listening to a number of well-known ports for these services.
  - When a request comes in, the daemon forks a process to take further care of the request.
  - That process will exit after it is finished.

Design issue

Q30. Write about stateless and stateful servers.

Ans :

A **stateless server** is a server that treats each request as an independent transaction that is unrelated to any previous request. A stateless server does not keep information on the state of its clients, and can change its own state without having to inform any client.

Example: A Web server is stateless.

- It contains the following characterisitcs:
- It merely responds to incoming HTTP requests, which can be either for uploading a file to the server or (most often) for fetching a file.
- When the request has been processed, the Web server forgets the client completely.
- The collection of files that a Web server manages (possibly in cooperation with a file server), can be changed without clients having to be informed.
- Form of a stateless design - soft state.
  - The server promises to maintain state on behalf of the client, but only for a limited time.
  - After that time has expired, the server falls back to default behavior, thereby discarding any information it kept on account of the associated client.
  - Soft-state approaches originate from protocol design in computer networks, but can be applied to server design.

Stateful Server

A stateful server remembers client data (state) from one request to the next.Information needs to be explicitly deleted by the server.

Example

A file server that allows a client to keep a local copy of a file, even for performing update operations.

- The server maintains a table containing (client, file) entries.
- This table allows the server to keep track of which client currently has the update permissions on which file and the most recent version of that file.
- Improves performance of read and write operations as perceived by the client.

Advantages / Disadvantages

- Using a stateless file server, the client must specify complete file names in each request specify location for reading or writing re-authenticate for each request.

- Using a stateful file server, the client can send less data with each request
- A stateful server is simpler
- A stateless server is more robust: lost connections can't leave a file in an invalid state rebooting the server does not lose state information rebooting the client does not confuse a stateless server.

Q31. Differentiate between (temporary) session state and permanent state.

Ans :

- Session state is maintained in three-tiered client-server architectures, where the application server needs to access a database server through a series of queries before being able to respond to the requesting client.
- No real harm is done if session state is lost, provided that the client can simply reissue the original request.
- Permanent state information is maintained in databases, such as customer information, keys associated with purchased software, etc.
- For most distributed systems, maintaining session state already implies a stateful design requiring special measures when failures do happen and making explicit assumptions about the durability of state stored at the server.

Q32. What is server cluster? Write about it.

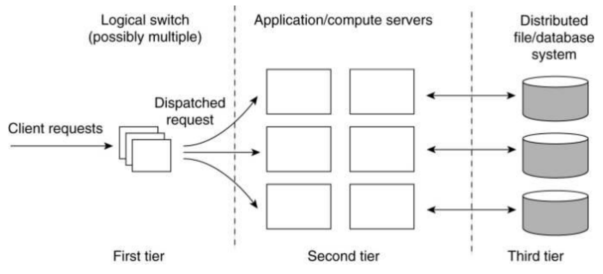
Ans :

Server Clusters

General Organization

A server cluster is a collection of machines connected through a network, where each machine runs one or more servers.

A server cluster is logically organized into three tiers


Logical switch (possibly multiple)
Application/compute servers
Distributed file/database system
Dispatched request
Client requests
First tier
Second tier
Third tier

First tier - consists of a (logical) switch through which client requests are routed.

Switches Vary

- Transport-layer switches accept incoming TCP connection requests and pass requests on to one of servers in the cluster,
- A Web server that accepts incoming HTTP requests, but that partly passes requests to application servers for further processing only to later collect results and return an HTTP response.

Second tier: Application Processing

Cluster computing servers run on high-performance hardware dedicated to delivering compute power.

Enterprise server clusters - applications may need to run on relatively low-end machines, as the required compute power is not the bottleneck, but access to storage is.

Third tier

Data-processing servers - notably file and database servers.

- These servers may be running on specialized machines, configured for high-speed disk access and having large server-side data caches.

1.3.5 Code Migration

Approaches to Code Migration

Q33. What are the reasons for code migration.

Ans :

Reasons for Migrating Code

Traditional method - process migration in which an entire process is moved from one machine to another.

Reasons

- Overall system performance can be improved if processes are moved from heavily-loaded to lightly-loaded machines.
- Load is expressed in terms of the CPU queue length or CPU utilization.

- Many modern distributed systems - optimizing computing capacity is less an issue than minimizing communication.
- Platform and network heterogeneity make decisions for performance improvement through code migration based on qualitative reasoning instead of mathematical models.

Examples

Client-server system where server manages a huge database

- If a client application needs to perform many database operations involving large quantities of data, it may be better to ship part of the client application to the server and send only the results across the network.
- Otherwise, the network may be swamped with the transfer of data from the server to the client. In this case, code migration is based on the assumption that it generally makes sense to process data close to where those data reside.

Migrating Parts of the Server to the Client

- Many interactive database applications clients need to fill in forms that are subsequently translated into a series of database operations.
- Processing the form at the client side, and sending only the completed form to the server, can avoid that a relatively large number of small messages need to cross the network.
- Result - the client perceives better performance, while at the same time the server spends less time on form processing and communication.

Reasons

flexibility - It is possible to dynamically configure distributed systems.

- Example - Client / Server application

Advantages

1. Clients need not have all the software preinstalled to talk to servers.

   The software can be moved as required, and discarded when no longer needed.

2. With standardized interfaces the client-server protocol and its implementation can be changed at will.

Changes will not affect existing client applications that rely on the server.

Disadvantages

Security

Blindly trusting that the downloaded code implements only the advertised interface while accessing an unprotected hard disk.

Q34. Write about the code migration.

Ans :

Models for Code Migration

A process consists of three segments:

1. Code segment - contains the set of instructions that make up the program that is being executed.
2. Resource segment - contains references to external resources needed by the process (e.g. files, printers, devices, other processes, etc.)
3. Execution segment - stores the current execution state of a process, consisting of private data, the stack, and the program counter.

Types of Process Mobility

Weak mobility - Transfer only the code segment, along with some initialization data.

- **Feature:** a transferred program is always started from one of several predefined starting positions.
- e.g., Java applets start execution from the beginning.
- Benefit: simplicity.
- Weak mobility requires only that the target machine can execute that code

  Strong mobility - execution segment is transferred as well.
- **Feature:** A running process can be stopped, moved to another machine, and resume execution where it left off.

- More general than weak mobility, but more difficult to implement.
- Can also be supported by remote cloning
- Cloning yields an exact copy of the original process, but running on a different machine.
- The cloned process is executed in parallel to the original process.
- In UNIX systems, remote cloning takes place by forking off a child process and letting that child continue on a remote machine.
- The benefit of cloning is that the model closely resembles the one that is already used in many applications.
- Migration by cloning is a simple way to improve distribution transparency.

Migration Initiation

Sender-initiated- migration is initiated at the machine where the code currently resides or is being executed.

- Done when uploading programs to a compute server.
- Securely uploading code to a server often requires that the client has previously been registered and authenticated at that server.

  Receiver-initiated-Initiative for code migration is taken by the target machine.
- Java applets are an example.
- Receiver-initiated migration is simpler than sender-initiated migration.
- Code migration usually occurs between a client and a server, where the client takes the initiative for migration.
- Downloading code can often be done anonymously.
- The server is not interested in the client's resources.
- Code migration to the client is done only for improving clientside performance.
- Only a limited number of resources need to be protected, such as memory and network connections.

Q35. Write about process to resource binding types.

Ans :

Types of process-to-resource bindings:

1. Binding by identifier - strongest binding - when a process refers to a resource by its identifier.
   - The process requires precisely the referenced resource, and nothing else.
   - Example : a process uses a URL to refer to a specific Web site or when it refers to an FTP server by means of that server's Internet address.
2. Binding by value - weaker binding - when only the value of a resource is needed.
   - Process execution is not affected if another resource provided that same value.
   - Example: a program relies on standard libraries, such as those for programming in C or Java.
   - Such libraries should always be locally available, but their exact location in the local file system may differ between sites.
   - Not the specific files, but their content is important for the proper execution of the process.
3. Binding by type – weakest binding - when a process indicates it needs only a resource of a specific type.
   - Exemplified by references to local devices, such as monitors, printers, etc.

Code migration requires consideration of the resource-to-machine bindings

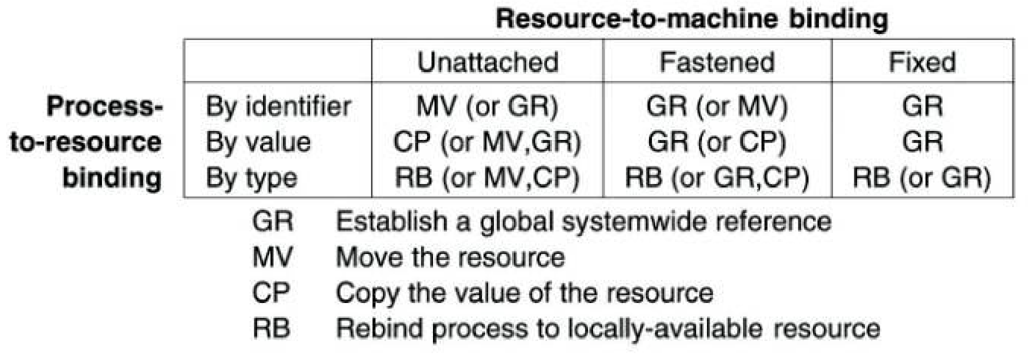
1. Unattached resources can be easily moved between different machines
   - Typically (data) files associated only with the program that is to be migrated.
2. Moving or copying a fastened resource may be possible, but only at relatively high costs.
   - Typical examples of fastened resources are local databases and complete Web sites.

3. Fixed resources are intimately bound to a specific machine or environment and cannot be moved.

   ➢ Fixed resources are often local devices or a local communication end point.

   Combining three types of process-to-resource bindings, and three types of resource-to-machine bindings, leads to nine combinations that we need to consider when migrating code.

   Actions to be taken with respect to the references to local resources when migrating code to another machine.



		Resource-to-machine binding		
		Unattached	Fastened	Fixed
Process-to-resource binding	By identifier	MV (or GR)	GR (or MV)	GR
	By value	CP (or MV,GR)	GR (or CP)	GR
	By type	RB (or MV,CP)	RB (or GR,CP)	RB (or GR)

GR	Establish a global systemwide reference
MV	Move the resource
CP	Copy the value of the resource
RB	Rebind process to locally-available resource

Examples

A process is bound to a resource by identifier.

i. When the resource is unattached, it is best to move it along with the migrating code.

ii. When the resource is shared by other processes establish a global reference

   1. A reference that can cross machine boundaries.

   2. e.g. a URL.

iii. When the resource is fastened or fixed, the best solution is to create a global reference.

Problems with global reference

Reference may be expensive to construct

e.g. a program that generates high-quality images for a dedicated multimedia workstation.

➢ Fabricating high-quality images in real time is a compute-intensive task, for which reason the program may be moved to a high-performance compute server.

➢ Establishing a global reference to the multimedia workstation means setting up a communication path between the compute server and the workstation.

➢ Significant processing involved at both the server and the workstation to meet the bandwidth requirements of transferring the images.

➢ Nt result - moving the program to the compute server is not such a good idea, only because the cost of the global reference is too high.

Bindings by value

➢ A fixed resource.

   ➢ The combination of a fixed resource and binding by value occurs when a process assumes that memory can be shared between processes.

  - ➢ Establishing a global reference would require a distributed form of shared memory.
  - ➢ In many cases, this is not really a viable or efficient solution.

- ➢ Fastened resources.
  - ➢ Fastened resources that are referred to by their value, are typically runtime libraries.
  - ➢ Normally, copies of such resources are readily available on the target machine, or should otherwise be copied before code migration takes place.
  - ➢ Establishing a global reference is a better alternative when huge amounts of data are to be copied, as may be the case with dictionaries and thesauruses in text processing systems.
- ➢ Unattached resources.
  - ➢ The best solution is to copy (or move) the resource to the new destination, unless it is shared by a number of processes.
  - ➢ In the latter case, establishing a global reference is the only option.

Bindings by type

- ➢ Irrespective of the resource-to-machine binding, the solution is to rebind the process to a locally available resource of the same type.
- ➢ If the resource is not available, must copy or move the original one to the new destination, or establish a global reference.

Q36. Explain the migration process in heterogeneous systems.

Ans :

Migration in Heterogeneous Systems

- ➢ Distributed systems are constructed on a heterogeneous collection of platforms, each having their own operating system and machine architecture.
- ➢ Migration requires that each platform be supported - the code segment can be executed on each platform.
- ➢ Must ensure that the execution segment can be properly represented at each platform.

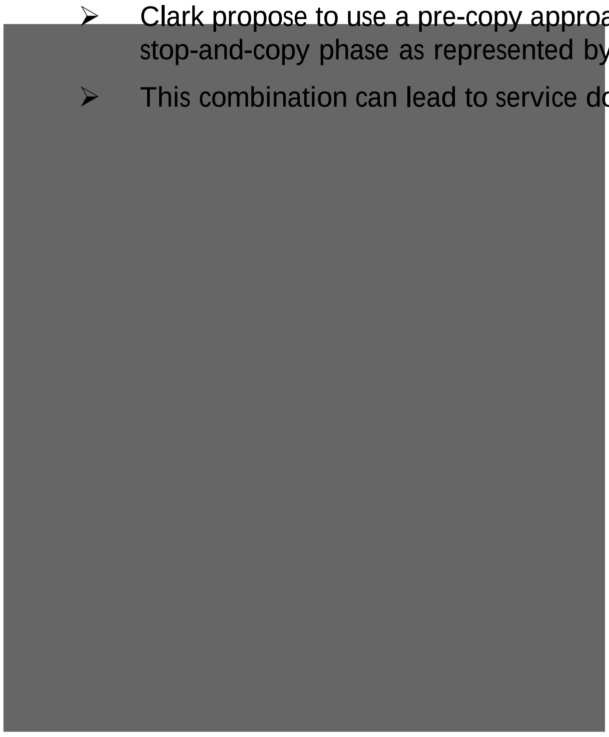
Migrate Entire Computing Environments

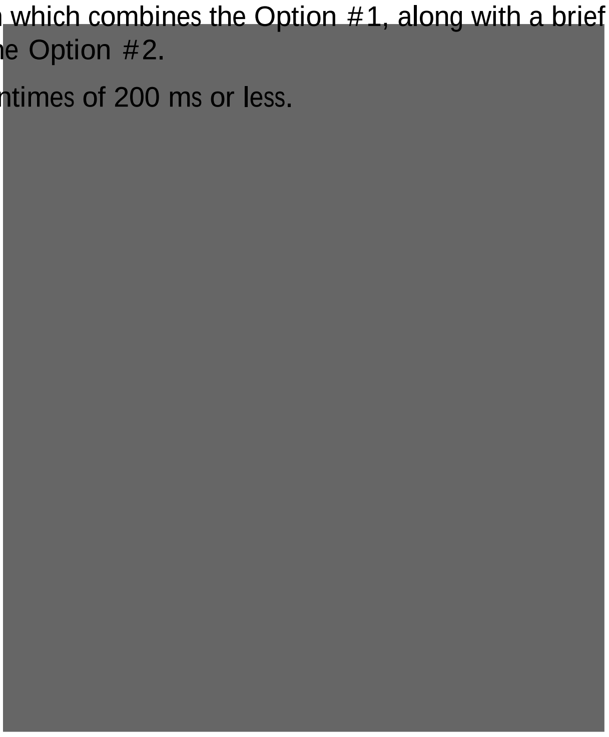
- ➢ Compartmentalize the overall environment to provide processes in the same part their own view on their computing environment.
- ➢ Can decouple a part from the underlying system and migrate it to another machine.
- ➢ Migration would provide a form of strong mobility for processes, as they can then be moved at any point during their execution, and continue where they left off when migration completes.
- ➢ Solves bindings to local resources problem - local resources become part of the environment that is being migrated.

Real-time migration of a virtualized operating system:

Three ways to handle migration (which can be combined):

1. Pushing memory pages to the new machine and resending the ones that are later modified during the migration process.
2. Stopping the current virtual machine; migrate memory, and start the new virtual machine.
3. Letting the new virtual machine pull in new pages as needed, that is, let processes start on the new virtual machine immediately and copy memory pages on demand.
   - ➢ Option #2 may lead to unacceptable downtime if the migrating virtual machine is running a live service.
   - ➢ Option #3 extensively prolong the migration period lead to poor performance because it takes a long time before the working set of the migrated processes has been moved to the new machine.
   - ➢ Clark propose to use a pre-copy approach which combines the Option #1, along with a brief stop-and-copy phase as represented by the Option #2.
   - ➢ This combination can lead to service downtimes of 200 ms or less.





UNIT II

Communication: Remote Procedure Call, Message-Oriented Communication, Stream-Oriented Communication, Multicast Communication. Naming:names, identifiers, and addresses, flat naming, structured naming, attribute based naming. Synchronization:clock synchronization, logical clocks, mutual exclusion, global positioning of nodes, election algorithms.

2.1 Communication

Q1. Write about the communication in distributed systems.

Ans :

Interprocess communication is fundamental communication to all distributed systems. Communication in distributed systems is always based on low-level message passing as offered by the underlying network.

Expressing communication through message passing is harder than using primitives based on shared memory. Modern distributed systems often consist of thousands or even millions of processes scattered across a network with unreliable communication. Unless the primitive communication facilities of computer networks are replaced by something else, development of large-scale distributed applications is extremely difficult.

Layered protocols are used for communication in distributed system. Due to the absence of shared memory, all communication in distributed systems is based on sending and receiving (low level) messages.

When process A wants to communicate with process B, it first builds a message in its own address space. Then it executes a system call that causes the operating system to send the message over the network to B.

Q2. Explain ISO OSI reference Model.

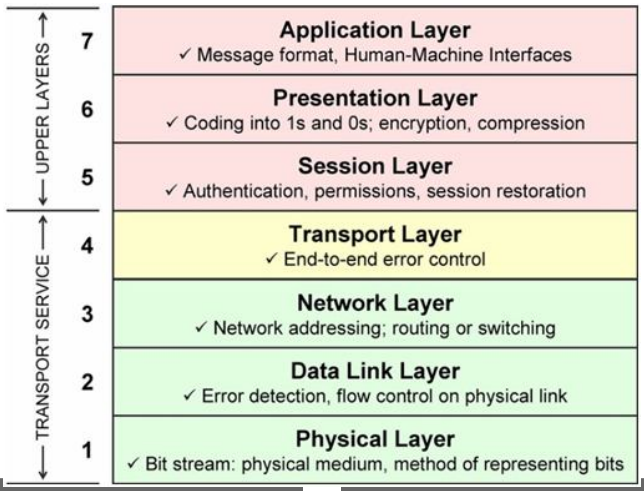
Ans :

ISO OSI Reference Model

Distributed systems use the general reference model which is known as ISO OSI reference model.

- Protocols that were developed as part of the OSI model were never widely used.
- Underlying model useful for understanding computer networks.
- OSI model is designed to allow open systems to communicate.
- An open system is one that is prepared to communicate with any other open system by using standard rules that govern the format, contents, and meaning of the messages sent and received.
- Rules are formalized into protocols.
- Groups of computers communicate over a network by agreeing on the protocols to be used.

In OSI model Communication is divided up into seven levels or layers. Each layer deals with one specific aspect of the communication. Each layer provides an interface to the one above it. The interface consists of a set of operations that together define the service the layer is prepared to offer its users.


UPPER LAYERS
TRANSPORT SERVICE
7
Application Layer
✓ Message format, Human-Machine Interfaces
6
Presentation Layer
✓ Coding into 1s and 0s; encryption, compression
5
Session Layer
✓ Authentication, permissions, session restoration
4
Transport Layer
✓ End-to-end error control
3
Network Layer
✓ Network addressing; routing or switching
2
Data Link Layer
✓ Error detection, flow control on physical link
1
Physical Layer
✓ Bit stream: physical medium, method of representing bits

Layer 7: Application Layer

- ➢ Application Layer provides means for the user to access information on the network through an application.
- ➢ This layer is the main interface for the user(s) to interact with the application and therefore the network.
- ➢ Some examples of application layer protocols include Telnet, applications which use File Transfer Protocol (FTP), applications which use Simple Mail Transfer Protocol (SMTP) and applications which use Hypertext Transfer Protocol (HTTP).

Layer 6: Presentation Layer

- ➢ Transforms data to provide a standard interface for the Application layer.
- ➢ MIME encoding, data compression, data encryption and similar manipulation of the presentation is done at this layer to present the data as a service or protocol developer sees fit.
- ➢ Examples: converting an EBCDIC-coded text file to an ASCII-coded file, or serializing objects and other data structures into and out of, e.g., XML.

Layer 5: Session Layer

- ➢ Controls the dialogues (sessions) between computers.
- ➢ It establishes, manages, and terminates the connections between the local and remote application.
- ➢ It provides for either duplex or half-duplex operation and establishes checkpointing, adjournment, termination, and restart procedures.
- ➢ The OSI model made this layer responsible for “graceful close” of sessions, which is a property of TCP, and also for session checkpointing and recovery, which is not usually used in the Internet protocols suite.

Layer 4: Transport Layer

- ➢ Provides transparent transfer of data between end users, thus relieving the upper layers from any concern while providing reliable data transfer.
- ➢ Controls the reliability of a given link through flow control, segmentation/desegmentation, and error control.

- Some protocols are state and connection oriented. This means that the transport layer can keep track of the packets and retransmit those that fail.
- Best known example of a layer 4 protocol is the Transmission Control Protocol (TCP).
- The transport layer is the layer that converts messages into TCP segments or User Datagram Protocol (UDP), Stream Control Transmission Protocol (SCTP), etc. packets.

Layer 3: Network Layer

- Provides the functional and procedural means of transferring variable length data sequences from a source to a destination via one or more networks while maintaining the quality of service requested by the Transport layer.
- The Network layer performs network routing functions.
- Routers operate at this layer - sending data throughout the extended network and making the Internet possible (also existing at layer 3 (or IP) are switches).
  - This is a logical addressing scheme – values are chosen by the network engineer.
  - The addressing scheme is hierarchical.
- The best known example of a layer 3 protocol is the Internet Protocol (IP).

Layer 2: Data Link Layer

- Provides the functional and procedural means to transfer data between network entities and to detect and possibly correct errors that may occur in the Physical layer.
- Best known example of this is Ethernet.
- On IEEE 802 local area networks, and some non-IEEE 802 networks such as FDDI, this layer may be split into a Media Access Control (MAC) layer and the IEEE 802.2 Logical Link Control (LLC) layer. It arranges bits from physical layer into logical chunks of data, known as frames.
- This is the layer at which the bridges and switches operate.
- Connectivity is provided only among locally attached network nodes forming layer 2 domains for unicast or broadcast forwarding.
- Other protocols may be imposed on the data frames to create tunnels and logically separated layer 2 forwarding domains.

Layer 1: Physical Layer

- Defines all the electrical and physical specifications for devices.
- This includes the layout of pins, voltages, and cable specifications. Hubs, repeaters, network adapters and Host Bus Adapters are physical-layer devices.
- The major functions and services performed by the physical layer are:
- Establishment and termination of a connection to a communications medium.
- Participation in the process whereby the communication resources are effectively shared among multiple users.
- Modulation or conversion between the representation of digital data in user equipment and the corresponding signals transmitted over a communications channel.
- Parallel SCSI buses operate in this layer.
- Various physical-layer Ethernet standards are also in this layer;
- Ethernet incorporates both this layer and the data-link layer.
- The same applies to other local-area networks, such as Token ring, FDDI, and IEEE 802.11, as well as personal area networks such as Bluetooth and IEEE 802.15.4.

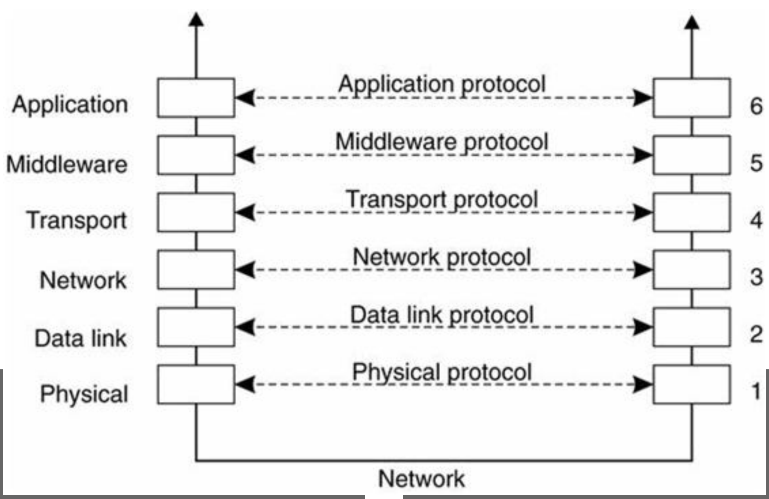
Q3. Write about middleware protocols and types of communication supports by middleware protocols.

Ans :

Middleware Protocols

Middleware is an application that logically lives (mostly) in the application layer, but contains many general-purpose protocols that warrant their own layers, independent of other, more specific applications.

Middleware communication protocols support high-level communication services. It adapted reference model for communication. Compared to the OSI model, the session and presentation layer have been replaced by a single middleware layer that contains application-independent protocols.



Viewing middleware as an intermediate (distributed) service in application-level communication.

Example: An electronic mail system.

1. The core of the mail delivery system viewed as a middleware communication service.
2. Each host runs a user agent allowing users to compose, send, and receive e-mail.
3. A sending user agent passes such mail to the mail delivery system, expecting it, to deliver the mail to the intended recipient.
4. The user agent at the receiver's side connects to the mail delivery system to see whether any mail has arrived.
5. If so, the messages are transferred to the user agent so that they can be displayed and read by the user.

Types of Communication

View middleware as an additional service in client-server computing,

Persistent Communication

An electronic mail system is a typical example of communication persistent communication.

1. With persistent communication, a message that has been submitted for transmission is stored by the communication middleware as long as it takes to deliver it to the receiver.
2. The middleware will store the message at one or several of the storage facilities (above figure).
   i) Not necessary for the sending application to continue execution after submitting the message.
   ii) The receiving application need not be executing when the message is submitted.

Transient Communication

A message is stored by the communication system only as long as the sending and receiving application are executing.

1. The middleware cannot deliver a message if there is a transmission interrupt or the recipient is currently not active it will discard the message.
2. All transport-level communication services offer only transient communication.

i) The communication system consists of traditional store-and-forward routers.

ii) If a router cannot deliver a message to the next one or the destination host, it will drop the message.

Asynchronous Communication

A sender continues executing immediately after it has submitted its message for transmission. This means that the message is (temporarily) stored by the middleware immediately upon submission.

Synchronous Communication

A sender is blocked until its request is known to be accepted.

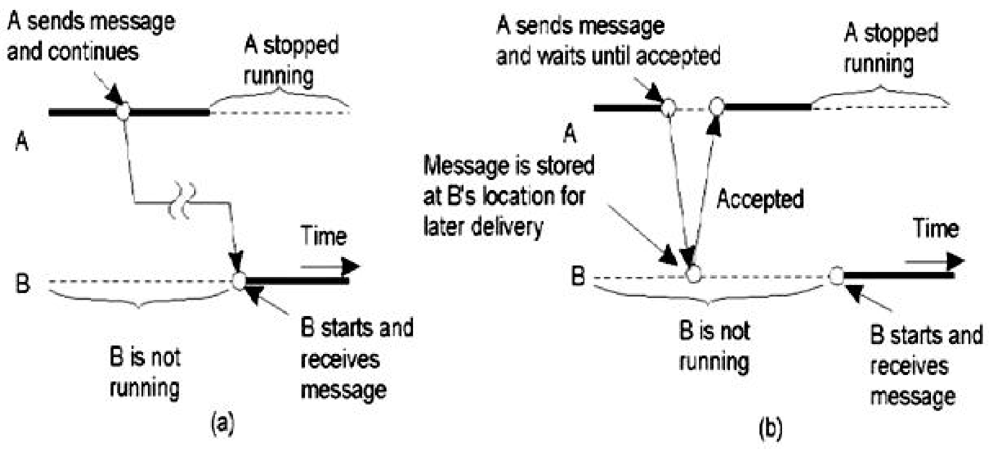
Three points where synchronization can take place:

1. The sender may be blocked until the middleware notifies that it will take over transmission of the request.
2. The sender may synchronize until its request has been delivered to the intended recipient.
3. Synchronization may take place by letting the sender wait until its request has been fully processed, that is, up the time that the recipient returns a response.

The following figures shows the distributed Communications Classifications for:

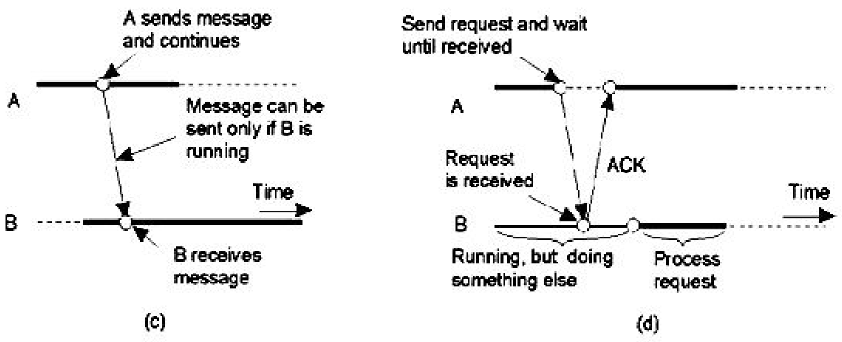
a) Persistent asynchronous communication.

b) Persistent synchronous communication.


A sends message and continues
A stopped running
A
Time
B
B is not running
B starts and receives message
(a)
A sends message and waits until accepted
A stopped running
A
Message is stored at B's location for later delivery
Accepted
Time
B
B is not running
B starts and receives message
(b)

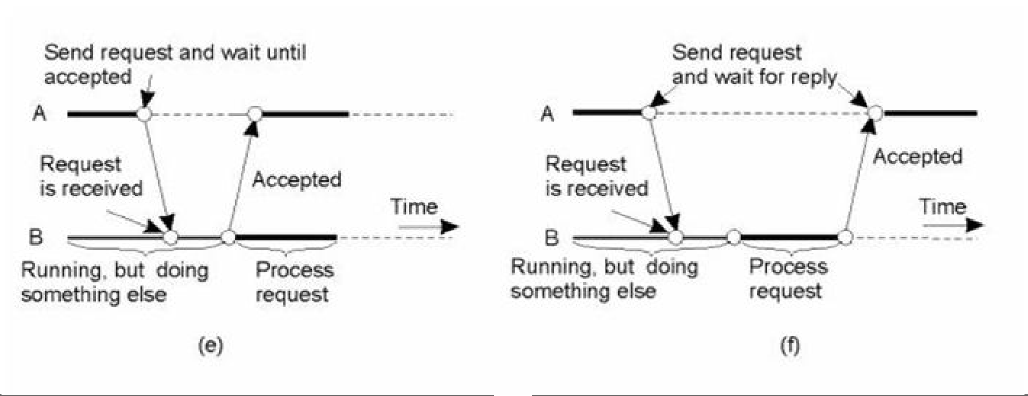
c) Transient asynchronous communication.

d) Receipt-based transient synchronous communication.


A sends message and continues
A
Message can be sent only if B is running
Time
B
B receives message
(c)
Send request and wait until received
A
Request is received
ACK
Time
B
Running, but doing something else
Process request
(d)

e) Delivery-based transient synchronous communication at message delivery.

f) Response-based transient synchronous communication.


Send request and wait until accepted
A
Request is received
Accepted
Time
B
Running, but doing something else
Process request
(e)
Send request and wait for reply
A
Request is received
Accepted
Time
B
Running, but doing something else
Process request
(f)

2.1.1 Remote Procedure Call

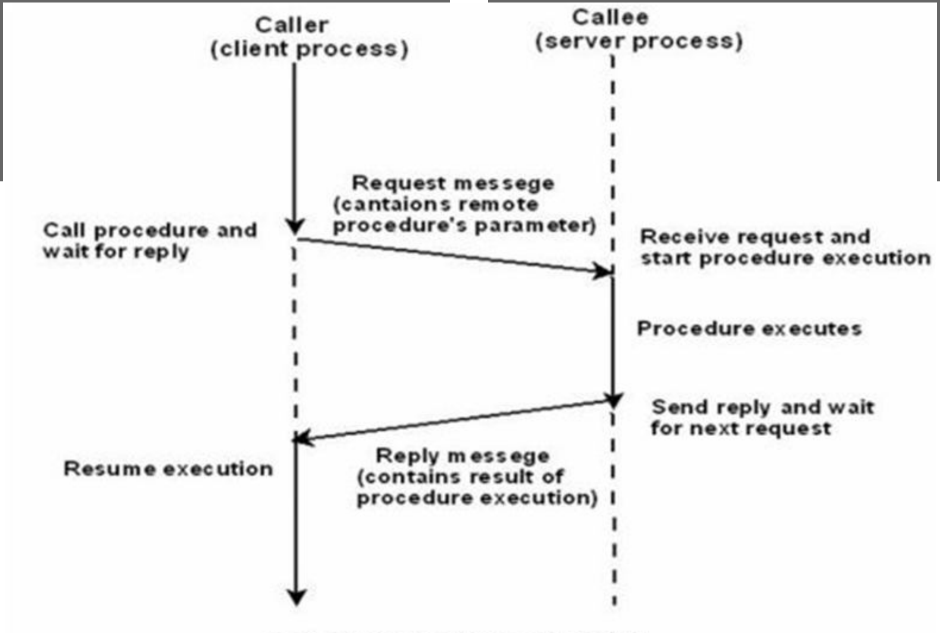
Q4. What is RPC? Explain the working of RPC.

Ans :

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

Remote Procedure Call (RPC) is a powerful technique for constructing **distributed, client-server based applications**. It is based on extending the conventional local procedure calling, so that the **called procedure need not exist in the same address space as the calling procedure**. The two processes may be on the same system, or they may be on different systems with a network connecting them.

When Making a Remote Procedure Call


Caller (client process)
Callee (server process)
Request messege (cantaions remote procedure's parameter)
Call procedure and wait for reply
Receive request and start procedure execution
Procedure executes
Send reply and wait for next request
Resume execution
Reply messege (contains result of procedure execution)

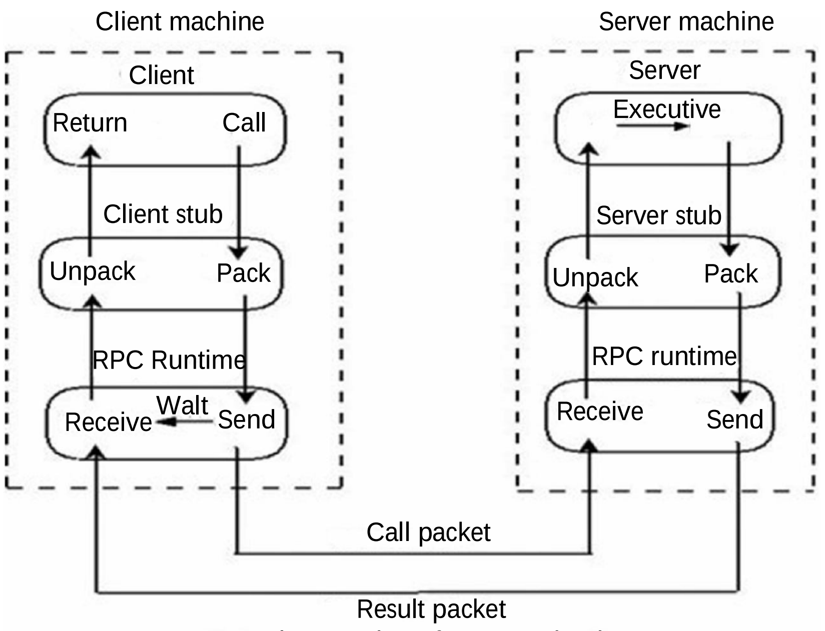
Remote procedure call model

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.

Note :

RPC is especially well suited for client-server **(e.g. query-response)** interaction in which the flow of control **alternates between the caller and called**. Conceptually, the client and server do not both execute at the same time. Instead, the thread of execution jumps from the caller to the callee and then back again.

Working of RPC



Implementation of RPC mechanism

The following steps take place during a RPC

1. A client invokes a **client stub procedure**, passing parameters in the usual way. The client stub resides within the client's own address space.
2. The client stub **marshalls(pack)** the parameters into a message. Marshalling includes converting the representation of the parameters into a standard format, and copying each parameter into the message.
3. The client stub passes the message to the transport layer, which sends it to the remote server machine.
4. On the server, the transport layer passes the message to a server stub, which **demarshalls(unpack)** the parameters and calls the desired server routine using the regular procedure call mechanism.
5. When the server procedure completes, it returns to the server stub **(e.g., via a normal procedure call return)**, which marshalls the return values into a message. The server stub then hands the message to the transport layer.
6. The transport layer sends the result message back to the client transport layer, which hands the message back to the client stub.
7. The client stub demarshalls the return parameters and execution returns to the caller.

Q5. Write about the issues of RPC. And Mention its advantages.

Ans :

RPC Issues

Issues that Must be Addressed

1. **RPC Runtime**

   RPC run-time system, is a library of routines and a set of services that handle the network communications that underlie the RPC mechanism. In the course of an RPC call, client-side and server-side run-time systems' code handle **binding, establish communications over an appropriate protocol, pass call data between the client and server, and handle communications errors.**

2. **Stub**

   The function of the stub is to **provide transparency to the programmer-written application code**.

   On the client side, the stub handles the interface between the client's local procedure call and the run-time system, marshaling and unmarshaling data, invoking the RPC run-time protocol, and if requested, carrying out some of the binding steps.

   On the server side, the stub provides a similar interface between the run-time system and the local manager procedures that are executed by the server.

3. **Binding: How does the client know who to call, and where the service resides?**

   The most flexible solution is to use dynamic binding and find the server at run time when the RPC is first made. The first time the client stub is invoked, it contacts a name server to determine the transport address at which the server resides.

   ➢ **Binding consists of two parts:**

   Naming

   Remote procedures are named through interfaces. **An interface uniquely identifies a particular service, describing the types and numbers of its arguments**. It is similar in purpose to a type definition in programming languages.

   Locating

   Finding the transport address at which the server actually resides. Once we have the transport address of the service, we can send messages directly to the server.

   A Server having a service to offer exports an interface for it. Exporting an interface registers it with the system so that clients can use it.

   A Client must import an (exported) interface before communication can begin.

Advantages

1. RPC provides **ABSTRACTION** i.e. message-passing nature of network communication is hidden from the user.
2. 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.
3. RPC enables the usage of the applications in the distributed environment, not only in the local environment.
4. With RPC code re-writing / re-developing effort is minimized.
5. Process-oriented and thread oriented models supported by RPC.

Q6. Explain about RPC model of DCE.

Ans :

RPC models and alternative methods for client-server communication.

There are several RPC models and distributed computing implementations. A popular model and implementation is the Open Software Foundation's Distributed Computing Environment (DCE). The specifications adopted in Microsoft's base system for distributed computing, DCOM.

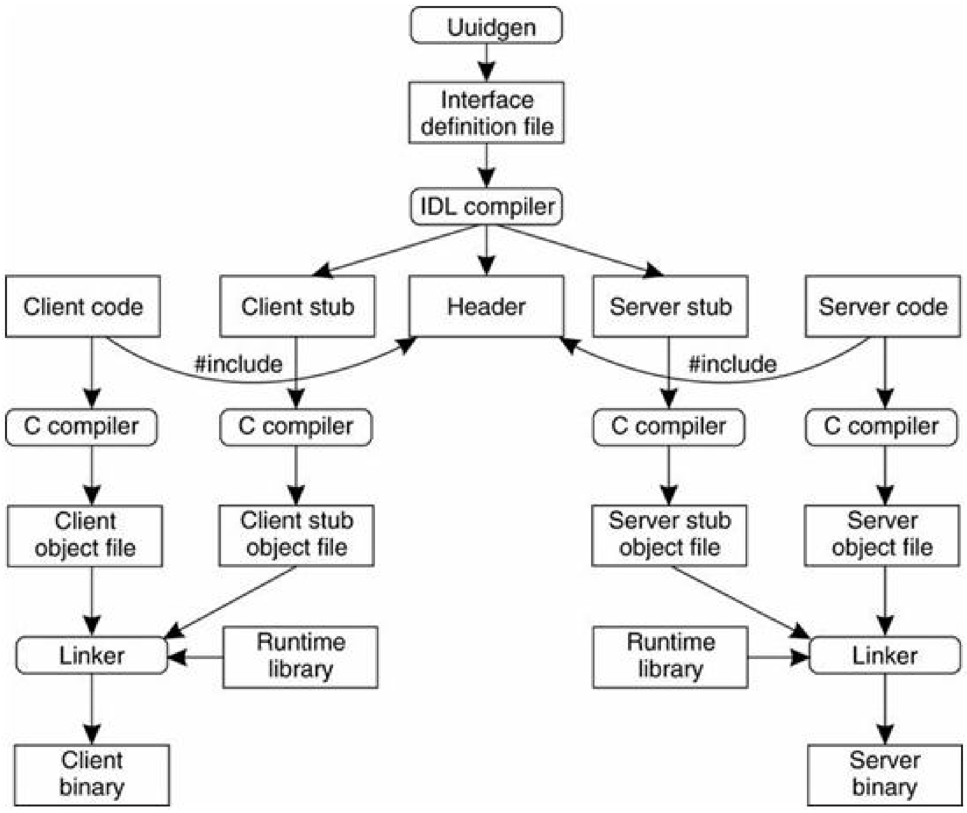
DCE is a true middleware system in that it is designed to execute as a layer of abstraction between existing (network) operating systems and distributed applications. It takes a collection of existing machines, add the DCE software, and then run distributed applications, all without disturbing existing (nondistributed) applications. The main model in DCE is Client-server model . All communication between clients and servers takes place by means of RPCs.

DCE Services

- **Distributed file service** - worldwide file system that provides a transparent way of accessing any file in the system in the same way.
- **Directory service** - keeps track of the location of all resources in the system (e.g. machines, printers, servers, data, etc.)
- **Security service** - provides access restrictions.
- **Distributed time service** - attempts to keep clocks on the different machines globally synchronized.

Writing a Client and a Server

The steps in writing a client and a server in DCE RPC.



Interface Definition Language (IDL)

An Interface Definition Language (IDL) is a language that is used to define the interface between a client and server process in a distributed system. Each interface definition language has a set of associated IDL compilers, one per supported target language.

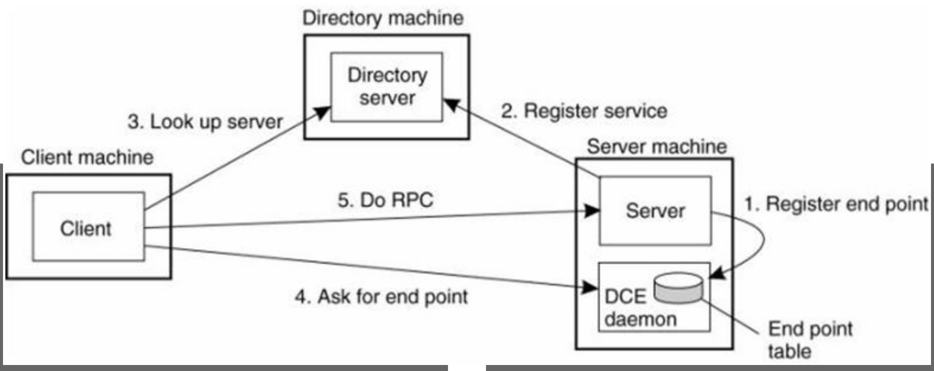
An IDL compiler compiles the interface specifications, listed in an IDL input file, into source code (e.g., C/C++, Java) that implements the low-level communication details required to support the defined interfaces.

The output of the IDL compiler consists of three files:

1. A header file (e.g., interface.h, in C terms).
2. The client stub.
3. The server stub.

Binding a Client to a Server

Client-to-server binding in DCE.



2.1.2 Message-oriented Communication

Q7. What is Message Oriented Communication? Explain about the types of message oriented communication.

Ans :

Message-oriented communication is a way of communicating between processes. Messages, which correspond to events, are the basic units of data delivered. Message-oriented communication is classified according to two factors transient or persistent communication.

Message-Oriented Transient Communication

Message-oriented model using transport layer. It uses transport-level sockets

Socket - communication end point to which an application can write data that are to be sent out over the underlying network, and from which incoming data can be read.

Berkeley Sockets

- Developed in the early 1980s at the University of California at Berkeley.
- It is an API.
- Its implementation is usually requires kernel code.
- It is the de facto standard for communications programming.
- Used for point-to-point communications between computers through an inter-systems pipe. Namely can use the UNIX read, write, close, select, etc. system calls.
- There are higher level tools for programs that span more than one machine. (e.g. RPC, DCOM).
- Supports broadcast.
- Available on every UNIX system that I know of and somewhat available in WIN32.
- Built for client/server development.

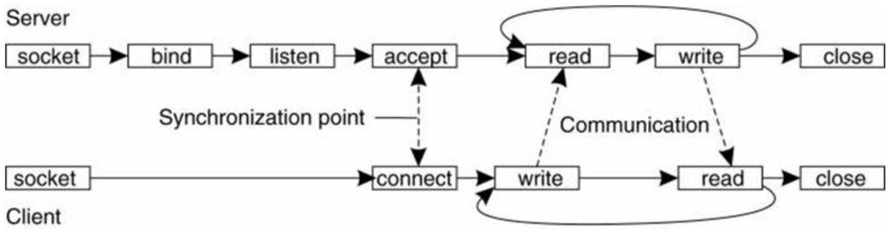
- Supports two types of communications that sit on top of the TCP Internet datagrams.
  - TCP - connection oriented, stream, reliable.
  - UDP - connection less, record oriented, unreliable.

The socket primitives for TCP/IP.

Primitive	Meaning
Socket	Create a new communication end point
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection
Write	**Send data on the connection**
Read	**Get data that was sent on the connection**

Servers execute the first four primitives - in the order given.

Connection-oriented communication pattern using sockets.



The Message-Passing Interface (MPI)

The Message Passing Interface (MPI) is a library specification for message passing. It is a standard API that can be used to create applications for high performance multicomputers. It uses specific network protocols (not TCP/IP). It is message-based communication.

Some of the message-passing primitives of MPI.

Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there is none
MPI_irecv	Check if there is an incoming message, but do not block

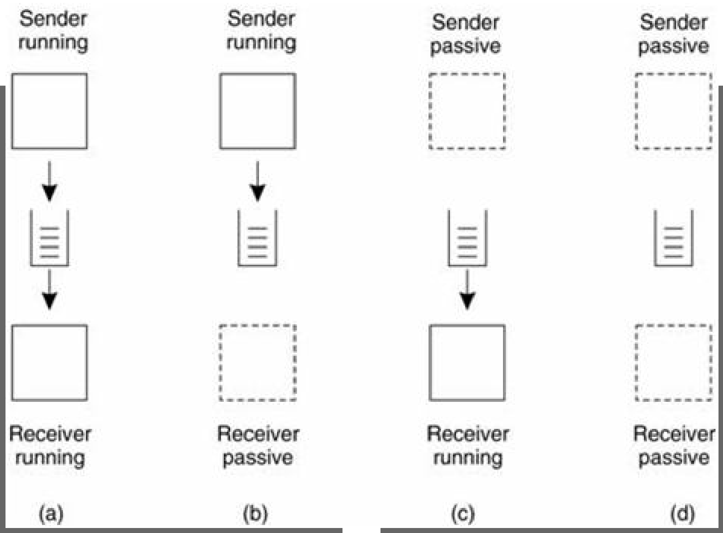
Message-Oriented Persistent Communication

It is also known as: "Message-queuing systems" or "Message-Oriented Middleware (MOM)". They support persistent, asynchronous communications. Typically, transport can take minutes (hours?) as opposed to seconds/milliseconds.

The basic idea is applications communicate by putting messages into and taking messages out of "message queues".

- Only guarantee: your message will eventually make it into the receiver's message queue.
- This leads to "loosely-coupled" communications.

Four combinations for loosely-coupled communications using queues.

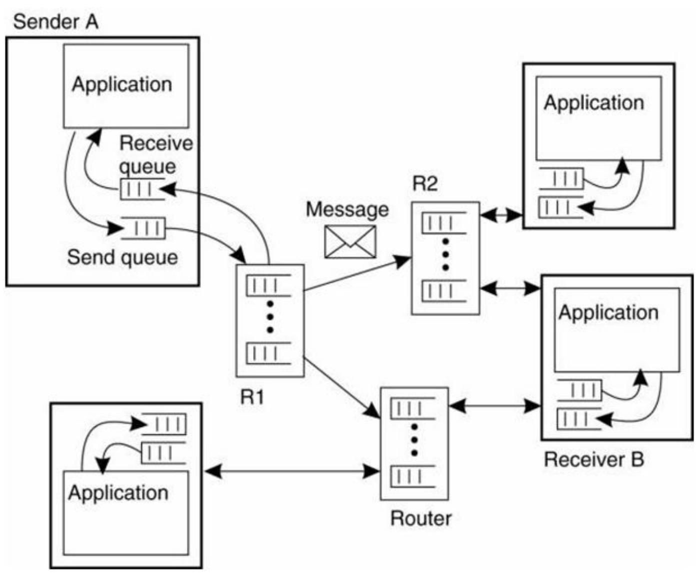


Basic Interface to a Queue in a Message-queuing System

Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block
Notify	Install a handler to be called when a message is put into the specified queue

General Architecture of a Message-Queuing System

- Messages are "put into" a source queue.
- They are then "taken from" a destination queue.
- Queues are managed by queue managers
  - i) They move a message from a source queue to a destination queue.
  - ii) Special queue managers operate as routers or relays: they forward incoming messages to other queue managers.
- The general organization of a message-queuing system with routers.



The Role of Message Brokers

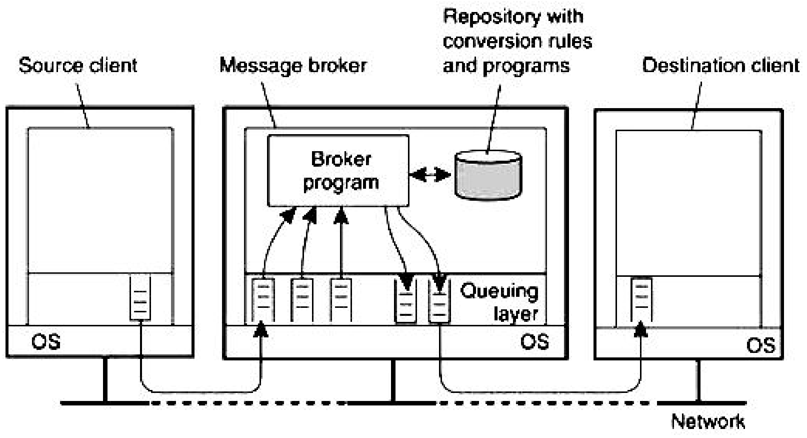
Often, there's a need to integrate new/existing apps into a "single, coherent Distributed Information System (DIS)".

Problem: different message formats exist in legacy systems. It may not be convenient to "force" legacy systems to adhere to a single, global message format in terms of cost. It is often necessary to live with diversity (there's no choice).

Solution: the "Message Broker".

Message Brokers (aka "Interface engine")

- In message-queuing systems, conversions are handled by special nodes in a queuing network, known as message brokers.
- A message broker acts as an application-level gateway in a message-queuing system.
- Purpose - convert incoming messages so that they can be understood by the destination application.
    - Note: a message broker is just another application - not considered to be an integral part of the queuing system.
- Message brokers can be simple (reformat messages) or complex (find associated applications, convert data)
- The general organization of a message broker in a message-queuing system.



Message-Queuing (MQ) Applications

General-purpose MQ systems support a wide range of applications, including:

- Electronic mail.
- Workflow.
- Groupware.
- Batch Processing.

2.1.3 Stream-oriented Communication

Q8. Explain about Stream oriented communication.

Ans :

ith RPC, RMI and MOM, the effect that time has on correctness is of little consequence. However, audio and video are time-dependent data streams-if the timing is off, the resulting "output" from the system will be incorrect.

Time-dependent information is known as "continuous media" communications.

Timing is crucial in certain forms of communication

- **Example: voice:** PCM: 1/44100 sec intervals on playback.
- **Example: video:** 30 frames per second (30-40 msec per image)

Transmission Modes

- Asynchronous transmission mode – the data stream is transmitted in order, but there's no timing constraints placed on the actual delivery (e.g., File Transfer).
- Synchronous transmission mode – the maximum end-to-end delay is defined (but data can travel faster).
- Isochronous transmission mode - data transferred "on time" – there's a **maximum** and **minimum** end-to-end delay (known as "bounded jitter").
- Known as "streams" – isochronous transmission mode is very useful for multimedia systems.

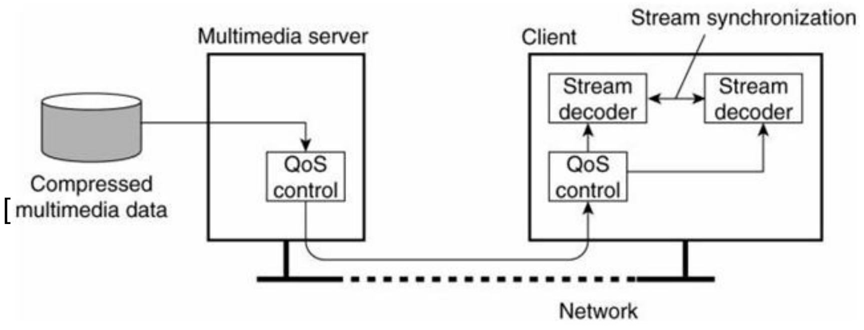
Two Types of Streams

- Simple Streams - one single sequence of data, for example: voice.
- Complex Streams - several sequences of data (substreams) that are "related" by time.

i) Think of a lip synchronized movie, with sound and pictures, together with sub-titles …

ii) This leads to data synchronization problems … which are not at all easy to deal with.

A General Architecture for Streaming Stored Multimedia Data over a Network



Quality of Service

Definition: "Ensuring that the temporal relationships in the stream can be preserved".

QoS is all about three things:

a) Timeliness,

b) Volume and

c) Reliability.

Most current operating systems and networks do not include the QoS management facilities. Bleeding edge of the discipline

Must specifying the following:

1. The required bit rate at which data should be transported.
2. The maximum delay until a session has been set up (i.e., when an application can start sending data).
3. The maximum end-to-end delay (i.e., how long it will take until a data unit makes it to a recipient).
4. The maximum delay variance, or jitter.
5. The maximum round-trip delay.

Q9. Write about Overlay Networks.

Ans :

An overlay network is a virtual network of nodes and logical links that is built on top of an existing network with the purpose to implement a network service that is not available in the existing network.

Applications of Overlays

- ➢ Routing
- ➢ Addressing
- ➢ Security
- ➢ Multicast
- ➢ Mobility

Advantages / Disadvantages

1. Do not have to deploy new equipment, or modify existing software/protocols
   i) probably have to deploy new software on top of existing software.
   ii) e.g., adding IP on top of Ethernet does not require modifying Ethernet protocol or driver
   iii) allows bootstrapping
      a) expensive to develop entirely new networking hardware/software
      b) all networks after the telephone have begun as overlay networks
2. Do not have to deploy at every node
   i) Not every node needs/wants overlay network service all the time

      e.g., QoS guarantees for best-effort traffic
   ii) Overlay network may be too heavyweight for some nodes

      e.g., consumes too much memory, cycles, or bandwidth
   iii) Overlay network may have unclear security properties

      e.g., may be used for service denial attack
   iv) Overlay network may not scale (not exactly a benefit)

      e.g. may require n2 state or communication

Two Approaches to Organization

1. Nodes may organize themselves into a tree, meaning that there is a unique (overlay) path between every pair of nodes.
2. Nodes organize into a mesh network in which every node will have multiple neighbors and there exist multiple paths between every pair of nodes more robust

2.1.4 Multicast Communication

Q10. Explain in detail about Multicast communication in distributed systems.

Ans :

Multicasting is the process whereby a source host or protocol entity sends a packet to multiple destinations simultaneously using a single, local 'transmit' operation.

There are two categories in multicast communication:

- Unicasting
- Broadcasting

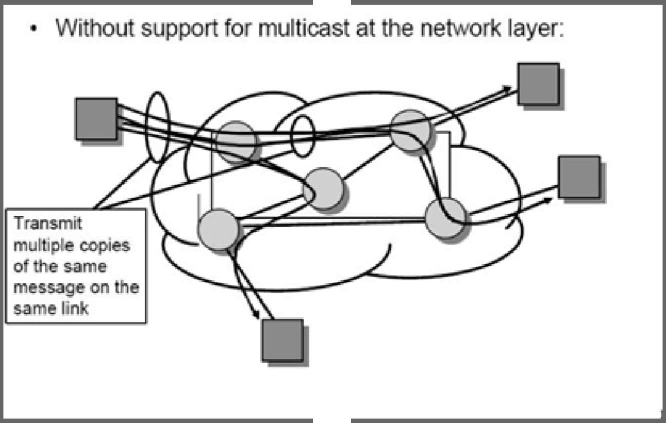
Multicast communications refers to one-to-many or many-to many communications. It implements a one-to-many send operation:

We can implement Multi cast on

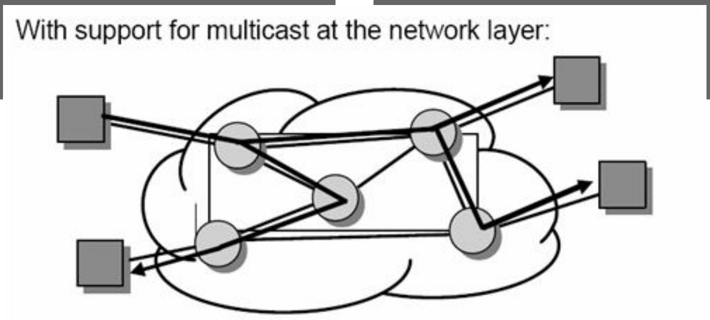
- Network Layer (IP)
- Application Layer

Need for Multicast at Network Layer (IP Multicast)

The following figures describe how multicasting can be used with or without support of network layer.


• Without support for multicast at the network layer:
Transmit multiple copies of the same message on the same link

The above figure shows that the usage of multicasting with network layer support. In this the multiple copies of the same packet are transmitted on the same link several times, because here the network layer is not active to determine the proper route to the destination.

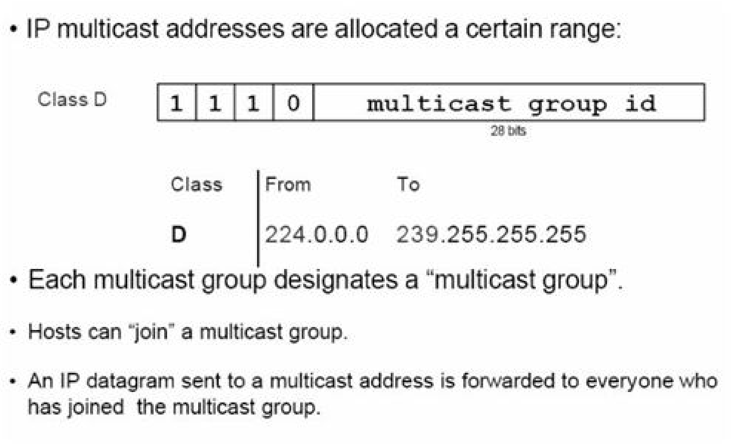


Requires a set of mechanisms at the network layer:

(1) Routers must be able to send multiple copies of same packet
(2) Multicast routing algorithms needed to build up a dissemination tree

The above figure shows that the multicasting is used in the network layer .here the routers are able to send multiple copies of same packet. With the help of multicast routing algorithms the network layer built destination tree to carry the packets.

Multicast works at IP Level



- IP multicast addresses are allocated a certain range:

Class D
1
1
1
0
multicast group id
28 bits

Class	From	To
D	224.0.0.0	239.255.255.255

- Each multicast group designates a "multicast group".
- Hosts can "join" a multicast group.
- An IP datagram sent to a multicast address is forwarded to everyone who has joined the multicast group.

Multicast at Application Layer

Application layer provides multicast functionality above IP layer. The data is transmitted between neighbours in the overlay networks. No multicast needed in overlay network.

Advantages

1. Scalability
   i) Routers do not maintain per-group state
   ii) End systems do, but they participate in very few groups
2. Easier to Deploy
   i) Only requires adding software to end hosts
   ii) Potentially simplifies support for higher level functionality
3. Use hop-by-hop approach, but end hosts are routers
   i) Leverage computation and storage of end systems
   ii) e.g., packet buffering, transcoding of media streams
4. Leverage solutions for unicast congestion control and reliability

Overlay-based Approaches for Multicasting

Build an overlay mesh network and embed trees into the mesh:

- Build a shared tree:
- Build a graph with well-known properties
  - N-dimensional torus: CAN
  - Hypercube inspired: Chord
  - Triangulation: Delaunay Triangulation

Properties of the Overlay Graph

1. Number of neighbors (routing table size)
   i) Many Distributed Hash Tables (DHTs), hypercubes: O(log N) (max.)
   ii) Triangulation graphs: O(N) (max.), 6 (avg.)
   iii) Meshes, trees: no *a priori* bound, but bounds can be enforced

2. Path lengths in the overlay
   i) Many DHTs, hypercubes: O(log N) (max.)
   ii) Triangulation graphs: O(N) (max.), O(vN) (best case avg.)
   iii) Meshes, trees: no a priori bound

2.2 Naming

2.2.1 Names

Q11. Write about names in distributed environment.

Ans :

Names

1. A **name** is human-readable value (usually a string) that can be **resolved** to an identifier or address like, Internet domain name, file pathname, process number

   e.g. /etc/passwd, http://www.csc.liv.ac.uk/
2. For many purposes, names are preferable to identifiers because the binding of the named resource to a physical location is deferred and can be changed because they are more meaningful to users.
3. Resource names are **resolved** by name services to give identifiers and other useful attributes
4. Naming is about mapping between names, addresses, identifiers and the referred entities

Examples

Resource	Name	Identifiers
File	Pathname	File within a given file system
Process	Process ID	Process on a given computer
Port	Port number	IP port on a given computer

Uses of Names

1. Names are used to denote entities in a distributed system.
2. To operate on an entity, we need to access it at an **access point**.
3. Access points are entities that are named by means of an **address**.

Flat Naming

1. Identifiers are simply random bit strings referred to as unstructured or flat names.
2. Property: it does not contain any information on how to locate the access point of its associated entity.

2.2.2 Identifiers

Q12. What are identifiers in distributed environment.

Ans :

Identifiers

A name that has no meaning at all; it is just a random string. Pure names can be used for comparison only.

Identifier: A name having the following properties:

P1 Each identi_er refers to at most one entity

P2 Each entity is referred to by at most one identifier

P3 An identi_er always refers to the same entity (prohibits reusing an identifier)

An identifier need not necessarily be a pure name, i.e., it may have content.

Identifiers use Special type of (usually, computer readable) name with the following properties:

- An id refers to at most one entity
- Each entity is referred by at most one id
- An id always refers to the same entity (never reused)

Identifier includes or can be transformed to an address for an object

e.g. NFS file handle, Java RMI remote object reference, etc.

Properties of a True Identifier

- An identifier refers to at most one entity.
- Each entity is referred to by at most one identifier.
- An identifier always refers to the same entity

2.2.3 Addresses

Q13. Write a note address in distributed system.

Ans :

Before you can send a message, you must know the destination address. It is extremely important to understand that each computer has several addresses, each used by a different layer. One address is used by the data link layer, another by the network layer, and still another by the application layer.

An address is a name that refers to an access point of an entity. For eg. a server's address consists of an IP address and a port number. An entity may have multiple access points and addresses. Like A person has several phone numbers (e.g. work, home, mobile).

An entity may change its access points for eg, a service is moved to a different host, a person changes its email address after changing his job.

In order for a client to send a message to a server, it must know the server's address. If there is only one process running on the destination machine, the kernel will know what to do with the incoming message give it to the one and only process running there.

If there are several processes running on the destination machine, to which one gets the message. The kernel has no way of knowing. A scheme that uses network addresses to identify processes means that only one process can run on each machine. While this limitation is not fatal, it is sometimes a serious restriction.

2.2.4 Flat Naming

Q14. Write a note on Flat Naming service.

Ans :

Identifiers are simply random bit strings referred to as unstructured or flat names.

Property

It does not contain any information on how to locate the access point of its associated entity.

Locating an Entity

Broadcasting

1. A message containing the identifier of the entity is broadcast to each machine and each machine is requested to check whether it has that entity.
2. Only the machines that can offer an access point for the entity send a reply message containing the address of that access point.

   e.g. used in the Internet Address Resolution Protocol (ARP) to find the data-link address of a machine when given only an IP address.

   - 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.

Multicasting

- Internet supports network-level multicasting by allowing hosts to join a specific multicast group.
- These groups are identified by a multicast address.
- When a host sends a message to a multicast address, the network layer provides a best-effort service to deliver that message to all group members. See Deering and Cheriton (1990) and Deering et al. (1996).

Forwarding Pointers

When an entity moves from A to B, it leaves behind in A a reference to its new location at B.

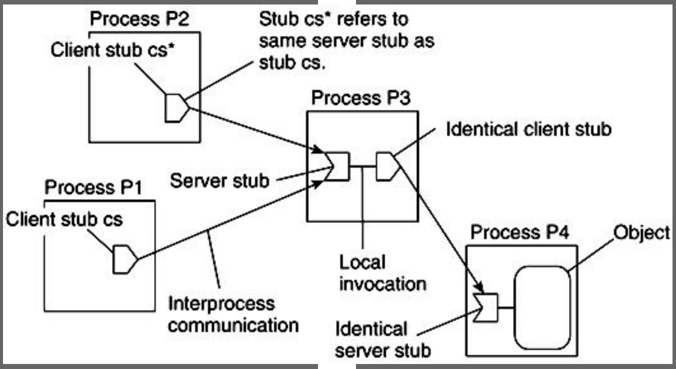
Advantages

Simple

Disadvantages

1. No special measures are taken, a chain for a highly mobile entity can become so long that locating that entity is prohibitively expensive.
2. All intermediate locations in a chain will have to maintain their part of the chain of forwarding pointers as long as needed.
3. A drawback is the vulnerability to broken links.

The principle of forwarding pointers using (client stub, server stub) pairs.



- When an object moves from address space A to B, it leaves behind a client stub in its place in A and installs a server stub that refers to it in B.
- Migration is completely transparent to a client.
- The only thing the client sees of an object is a client stub.
- A client's request is forwarded along the chain to the actual object.

2.2.5 Structured Naming

Q15. What are name spaces? Explain about them.

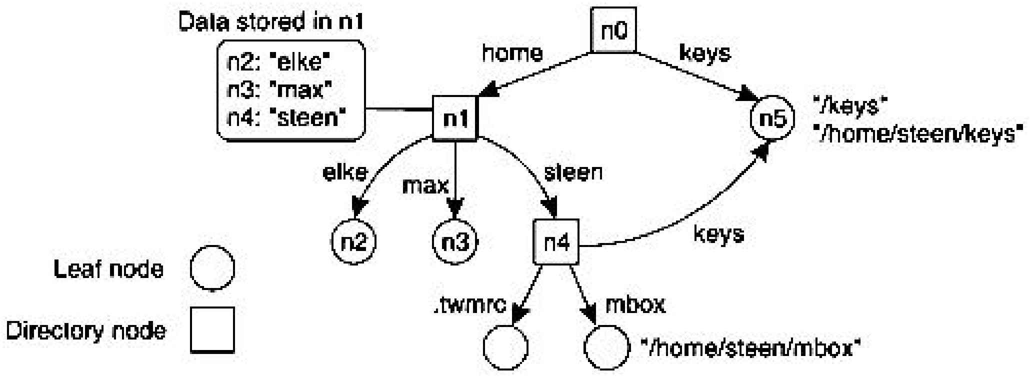
Ans :

Name Spaces

Names are commonly organized into what is called a name space. The name spaces for structured names can be represented as a labelled, directed graph with two types of nodes.

1. A leaf node represents a named entity and has the property that it has no outgoing edges.
   - ➢ generally stores attribute information on the entity it is representing - for example, its address - so that a client can access it.
   - ➢ could store the state of that entity, such as in the case of file systems in which a leaf node actually contains the complete file it is representing. We return to the contents of nodes below.
2. Directory node has a number of outgoing edges, each labeled with a name of other nodes
3. A directory node stores a directory table in which an outgoing edge is represented as a pair (edge label, node identifier).
4. Directory nodes can also have attributes, besides just storing a directory table with (edge label, node identifier) pairs.

The following is a general naming graph with a single root node.



The naming graph has one node, namely n0, which has only outgoing and no incoming edges called the root (node) of the naming graph. Each path in a naming graph can be referred to by the sequence of labels corresponding to the edges in that path, such as

N:<label-1, label-2, ..., label-n> where N refers to the first node in the path.

Such a sequence is called a path name. If the first node in a path name is the root of the naming graph, it is called an absolute path name ELSE, it is called a relative path name.

NOTE: path naming similar to file systems.

But path names in file systems are represented as a single string in which the labels are separated by a special separator character, such as a slash ("/").

This character is also used to indicate whether a path name is absolute.

- ➢ E.g, instead of using n0:<home, steen, mbox>, the actual path name common practice to use its string representation /home/steen/mbox.

If several paths that lead to the same node, it can be represented by different path names.

- ➢ E.g, node n5 can be referred to by /home/steen/keys or /keys. The string representation of path names can be equally well applied to naming graphs other than those used for only file systems.

More general example of naming is Plan 9 operating system. All resources, such as processes, hosts, I/O devices, and network interfaces, are named in the same fashion as traditional files. Approach is analogous to implementing a single naming graph for all resources in a distributed system.

Q16. Write about name resolution techniques.

Ans :

Name Resolution

To resolve a name we need a directory node. It uses the mechanism called **Closure mechanism,** The mechanism to select the implicit context from which to start name resolution:

- www.cs.vu.nl: start at a DNS name server
- /home/steen/mbox: start at the local NFS file server (possible recursive search)
- 0031204447784: dial a phone number
- 130.37.24.8: route to the VU's Web server

A closure mechanism may also determine how name resolution should proceed.

Linking and Mounting

Name Linking : allow multiple absolute path names to refer to the same node.

Two types of linking mechanisms are there,

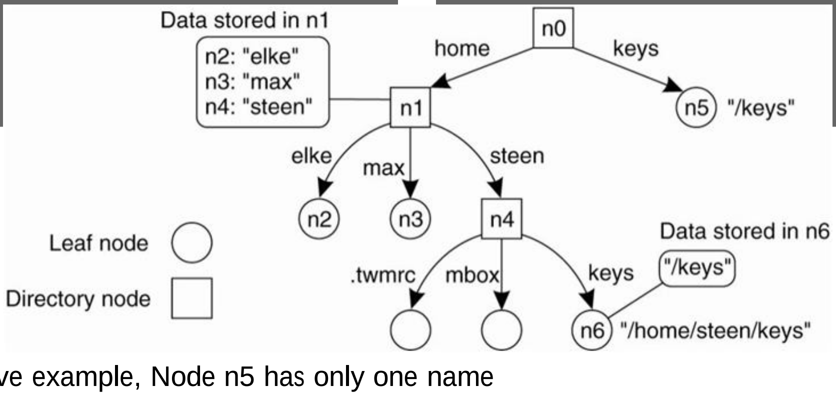
i) **Hard link:** What we have described so far as a **path name**: a name that is resolved by following a specific path in a naming graph from one node to another.

ii) **Soft link:** Allow a node O to contain a name of another node:

- First resolve O's name (leading to O)
- Read the content of O, yielding name
- Name resolution continues with name

The name resolution process determines that we read the content of a node, in particular, the name in the other node that we need to go to.One way or the other, we know where and how to start name resolution given name

The concept of a symbolic link explained in a naming graph.

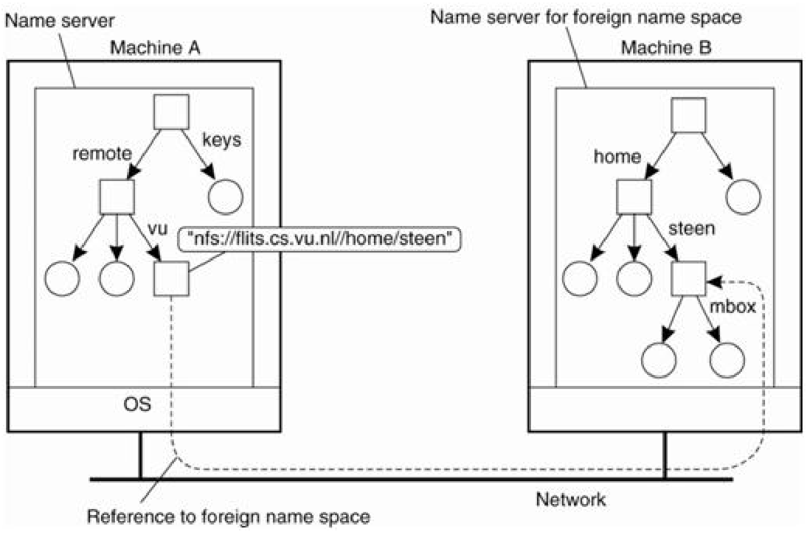


In the above example, Node n5 has only one name

Mounting :allow a node to refer to a node from a different name space. Mounting remote name spaces ("remote symbolic link") needs :

- A specific protocol (e.g. NFS)
- At a certain mount point of a given server
- A name, containing access protocol, remote server, foreign mounting point (e.g. URL)

Mounting remote name spaces through a specific access protocol.


Name server
Machine A
remote
keys
vu
"nfs://flits.cs.vu.nl//home/steen"
OS
Name server for foreign name space
Machine B
home
steen
mbox
Network
Reference to foreign name space

NOTE:

- The root directory has a number of user-defined entries, including a sub directory called /remote.
- This subdirectory includes mount points for foreign name spaces such as the user's home directory at the Vrije Univer site it.
- A directory node named /remote/vu is used to store the URL nfs://flits.cs.vu.nl//home/steen.

Consider the name/remote/vu/mbox.

- This name is resolved by starting in the root directory on the client's machine and continues until the node /remote/vu is reached.
- The process of name resolution then continues by returning the URL nfs://flits.cs.vu.nl//home/steen, leading the client machine to contact the file server flits.cs.vu.nl by means of the NFS protocol, and to subsequently access directory /home/steen.
- Name resolution can then be continued by reading the file named mbox in that directory, after which the resolution process stops.

The Implementation of a Name Space

Basic issue: Distribute the name resolution process as well as name space management across multiple machines, by distributing nodes of the naming graph.

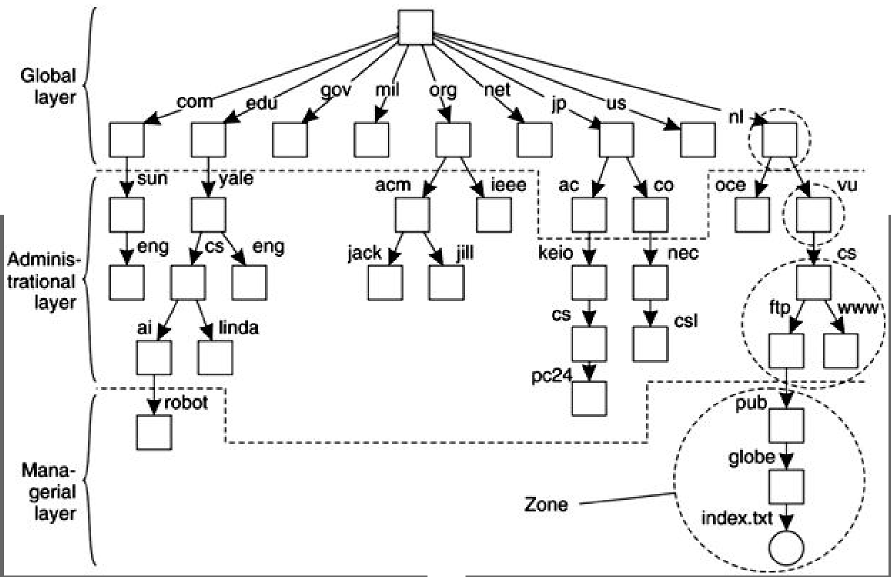
Name Space Distribution

Consider a hierarchical naming graph and distinguish three levels:

- **Global Level:** Consists of the high-level directory nodes. Main aspect is that these directory nodes have to be jointly managed by different administrations
- **Administrational Level:** Contains mid-level directory nodes that can be grouped in such a way that each group can be assigned to a separate administration.
- **Managerial Level:** Consists of low-level directory nodes within a single administration. Main issue is effectively mapping directory nodes to local name servers.

To make matters more concrete, Fig. 5-13 shows an example of the partitioning of part of the DNS name space, including the names of files within an organization that can be accessed through the Internet, for example, Web pages and transferable files. The name space is divided into non overlapping parts, called zones in DNS (Mockapetris, 1987). A zone is a part of the name space that is implemented by a separate name server.

An example partitioning of the DNS(Domain Name System) name space, including Internet-accessible files, into three layers.



NOTE:

- Name space is divided into nonoverlapping parts, called zones in DNS.
- A zone is a part of the name space that is implemented by a separate name server.

Implementation of Name Resolution

To resolve the name resolution, we want to resolve absolute pathname:

root:<nl, vu, cs, ftp, pub, globe, index.html >

It could alternatively use URL notation : ftp://ftp.cs.vu.nl/pub/globe/index.html

Iterative Name Resolution

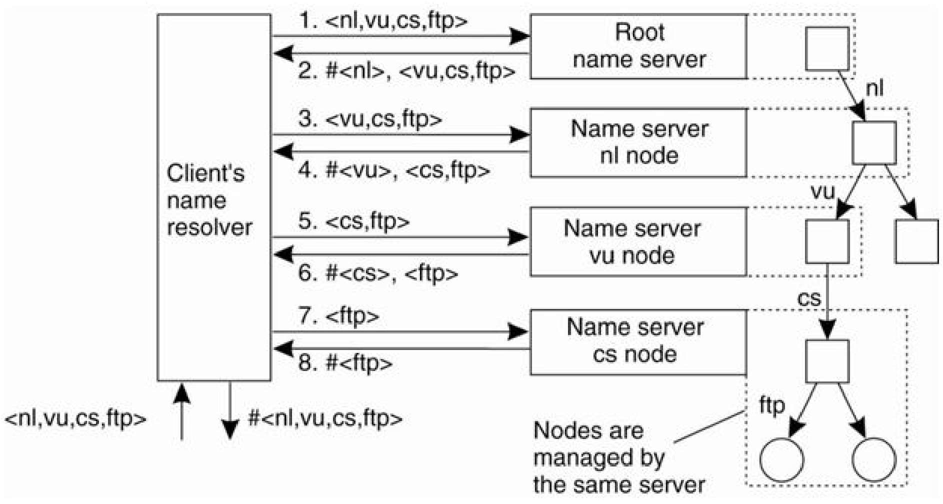
1. The address of the root server must be well known
2. A name resolver hands over the complete name to the root name server
3. The root server will resolve the path name as far as it can, and return the result to the client.
   - the root server can resolve only the label nl, for which it will return the address of the associated name server.
4. The client passes the remaining path name (i.e., **nl:<vu, cs, ftp, pub, globe, index.html >**) to that name server.
5. This server can resolve only the label vu, and returns the address of the associated name server, along with the remaining path name vu:<cs, ftp, pub, globe, index.html >.
6. The client's name resolver will then contact this next name server, which responds by resolving the label cs, and also ftp, returning the address of the FTP server along with the path name ftp:<pub, globe, index.html >.

7. The client then contacts the FTP server, requesting it to resolve the last part of the original path name.

8. The FTP server will resolve the labels pub, globe, and index.html, and transfer the requested file (in this case using FTP).

   ➢ (The notation #<cs> is used to indicate the address of the server responsible for handling the node referred to by <cs>.)

   The principle of iterative name resolution.

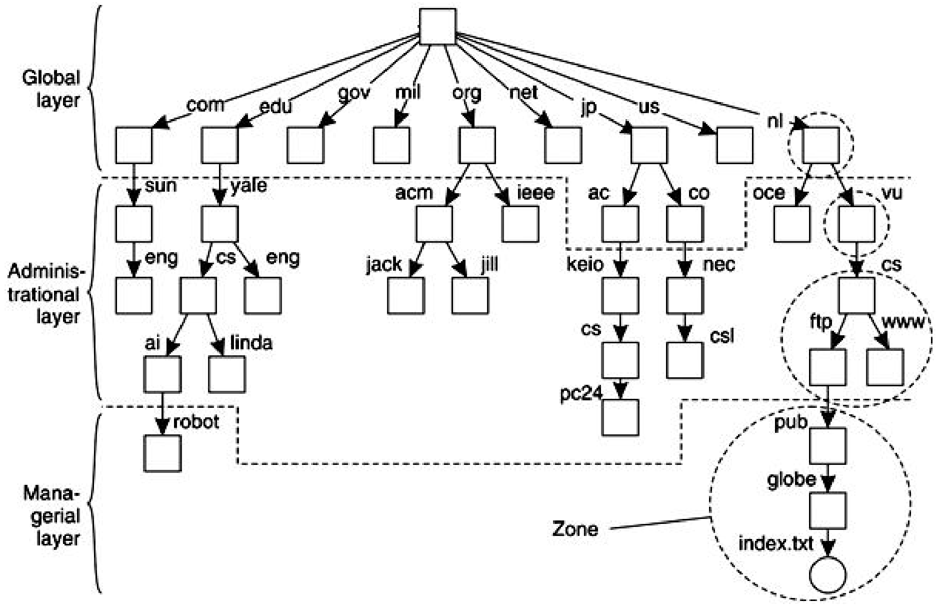


Q17. Explain about DNS.

Ans :

The Domain Name System

The DNS name space is hierarchically organized as a rooted tree.



- A label is a case-insensitive string made up of alphanumeric characters.
- A label has a maximum length of 63 characters; the length of a complete path name is restricted to 255 characters.
- The string representation of a path name consists of listing its labels, starting with the rightmost one, and separating the labels by a dot (“.”).
- The root is represented by a dot.

  e.g, the path name root:<nl, vu, cs, flits>, is represented by the string flits.cs.vu.nl. , which includes the rightmost dot to indicate the root node - omitted for readability.
- Because each node in the DNS name space has exactly one incoming edge (with the exception of the root node, which has no incoming edges), the label attached to a node’s incoming edge is also used as the name for that node.
- A subtree is called a domain; a path name to its root node is called a domain name.

  Note - just like a path name, a domain name can be either absolute or relative.

The most important types of resource records forming the contents of nodes in the DNS name space.

- SOA (start of authority) resource record - contains information such as an e-mail address of the system administrator responsible for the represented zone, the name of the host where data on the zone can be fetched, and so on.
- A (address) record - contains an IP address for the Internet host to allow communication.
- If a host has several IP addresses, as is the case with multi-homed machines, the node will contain an A record for each address.
- MX (mail exchange) record - a symbolic link to a node representing a mail server.
- SRV records - contain the name of a server for a specific service. The service itself is identified by means of a name along with the name of a protocol. SRV records are defined in Gulbrandsen (2000).
- NS (name server) records - an NS record contains the name of a name server that implements the zone represented by the node.
- CNAME record - contains the canonical name of a host . DNS distinguishes aliases from what are called canonical names. Each host is assumed to have a canonical, or primary name. The name of the node storing such a record is thus the same as a symbolic link
- PTR (pointer) record - DNS maintains an inverse mapping of IP addresses to host names by means of PTR (pointer) records. To accommodate the lookups of host names when given only an IP address, DNS maintains a domain named in-addr.arpa, which contains nodes that represent Internet hosts and which are named by the IP address of the represented host.
- HINFO (host info) record - used to store additional information on a host such as its machine type and operating system.
- TXT records - used for any other kind of data that a user finds useful to store about the entity represented by the node.

DNS Implementation

- Each zone is implemented by a name server, which is virtually always replicated for availability.
- A DNS database is implemented as a (small) collection of files -the most important contain all the resource records for all the nodes in a particular zone.
- Nodes are identified by means of their domain name, by which the notion of a node identifier reduces to an (implicit) index into a file.

2.2.6 Attribute Based Naming

Q18. Explain about directory services.

Ans :

Directory Services

Directory Server provides a central repository for storing and managing information. Almost any kind of information can be stored, from identity profiles and access privileges to information about application and network resources, printers, network devices and manufactured parts. Information stored in Directory Server can be used for the authentication and authorization of users to enable secure access to enterprise and Internet services and applications. Directory Server is extensible, can be integrated with existing systems, and enables the consolidation of employee, customer, supplier, and partner information.

In many cases, it is much more convenient to name, and look up entities by means of their **attributes** which are known as traditional **directory services .**

Lookup operations can be extremely expensive, as they are required to match requested attribute values, against actual attribute values . So inspect all entities (in principle).

Inorder to Implement basic directory service as database, combine with traditional structured naming system.

A directory service can be considered an extension of a database, directory services generally have the following characteristics:

1. **Hierarchical Naming Model**

   A hierarchical naming model uses the concept of containment to reduce ambiguity between names and simplify administration. The name for most objects in the directory is relative to the name of some other object which conceptually contains it. For example, the name of an object representing an employee of a particular company contains the name of the object representing the company, and the name of the company might contain the name of the objects representing the country where the company operates, e.g. cn=John Smith, o=Example Corporation, c=US. Together the names of all objects in the directory service form a tree, and each Directory Server holds a branch of that tree, which in the Sun Java System Directory Server documentation is also referred to as a suffix.

2. **Extended Search Capability**

   Directory services provide robust search capabilities, allowing searches on individual attributes of entries.

3. **Distributed information model**

   A directory service enables directory data to be distributed across multiple servers within a network.

4. **Shared Network Access**

   While databases are defined in terms of APIs, directories are defined in terms of protocols. Directory access implies network access by definition. Directories are designed specifically for shared access among applications. This is achieved through the object-oriented schema model. By contrast, most databases are designed for use only by particular applications and do not encourage data sharing.

5. **Replicated Data**

   Directories support replication (copies of directory data on more than one server) which make information systems more accessible and more resistant to failure.

6. **Datastore Optimized for Reads**

   The storage mechanism in a directory service is generally designed to support a high ratio of reads to writes.

7. **Extensible Schema**

   The schema describes the type of data stored in the directory. Directory services generally support the extension of schema, meaning that new data types can be added to the directory.

About Enterprise-Wide Directory Services

Directory Server provides enterprise-wide directory services, meaning it provides information to a wide variety of applications. Until recently, many applications came bundled with their own proprietary user databases, with information about the users specific to that application. While a proprietary database can be convenient if you use only one application, multiple databases become an administrative burden if the databases manage the same information.

For example, suppose your network supports three different proprietary E-mail systems, each system with its own proprietary directory service. If users change their passwords in one directory, the changes are not automatically replicated in the others.

An enterprise-wide directory service solves the n + 1 directory problem by providing a single, centralized repository of directory information that any application can access. However, giving a wide variety of applications access to the directory requires a network-based means of communicating between the applications and the directory. Directory Server provides two ways in which applications can access its enterprise-wide directory:

- Lightweight Directory Access Protocol (LDAP)
- Directory Services Markup Language (DSML)

About LDAP

LDAP provides a common language that client applications and servers use to communicate with one another. LDAP-based applications can easily search, add, delete and modify directory entries. LDAP is a "lightweight" version of the Directory Access Protocol (DAP) defined in the ISO/ ITU-T X.500 standard. DAP gives any application access to the directory via an extensible and robust information framework, but at an expensive administrative cost. DAP does not use the Internet standard TCP/IP protocol and has complicated directory-naming conventions. LDAP preserves the best features of DAP while reducing administrative burdens. LDAP uses an open directory access protocol running over TCP/IP and uses simplified encoding methods. It retains the X 500 standard data model and can support millions of entries for a comparatively modest investment in hardware and network infrastructure.

About DSML

DSML is a markup language that enables you to represent directory entries and commands in XML. This means that XML-based applications using HTTP can take advantage of directory services while staying within the existing web infrastructure. Directory Server implements version 2 of the DSML standard (DSMLv2).

2.3 Synchronization

2.3.1 Clock Synchronization

Q19. What is clock synchronization in DS. Write about physical clocks.

Ans :

In a centralized system, time is unambiguous. When a process wants to know the time, it makes a system call and the kernel tells it. If process A asks for the time. and then a little later process B asks for the time, the value that B gets will be higher than (or possibly equal to) the value A got. It will certainly not be lower.

In a distributed system, clock synchronization is complicated.

There are three types of clocks used in distributed system

- Physical clocks
- Logical clocks
- Vector clocks

Physical Clocks

In some systems (e.g., real-time systems), the actual clock time is important. Under these circumstances, external physical clocks are needed. For reasons of efficiency and redundancy, multiple physical clocks are generally considered desirable, which yields two problems: (1) How do we synchronize them with realworld clocks. and (2) How do we synchronize the clocks with each other.

This can be achieved by Universal Coordinated Time (UTC):

The UTC works based on the number of transitions per second of the cesium 133 atom .At present, the real time is taken as the average of some 50 cesium-clocks around the world. It introduces a leap second from time to time to compensate that days are getting longer.

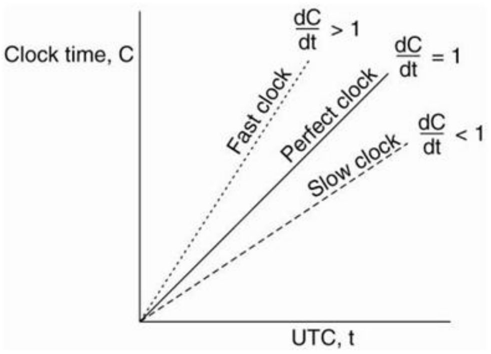
UTC is **broadcast** through short wave radio and satellite.

- Satellites can give an accuracy of about $\pm$ 0.5 ms.

Suppose we have a distributed system with a UTC-receiver somewhere in it $\Rightarrow$ we still have to distribute its time to each machine.

Basic Principle

- Every machine has a timer that generates an interrupt H times per second.
- There is a clock in machine p that **ticks** on each timer interrupt. Denote the value of that clock by $C_p(t)$, where t is UTC time.
- Ideally, we have that for each machine p, $C_p(t) = t$, or, in other words, $dC/dt = 1$.



$$1 - \rho = \frac{dC}{dt} \leq 1 + \rho$$

Example

Assuming that the clocks of the satellites are accurate and synchronized:

- t takes a while before a signal reaches the receiver.
- The receiver's clock is definitely out of synch with the satellite
- Δ_r is unknown deviation of the receiver's clock.
- x_r, y_r, z_r are unknown coordinates of the receiver.
- T_i is time stamp on a message from satellite i
- $\Delta_i = (T_{now} - T_i) + \Delta_r$ is measured delay of the message sent by satellite i.
- Measured distance to satellite i:c $X\Delta_i$ (c is speed of light)
- Real distance is $d_i = \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2}$

4 satellites $\Rightarrow$ 4 equations in 4 unknowns (with Δ_r as one of them):

$$d_i + c\Delta_r = c\Delta_i$$

Q20. Write about NTP algorithm in physical clocks.

Ans :

Network Time Protocol (NTP)

Many algorithms are proposed to do clock synchronization in distributed systems. The most commonly used algorithm is Network Time Protocol Algorithm. NTP is a protocol designed to synchronize the clocks of computers over a network.

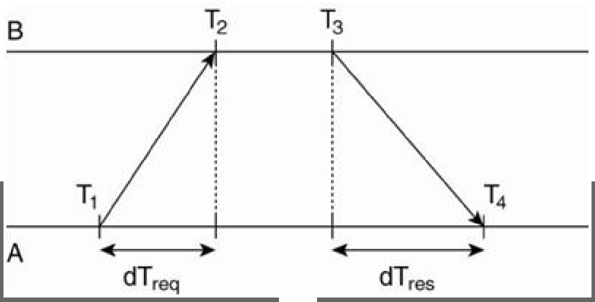
It works based on the following principles.

Clock Synchronization Principles

Principle I: Every machine asks a **time server** the accurate time at least once every $\delta/(2\rho)$ seconds (**Network Time Protocol**). But we need an accurate measure of round trip delay, including interrupt handling and processing incoming messages.

Principle II: Let the time server scan all machines periodically, calculate an average, and inform each machine how it should adjust its time relative to its present time.

Getting the Current time from a Time Server



1. A will send a request to B, time stamped with value T_1.
2. B will record the time of receipt T_2 (taken from its own local clock), and return a response timestamped with value T_3, piggybacking the previously recorded value T_2.
3. A records the time of the response's arrival, T_4.

➢ Assume that the propagation delay from A to B is the same as B to A, meaning that $T_2 - T_1 \approx T_4 - T_3$.

➢ A can estimate its offset relative to B as

$$\theta = T_3 - \frac{(T_2 - T_1) + (T_4 - T_3)}{2} = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$$

➢ If A's clock is fast $\Rightarrow \theta < 0$, A cannot set its clock backward which is not allowed.

➢ Must slow down clock gradually.

e.g. timer is set to generate 100 interrupts per second.

  ➢ Each interrupt would add 10 msec to the time.

  ➢ When slowing down, the interrupt routine adds only 9 msec each time until the correction has been made.

  ➢ Vice versa for advancing

➢ In the case of the network time protocol (NTP), this protocol is set up pair-wise between servers. In other words, B will also probe A for its current time.

➢ The offset θ is computed, along with the estimation δ for the delay:

$$\delta = \frac{(T_2 - T_1) + (T_4 - T_3)}{2}$$

➢ Eight pairs of (θ, δ) values are buffered, finally taking the minimal value found for δ as the best estimation for the delay between the two servers, and subsequently the associated value θ as the most reliable estimation of the offset.

Clock Accuracy

Some cocks are more accurate than NTP. NTP divides servers into strata.

When A contacts B, it will only adjust its time if its own stratum level is higher than that of B. After synchronization, A's stratum level will become one higher than that of B. Due to the symmetry of NTP, if A's stratum level was lower than that of B, B will adjust itself to A.

Stratum 0

Devices such as atomicclocks, GPS clocks or other radio clocks. Stratum-0 devices are not attached to the network; instead they are locally connected to computers.

Stratum 1

Computers attached to Stratum 0 devices. Normally they act as servers for timing requests from Stratum 2 servers via NTP. Also referred to as timeservers.

Stratum 2

Computers that send NTP requests to Stratum 1 servers. Normally a Stratum 2 computer will reference a number of Stratum 1 servers and use the NTP algorithm to gather the best data sample, dropping any Stratum 1 servers that seem obviously wrong.Stratum 2 computers act as servers for Stratum 3 NTP requests.

Stratum 3 and higher

These computers employ exactly the same NTP functions of peering and data sampling as Stratum 2, and can themselves act as servers for higher strata, potentially up to 16 levels. NTP (depending on what version of NTP protocol in use) supports up to 256 strata.

Q21. Explain Berkeley Algorithm.

Ans :

The Berkeley Algorithm

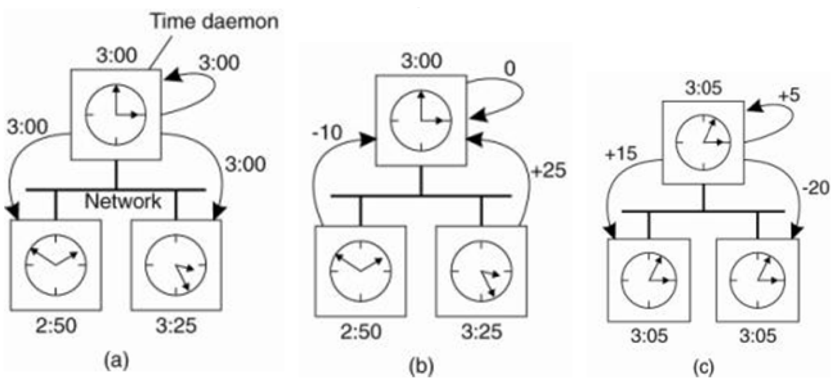
Many algorithms such as NTP, the time server is passive. - Other machines periodically ask it for the time. It is Opposite to Berkeley UNIX time server which is active, and polling every machine from time to time to ask what time it is there.

Based on the answers, it computes an average time and tells all the other machines to advance their clocks to the new time or slow their clocks down until some specified reduction has been achieved. This method is suitable for a system in which no machine has a WWV receiver. The time daemon's time must be set manually by the operator periodically.

(a) The time daemon asks all the other machines for their clock values.

(b) The machines answer.

(c) The time daemon tells everyone how to adjust their clock.



1. at 3:00, the time daemon tells the other machines its time and asks for theirs.
2. they respond with how far ahead or behind the time daemon they are.
3. the time daemon computes the average and tells each machine how to adjust its clock .

2.3.2 Logical Clocks

Q22. Write about Logical clocks? Explain Lamport's.

Ans :

In Logical clocks We first need to introduce a notion of ordering before we can order anything.

It use the **happened-before** relation on the set of events in a distributed system:

- If a and b are two events in the same process, and a comes before b, then a '!b.
- If a is the sending of a message, and b is the receipt of that message, then a'!b
- If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$

This introduces a **partial ordering of events** in a system with concurrently operating processes.

Problem:

How do we maintain a global view on the system's behavior that is consistent with the happened before relation?

Solution:

Attach a timestamp C(e) to each event e, satisfying the following properties:

P1: If a and b are two events in the same process, and $a \rightarrow b$, then we demand that $C(a) < C(b)$.

P2: If a corresponds to sending a message m, and b to the receipt of that message, then also $C(a) < C(b)$.

Lamport's Logical Clocks

Problem:

How to attach a timestamp to an event when there's no global clock => maintain a **consistent** set of logical clocks, one per process.

Solution:

Each process P_i maintains a **local** counter C_i and adjusts this counter according to the following rules: (Raynal and Singhal, 1996).

1. For any two successive events that take place within P_i, C_i is incremented by 1.
2. Each time a message m is sent by process P_i, the message receives a timestamp $t_s(m) = C_i$.
3. Whenever a message m is received by a process P_j, P_j adjusts its local counter C_j to max{C_j, ts(m)}; then executes step 1 before passing m to the application.

   Property **P1** is satisfied by (1);

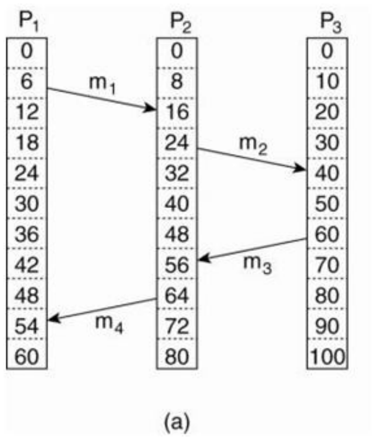
   Property **P2** by (2) and (3).

it can still occur that two events happen at the same time. Avoid this by breaking ties through process IDs.

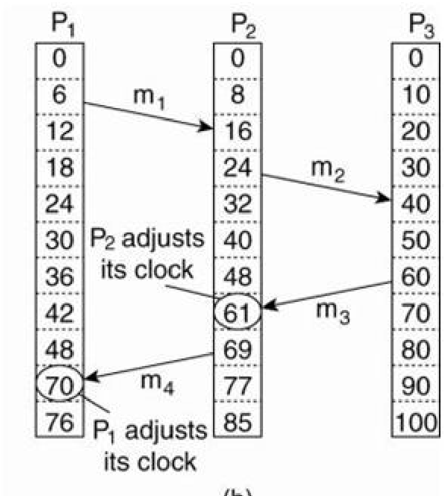
Example

(a) Three processes, each with its own clock. The clocks run at different rates.

(b) Lamport's algorithm corrects the clocks.



(a)

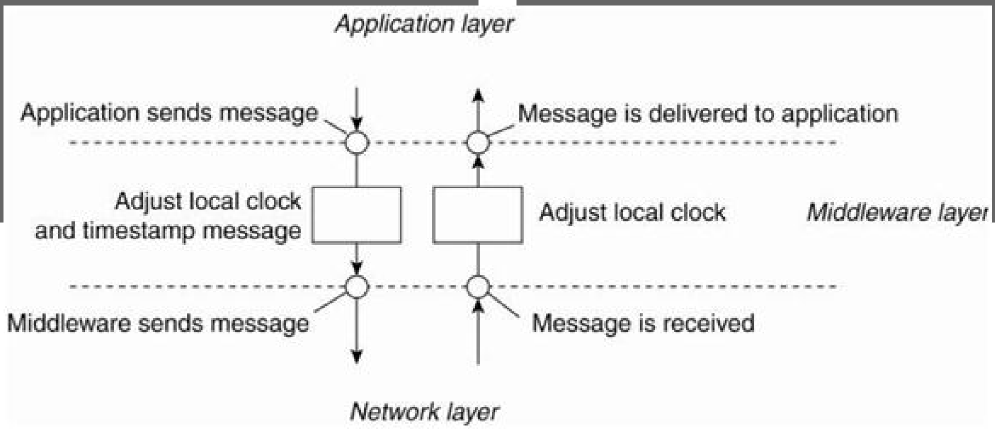


(b)

- The processes run on different machines, each with its own clock, running at its own speed.
- When the clock has ticked 6 times in process P1, it has ticked 8 times in process P2 and 10 times in process P3.
- Each clock runs at a constant rate, but the rates are different due to differences in the crystals.
- At time 6, process P1 sends message m1 to process P2, the clock in process P2 reads 16 when it arrives.
- If the message carries the starting time, 6, in it, process P2 will conclude that it took 10 ticks to make the journey.
- m3 leaves process P3 at 60 and arrives at P2 at 56.
- m4 from P2 to P1 leaves at 64 and arrives at 54.
- These values are clearly impossible. It is this situation that must be prevented.
- Since m3 left at 60, it must arrive at 61 or later.
  - Each message carries the sending time according to the sender's clock.
  - When a 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 the sending time.

Note:

Adjustments take place in the middleware layer:



2.3.3 Mutual Exclusion

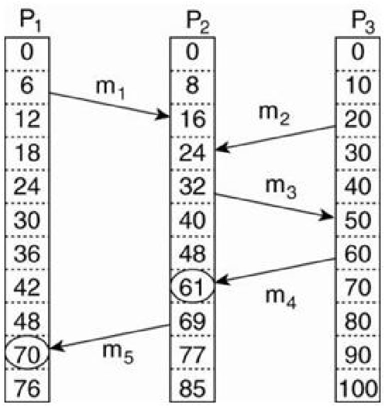
Q23. Write about Vector Clocks.

Ans :

Vector Clocks

Lamport's clocks do not guarantee that if C(a) < C(b) that **a causally preceded b**:

Concurrent Message Transmission using Logical Clocks



Observation

- Event a: m1 is received at T = 16.
- Event b: m2 is sent at T = 20.
- We **cannot** conclude that a causally precedes b.

Solution: Vector clocks

- 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'stimestamp.
- When a process P_j receives a message m from P_i with vector timestamp ts(m), it

  (1) updates each $VC_j[k]$ to $\max\{VC_j[k], ts(m)[k]\}$

  (2) increments $VC_j[j]$ by 1.

Enforcing Causal Communication

Causally Ordered Multicasting

We can now ensure that a message is delivered only if all causally preceding messages have already been delivered.

Pi increments $VC_i[i]$ only when sending a message, and P_j only adjusts VC_j when receiving a message (i.e., does not increment $VC_j[j]$.

Pj postpones delivery of m until:

- $ts(m)[i] = VC_j[i] + 1$.

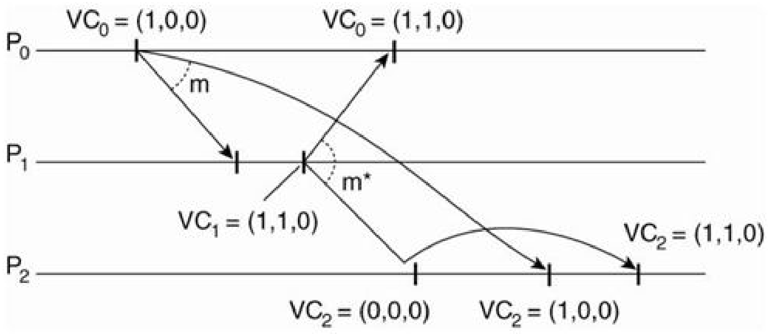
  Means that m is the next message that P_j was expecting from process P_i

- $ts(m)[k] \leq VC_j[k]$ for $k \neq j$

  Means that P_j has seen all the messages that have been seen by P_i when it sent message m.

Example

Enforcing causal communication.



- Three processes P0, P1, and P2.
  - At local time (1,0,0), P1 sends a message m to the other two processes.
  - After its receipt by P1, P1 sends m*, which arrives at P2 sooner than m.
  - At that point, the delivery of m* is delayed by P2 until m has been received and delivered to P2's application layer.

Q24. Explain about mutual exclusion.

Ans :

Mutual Exclusion

A number of processes in a distributed system want exclusive access to some resource.

- Via a **centralized server**
- **Completely decentralized**, using a peer-to-peer system.
- **Completely distributed**, with no topology imposed.
- Completely distributed along a **(logical) ring**.

Mutual Exclusion is very in simple in centralised systems

Distributed Mutual Exclusion Algorithms

1. Token-based solutions: mutual exclusion is achieved by passing a special message between the processes, known as a token.

   Only one token available and who ever has that token is allowed to access the shared resource.

   When finished, the token is passed on to a next process.

   Properties:

   i) Ensure that every process will get a chance at accessing the resource.

   ii) Deadlocks are avoided.

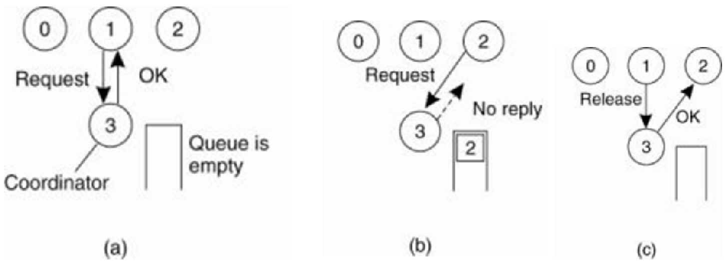
   iii) Drawback - token loss leads to a complex restart of procedure

2. Permission-based approach: a process wanting to access the resource first requires the permission of other processes.

Centralized Algorithm

Simulate how it is done in a one-processor system

- One process is elected as the coordinator.
- Whenever a process wants to access a shared resource, it sends a request message to the coordinator stating which resource it wants to access and asking for permission.
- If no other process is currently accessing that resource, the coordinator sends back a reply granting permission.
- When the reply arrives, the requesting process can proceed.
  (a) Process 1 asks the coordinator for permission to access a shared resource. Permission is granted.
  (b) Process 2 then asks permission to access the same resource. The coordinator does not reply.
  (c) When process 1 releases the resource, it tells the coordinator, which then replies to 2.



- The coordinator is a single point of failure, so if it crashes, the entire system may go down.
- In a large system, a single coordinator can become a performance bottleneck.

Deterministic distributed Mutual Exclusion Algorithm

- Requires that there be a total ordering of all events in the system. For any pair of events - it must be unambiguous which one actually happened first.
- When a process wants to access a shared resource, it builds a message containing the name of the resource, its process number, and the current (logical) time.
- It then sends the message to all other processes, conceptually including itself.
- When a process receives a request message from another process, the action it takes depends on its own state with respect to the resource named in the message.

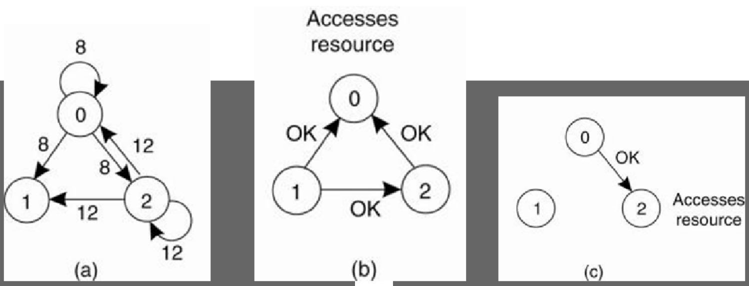
  Three different cases have to be clearly distinguished:

1. If the receiver is not accessing the resource and does not want to access it, it sends back an OK message to the sender.
2. If the receiver already has access to the resource, it simply does not reply. Instead, it queues the request.
3. If the receiver wants to access the resource as well but has not yet done so, it compares the timestamp of the incoming message with the one contained in the message that it has sent everyone.

   The lowest one wins.

   a) If the incoming message has a lower timestamp, the receiver sends back an OK message.
   b) If its own message has a lower timestamp, the receiver queues the incoming request and sends nothing.

- After sending out requests asking permission, a process waits until everyone else has given permission – then proceeds.
- When finished, it sends OK messages to all processes on its queue and deletes them all from the queue.
- If there is no conflict, it clearly works.
- Suppose that two processes try to simultaneously access the resource:
  - (a) Two processes want to access a shared resource at the same moment.
  - (b) Process 0 has the lowest timestamp, so it wins.
  - (c) When process 0 is done, it sends an OK also, so 2 can now go ahead.



1. Process 0 sends everyone a request with timestamp 8, while at the same time, process 2 sends everyone a request with timestamp 12.
2. Process 1 is not interested in the resource, so it sends OK to both senders.
3. Processes 0 and 2 both see the conflict and compare timestamps.
4. Process 2 sees that it has lost, so it grants permission to 0 by sending OK. Process 0 now queues the request from 2 for later processing and access the resource, as shown in Fig. 6 figure(b).
5. When it is finished, it removes the request from 2 from its queue and sends an OK message to process 2, allowing the latter to go ahead, as shown in Figure(c).

The algorithm works because in the case of a conflict, the lowest timestamp wins and everyone agrees on the ordering of the timestamps.

Advantages

- Mutual exclusion is guaranteed without deadlock or starvation.
- The number of messages required per entry is now 2(n - 1), where the total number of processes in the system is n.
- No single point of failure exists.

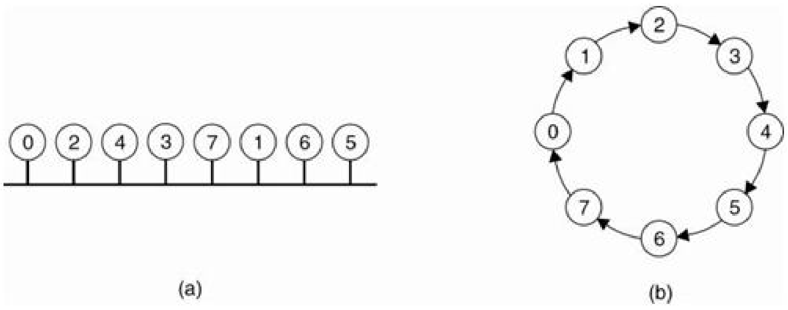
Q25. Write about Token Ring Algorithm for mutual Exclusion.

Ans :

Mutual Exclusion: Token Ring Algorithm

Organize processes in a logical ring, and let a token be passed between them.

1. The one that holds the token is allowed to enter the critical region (if it wants to)
   - a) An unordered group of processes on a network.
   - b) A logical ring constructed in software.



2. When the ring is initialized, process 0 is given a token.
3. The token circulates around the ring.
4. It is passed from process k to process k+1 (modulo the ring size) in point-to-point messages.
5. When a process acquires the token from its neighbor, it checks to see if it needs to access the shared resource.
   a) If so, the process goes ahead, does all the work it needs to, and releases the resources.
   b) After it has finished, it passes the token along the ring.
   c) It is not permitted to immediately enter the resource again using the same token.
6. If a process is handed the token by its neighbor and is not interested in the resource, it just passes the token along.

   As a consequence, when no processes need the resource, the token just circulates at high speed around the ring.

Advantages

1. Only one process has the token at any instant, so only one process can actually get to the resource.
2. Since the token circulates among the processes in a well-defined order, starvation cannot occur.
3. Once a process decides it wants to have access to the resource, at worst it will have to wait for every other process to use the resource.

Disadvantages

1. If the token is ever lost, it must be regenerated
2. Dead processes:
   a) If a process receiving the token must acknowledge receipt, a dead process will be detected when its neighbor tries to give it the token and fails.
   b) At that point the dead process can be removed from the group, and the token holder can throw the token over the head of the dead process to the next member down the line, or the one after that, if necessary.

2.3.4 Global Positioning of Nodes

Q26. Write about Geometric Overlay Network

Ans :

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.

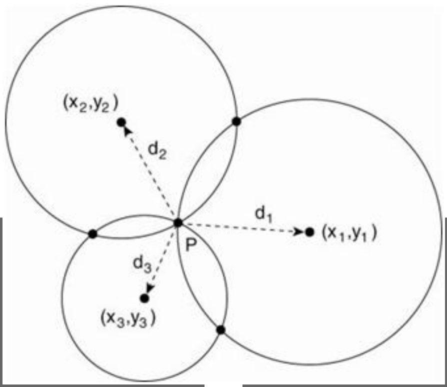
Example: Distance corresponds to internode latency.

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. construct a **geometric overlay network**, in which the distance d(P,Q) reflects the actual latency between P and Q.

A node P needs k + 1 **landmarks** to compute its own position in a d-dimensional space.

Consider two-dimensional Case

Computing a node's position in a two-dimensional space.



Just as 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.

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

➢ d_i corresponds to measuring the latency between P and the node at (x_i,y_i).

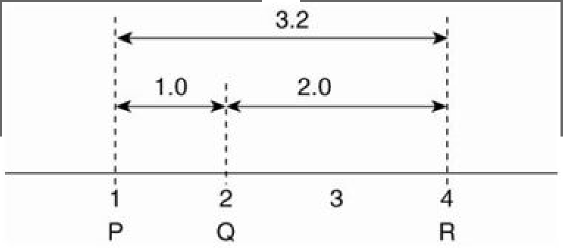
➢ Latency estimated as half the round-trip dela.

Problems:

Measured latencies to landmarks fluctuate

Computed distances will not even be consistent:

Inconsistent distance measurements in a one-dimensional space.



Solution :

Let the L landmarks measure their pair wise latencies $d(b_i, b_j)$ and let each node P minimize

$$\sum_{i=1}^{L}\left[\frac{d(b_i,P) - \hat{d}(b_i,P)}{d(b_i,P)}\right]^2$$

where $\hat{d}(b_i,b_j)$ denotes the distance to landmark b_i given a computed coordinate for P.

With well-chosen landmarks, m can be as small as 6 or 7, with $\hat{d}(P,Q)$ being no more than a factor 2 different from the actual latency d(P, Q) for arbitrary nodes P and Q.

2.3.5 Election Algorithms

Q27. Explain election algorithms.

Ans :

An algorithm requires that some process acts as a coordinator. In many systems the coordinator is chosen by hand (e.g. file servers). This leads to centralized solutions $\Rightarrow$ single point of failure.

Election by Bullying

Each process has an associated priority (weight). The process with the highest priority should always be elected as the coordinator.

To find the heaviest process

- Any process can just start an election by sending an election message to all other processes (assuming you don't know the weights of the others).
- If a process P_{heavy} receives an election message from a lighter process Plight, it sends a take-over message to P_{light}. P_{light} is out of the race.
- If a process doesn't get a take-over message back, it wins, and sends a victory message to all other processes.

The Bully Election Algorithm

- The group consists of eight processes, numbered from 0 to 7.
- Previously process 7 was the coordinator, but it has just crashed.

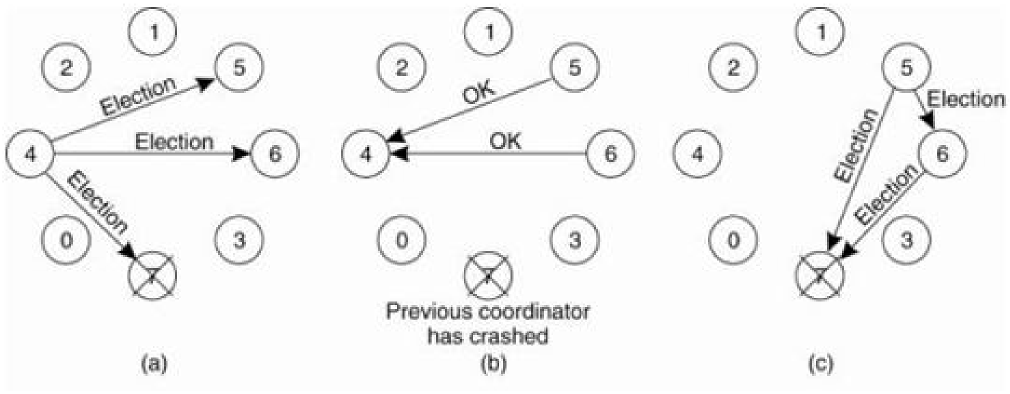
  (a) Process 4 is the first one to notice this, so it sends ELECTION messages to all the processes higher than it, namely 5, 6, and 7,

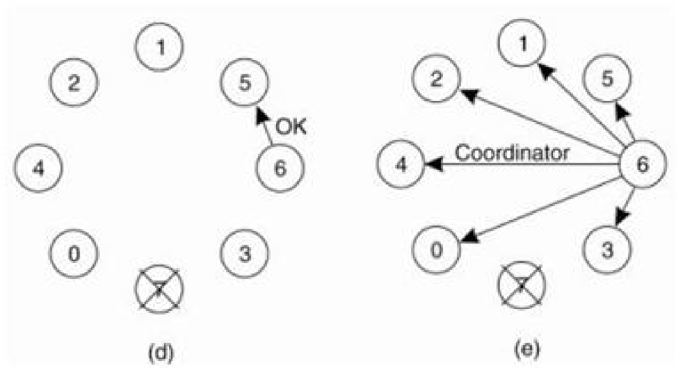
  (b) Processes 5 and 6 respond, telling 4 to stop.

  (c) Now 5 and 6 each hold an election.

  (d) Process 6 tells 5 to stop.

  (e) Process 6 wins and tells everyone.





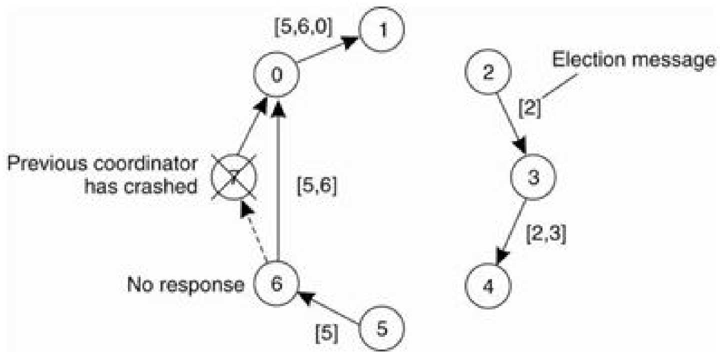
- If a process that was previously down comes back up, it holds an election.
- If it happens to be the highest-numbered process currently running, it will win the election and take over the coordinator’s job.
- Thus the biggest guy in town always wins, hence the name “bully algorithm.”

Election in a Ring

Process priority is obtained by organizing processes into a (logical) ring. Process with the highest priority should be elected as coordinator. Any process can start an election by sending an election message to its successor. If a successor is down, the message is passed on to the next successor. If a message is passed on, the sender adds itself to the list. When it gets back to the initiator, everyone had a chance to make its presence known. The initiator sends a coordinator message around the ring containing a list of all living processes. The one with the highest priority is elected as coordinator.

Example

- What happens if two processes, 2 and 5, discover simultaneously that the previous coordinator, process 7, has crashed.
- Each builds an ELECTION message and each starts circulating its message, independent of the other.
- Both messages will complete a circuit, and both 2 and 5 will convert them into COORDINATOR messages, with exactly the same members and in the same order.
- When both have completed a second circuit, both will be removed.



UNIT III

Consistency and Replication:introduction, data-centric consistency models, client-centric consistency models, replica management, consistency protocols. **Fault Tolerance:** Introduction, process resilience, reliable clientserver communication, reliable group communication, distributed commit, recovery. **Security:** introduction, secure channels, access control, security management.

3.1 Consistency and Replication

3.1.1 Introduction

Q1. Write about the reasons for replication

Ans :

Reasons for Replication

Performance and Scalability

To keep replicas consistent, we generally need to ensure that all **conflicting** operations are done in the same order everywhere. From the world of transactions:

- **Read-write conflict**: a read operation and a write operation act concurrently
- **Write-write conflict**: two concurrent write operations

Guaranteeing global ordering on conflicting operations may be a costly operation, downgrading scalability.

- weaken consistency requirements so that hopefully global synchronization can be avoided

3.1.2 Data-centric Consistency Models

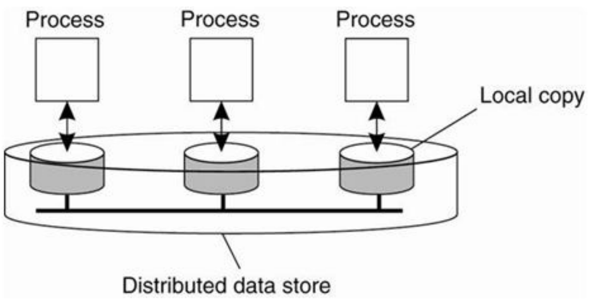
Q2. What are data centric consistency models

Ans :

A contract between a (distributed) data store and processes, in which the data store specifies precisely what the results of read and write operations are in the presence of concurrency.

A data store is a distributed collection of storages accessible to clients:

The general organization of a logical data store, physically distributed and replicated across multiple processes.

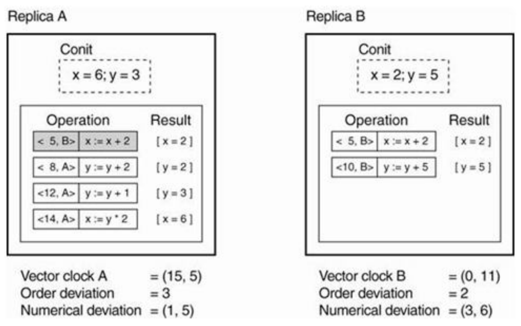


Continuous Consistency

We can actually talk a about a degree of consistency:

- replicas may differ in their numerical value
- replicas may differ in their relative staleness
- there may differences with respect to (number and order) of performed update operations conistency unit specifies the data unit over which consistency is to be measured.
- e.g. stock record, weather report, etc.

Example: numerical and ordering deviations



It contains the variables x and y:

- Each replica maintains a **vector clock**
- B sends A operation [h5, Bi: x := x + 2];

  A has made this operation **permanent** (cannot be rolled back)
- A has three **pending** operations) order deviation = 3
- A has missed **one** operation from B, yielding a max diff of 5 units) (1,5)

Strict Consistency

Any read on a data item 'x' returns a value corresponding to the result of the most recent write on 'x' (regardless of where the write occurred).

With Strict Consistency, all writes are instantaneously visible to all processes and absolute global time order is maintained throughout the distributed system. This is the consistency model "Holy Grail" – not at all easy in the real world, and all but impossible within a DS.

Example

Begin with:

Wi(x)a –write by process Pi to data item x with the value a

Ri(x)b - read by process Pi from data item x returning the value b

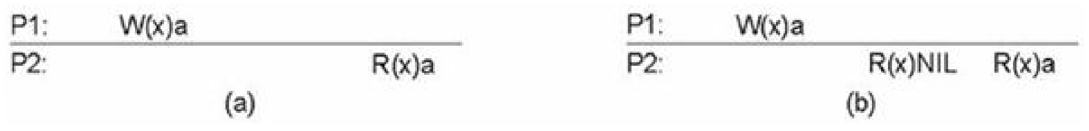
Assume

- Time axis is drawn horizontally, with time increasing from left to right
- Each data item is initially NIL.

  P1 does a write to a data item x, modifying its value to a.
- Operation W1(x)a is first performed on a copy of the data store that is local to P1, and is then propagated to the other local copies.

  P2 later reads the value NIL, and some time after that reads a

  It took some time to propagate the update of x to P2, which is perfectly acceptable.



Behavior of two processes, operating on the same data item:

(a) A strictly consistent data-store.

(b) A data-store that is not strictly consistent.

Sequential Consistency

It is a weaker consistency model, which represents a relaxation of the rules. It is also must easier (possible) to implement.

Sequential Consistency

The result of any execution is the same as if the (read and write) operations by all proceses on the data-store were executed in the same sequential order and the operations of each individual process appear in this sequence in the order specified by its program.

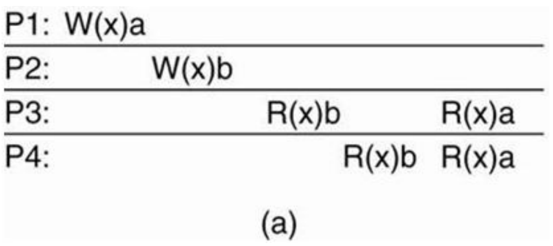
Example

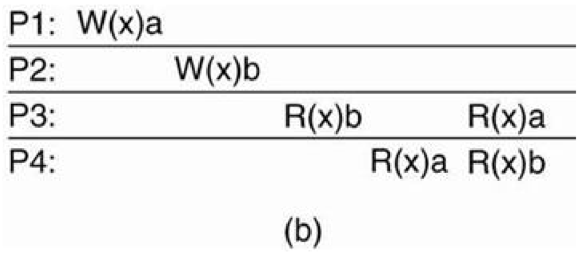
Time independent process

Four processes operating on the same data item x.

Figure (a)	Figure(b)
• Process P1 first performs W(x)a to x. • Later (in absolute time), process P2 performs a write operation, by setting the value of x to b. • Both P3 and P4 first read value b, and later value a. • Write operation of process P2 appears to have taken place before that of P1.	• Violates sequential consistency - not all processes see the same interleaving of write operations. • To process P3, it appears as if the data item has first been changed to b, and later to a. • BUT, P4 will conclude that the final value is b.

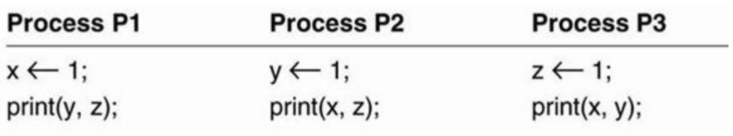
(a) A sequentially consistent data store. (b) A data store that is not sequentially consistent.





Example

- Three concurrently-executing processes P1, P2, and P3 (Dubois et al., 1988).
- Three integer variables x, y, and z, which stored in a (possibly distributed) shared sequentially consistent data store.
- Assume that each variable is initialized to 0.
- An assignment corresponds to a write operation, whereas a print statement corresponds to a simultaneous read operation of its two arguments.
- All statements are assumed to be indivisible.

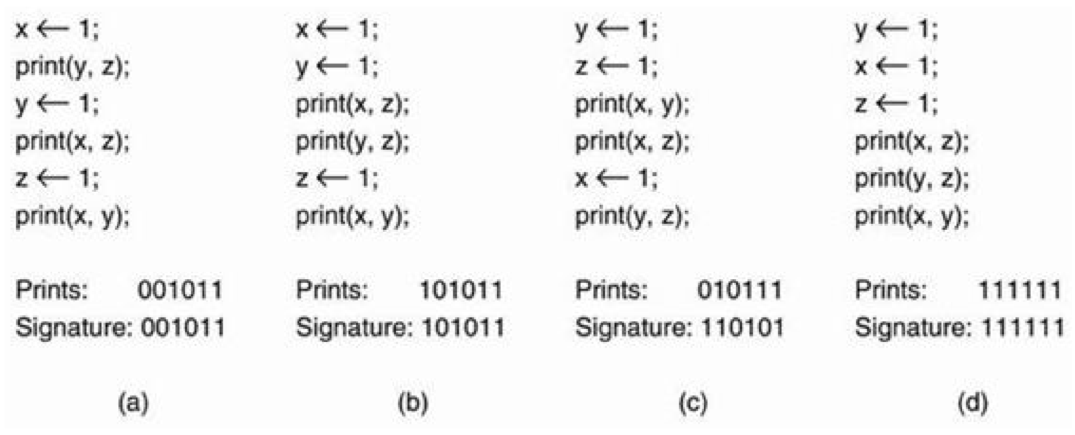


Process P1	Process P2	Process P3
$x \leftarrow 1$;	$y \leftarrow 1$;	$z \leftarrow 1$;
print(y, z);	print(x, z);	print(x, y);

- Various interleaved execution sequences are possible.
- With six independent statements, there are potentially 720 (6!) possible execution sequences
  - Consider the 120 (5!) sequences that begin with $x \leftarrow 1$.
  - Half of these have print (x, z) before $y \leftarrow 1$ and thus violate program order.
  - Half have print (x, y) before $z \leftarrow 1$ and also violate program order.

- o Only 1/4 of the 120 sequences, or 30, are valid.
- o Another 30 valid sequences are possible starting with $y \leftarrow 1$
- o Another 30 can begin with $z \leftarrow 1$, for a total of 90 valid execution sequences.

Four valid execution sequences for the processes. The vertical axis is time.



x ← 1;	x ← 1;	y ← 1;	y ← 1;
print(y, z);	y ← 1;	z ← 1;	x ← 1;
y ← 1;	print(x, z);	print(x, y);	z ← 1;
print(x, z);	print(y, z);	print(x, z);	print(x, z);
z ← 1;	z ← 1;	x ← 1;	print(y, z);
print(x, y);	print(x, y);	print(y, z);	print(x, y);
Prints: 001011	Prints: 101011	Prints: 010111	Prints: 111111
Signature: 001011	Signature: 101011	Signature: 110101	Signature: 111111
(a)	(b)	(c)	(d)

Figure (a) - The three processes are in order - P1, P2, P3.

- o Each of the three processes prints two variables.
- o Since the only values each variable can take on are the initial value (0), or the assigned value (1), each process produces a 2-bit string.
- o The numbers after Prints are the actual outputs that appear on the output device.

➢ Output concatenation of P1, P2, and P3 in sequence produces a 6-bit string that characterizes a particular interleaving of statements.

- o This is the string listed as the Signature.

Examples of possible output

➢ 000000 is not permitted - implies that the Print statements ran before the assignment statements, violating the requirement that statements are executed in program order.

➢ 001001 – is not permitted

- o First two bits 00 - y and z were both 0 when P1 did its printing.
  - This situation occurs only when P1 executes both statements before P2 or P3 start.
- o Second two bits 10 - P2 must run after P1 has started but before P3 has started.
- o Third two bits, 01- P3 must complete before P1 starts, but P1 execute first.

Causal Consistency

➢ Writes that are potentially causally related must be seen by all processes in the same order.

➢ Concurrent writes (i.e. writes that are NOT causally related) may be seen in a different order by different processes.

Example

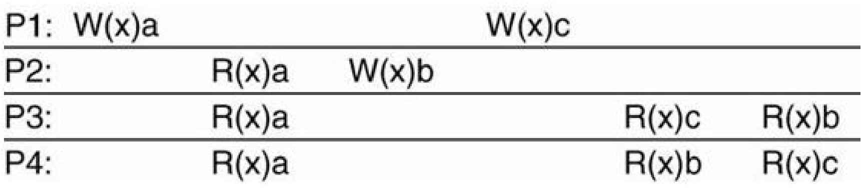
➢ Interaction through a distributed shared database.

➢ Process P1 writes data item x.

- Then P2 reads x and writes y.
- Reading of x and writing of y are potentially causally related because the computation of y may have depended on the value of x as read by P2 (i.e., the value written by P1).
- Conversly, if two processes spontaneously and simultaneously write two different data items, these are not causally related.
- Operations that are not causally related are said to be concurrent.
- For a data store to be considered causally consistent, it is necessary that the store obeys the following condition:
  - o Writes that are potentially causally related must be seen by all processes in the same order.
  - o Concurrent writes may be seen in a different order on different machines.

Example 1: Causal Consistency

This sequence is allowed with a causally-consistent store, but not with a sequentially consistent store.

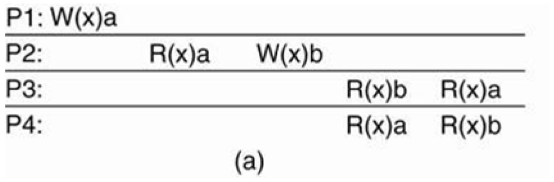


P1:	W(x)a			W(x)c		
P2:		R(x)a	W(x)b			
P3:		R(x)a			R(x)c	R(x)b
P4:		R(x)a			R(x)b	R(x)c

The writes W2(x)b and W1(x)c are concurrent, so it is not required that all processes see them in the same order.

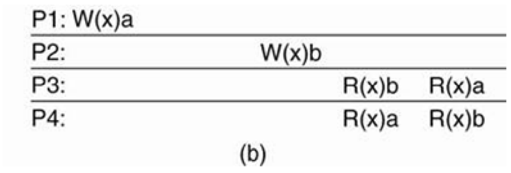
Example 2: Causal Consistency

• W2(x)b potentially depending onW1(x)a because b may result from a computation involving the value read by R2(x)a. • The two writes are causally related, so all processes must see them in the same order. • It is incorrect.	• Read has been removed, so W1(x)a and W2(x)b are now concurrent writes. • A causally-consistent store does not require concurrent writes to be globally ordered, • It is correct. • Note: situation that would not be acceptable for a sequentially consistent store.
(a) A violation of a causally-consistent store.	(b) A correct sequence of events in a causally-consistent store.



P1: W(x)a				
P2:	R(x)a	W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)a	R(x)b

(a)



P1: W(x)a			
P2:	W(x)b		
P3:		R(x)b	R(x)a
P4:		R(x)a	R(x)b

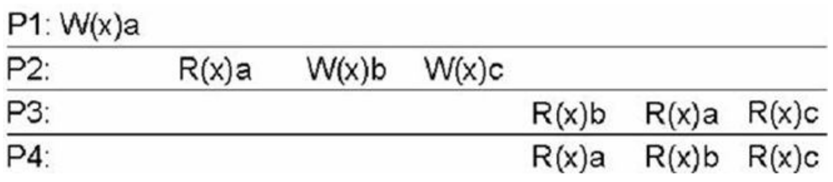
(b)

FIFO Consistency

Writes done by a single process are seen by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes.

- Also called “PRAM Consistency” – Pipelined RAM.
- Easy to implement -There are no guarantees about the order in which different processes see writes – except that two or more writes from a single process must be seen in order.

Example



P1:	W(x)a						
P2:		R(x)a	W(x)b	W(x)c			
P3:					R(x)b	R(x)a	R(x)c
P4:					R(x)a	R(x)b	R(x)c

- A valid sequence of FIFO consistency events.
- Note that none of the consistency models given so far would allow this sequence of events.

Weak Consistency

- Not all applications need to see all writes, let alone seeing them in the same order.
- Leads to Weak Consistency .
- This model introduces the notion of a synchronization variable”, which is used update all copies of the data-store.

Properties

1. Accesses to synchronization variables associated with a data-store are sequentially consistent.
2. No operation on a synchronization variable is allowed to be performed until all previous writes have been completed everywhere.
3. No read or write operation on data items are allowed to be performed until all previous operations to synchronization variables have been performed.

By doing a sync

- a process can force the just written value out to all the other replicas.
- a process can be sure it’s getting the most recently written value before it reads.

The weak consistency models enforce consistency on a group of operations, as opposed to individual reads and writes.

Grouping Operations

- Accesses to **synchronization variables** are sequentially consistent.
- No access to a synchronization variable is allowed to be performed until all previous writes have completed everywhere.
- No data access is allowed to be performed until all previous accesses to synchronization variables have been performed.

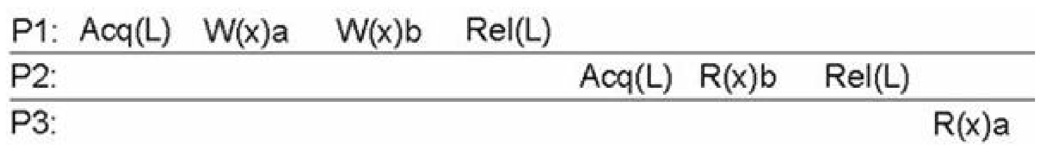
When a process enters its critical section it should acquire the relevant synchronization variables, and likewise when it leaves the critical section, it releases these variables.

Release Consistency

- It is possible to implement efficiencies if the data-store is able to determine whether the sync is a read or write.

- Two sync variables can be used, acquire and release, and their use leads to the Release Consistency model.
- When a process does an acquire, the data-store will ensure that all the local copies of the protected data are brought up to date to be consistent with the remote ones if needs be.
- When a release is done, protected data that have been changed are propagated out to the local copies of the data-store.

Example



A valid event sequence for release consistency.

- Process P3 has not performed an acquire, so there are no guarantees that the read of 'x' is consistent.
- The data-store is simply not obligated to provide the correct answer.
- P2 does perform an acquire, so its read of 'x' is consistent.

Release Consistency Rules

A distributed data-store is "Release Consistent" if it obeys the following rules:

1. Before a read or write operation on shared data is performed, all previous acquires done by the process must have completed successfully.
2. Before a release is allowed to be performed, all previous reads and writes by the process must have completed.
3. Accesses to synchronization variables are FIFO consistent (sequential consistency is not required).

Entry consistency

- Acquire and release are still used, and the data-store meets the following conditions:
- An acquire access of a synchronization variable is not allowed to perform with respect to a process until all updates to the guarded shared data have been performed with respect to that process.
- Before an exclusive mode access to a synchronization variable by a process is allowed to perform with respect to that process, no other process may hold the synchronization variable, not even in nonexclusive mode.
- After an exclusive mode access to a synchronization variable has been performed, any other process's next nonexclusive mode access to that synchronization variable may not be performed until it has performed with respect to that variable's owner.

So:

- At an acquire, all remote changes to guarded data must be brought up to date.
- Before a write to a data item, a process must ensure that no other process is trying to write at the same time.

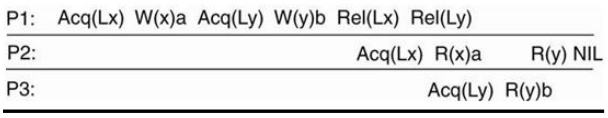
  Locks are associates with individual data items, as opposed to the entire data-store.
- a **lock** is a synchronization mechanism for enforcing limits on access to a resource in an environment where there are many threads of execution.

Example:

- P1 does an acquire for x, changes x once, after which it also does an acquire for y.
- Process P2 does an acquire for x but not for y, so that it will read value a for x, but may read NIL for y.
- Because process P3 first does an acquire for y, it will read the value b when y is released by P1.

Note: P2's read on 'y' returns NIL as no locks have been requested.

A valid event sequence for entry consistency.


P1: Acq(Lx) W(x)a Acq(Ly) W(y)b Rel(Lx) Rel(Ly)
P2: Acq(Lx) R(x)a R(y) NIL
P3: Acq(Ly) R(y)b

3.1.3 Client-centric Consistency Models

Q3. Write about Client-Centric Consistency Models?

Ans :

Client-Centric Consistency Models

- Above consistency models - maintaining a consistent (globally accessible) data-store in the presence of concurrent read/write operations.
- Another class of distributed datastore - characterized by the lack of simultaneous updates.

Here, the emphasis is more on maintaining a consistent view of things for the individual client process that is currently operating on the data-store.

Eventual Consistency

Special class of distributed data stores:

- Lack of simultaneous updates
- When updates occur they can easily be resolved.
- Most operations involve reading data.

These 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.

Example: Consistency for Mobile Users

- Consider a distributed database to which you have access through your notebook.
- Assume your notebook acts as a front end to the database.
  - o At location A you access the database doing reads and updates.
  - o At location B you continue your work, but unless you access the same server as the one at location A, you may detect inconsistencies:
- your updates at A may not have yet been propagated to B
- you may be reading newer entries than the ones available at A
- your updates at B may eventually conflict with those at A

Client-centric consistency models originate from the work on Bayou. Bayou is a database system developed for mobile computing, where it is assumed that network connectivity is unreliable and subject to various performance problems. Wireless networks and networks that span large areas, such as the Internet, fall into this category.

- o Bayou distinguishes four different consistency models:

1. monotonic reads
2. monotonic writes
3. read your writes
4. writes follow reads

Notation

- $x_i[t]$ denotes the version of data item x at local copy L_i at time t.
- WS $x_i[t]$ is the set of write operations at L_i that lead to version x_i of x (at time t);

- If operations in WS $x_i[t1]$ have also been performed at local copy L_j at a later time t2, we write WS ($x_i[t1]$, $x_j[t2]$).
- If the ordering of operations or the timing is clear from the context, the time index will be omitted.

Monotonic Reads

If a process reads the value of a data item x, any successive read operation on x by that process will always return that same or a more recent value.

- Monotonic-read consistency guarantees that if a process has seen a value of x at time t, it will never see an older version of x at a later time.

  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.

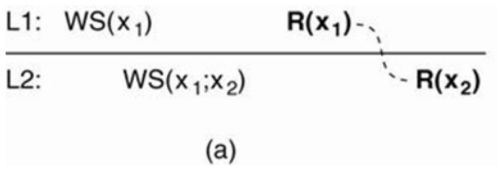
  Example: Reading (not modifying) incoming mail while you are on the move.

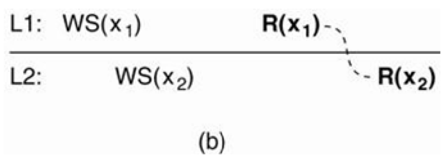
- Each time you connect to a different e-mail server, that server fetches (at least) all the updates from the server you previously visited.

Example

- The read operations performed by a single process P at two different local copies of the same data store.
- Vertical axis - two different local copies of the data store are shown - L1 and L2
- Time is shown along the horizontal axis
- Operations carried out by a single process P in boldface are connected by a dashed line representing the order in which they are carried out.

(a) A monotonic-read consistent data store. (b) A data store that does not provide monotonic reads.





(a)	(b)
• Process P first performs a read operation on x at L1, returning the value of x1 (at that time). o This value results from the write operations in WS (x1) performed at L1. • Later, P performs a read operation on x at L2, shown as R (x2). • To guarantee monotonic-read consistency, all operations in WS (x1) should have been propagated to L2 before the second read operation takes place.	• Situation in which monotonic-read consistency is not guaranteed. • After process P has read x1 at L1, it later performs the operation R (x2) at L2 . • But, only the write operations in WS (x2) have been performed at L2 . • No guarantees are given that this set also contains all operations contained in WS (x1).

Monotonic Writes

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.

Hence: 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.

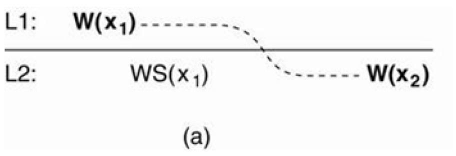
Example: Updating a program at server S2, and ensuring that all components on which compilation and linking depends, are also placed at S2.

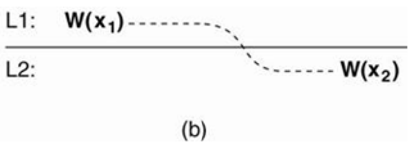
Example

The write operations performed by a single process P at two different local copies of the same data store.

(a) A monotonic-write consistent data store.

(b) A data store that does not provide monotonic-write consistency.





(a)	(b)
o Process P performs a write operation on x at local copy L1, presented as the operation W(x1). o Later, P performs another write operation on x, but this time at L2, shown as W (x2). o To ensure monotonic-write consistency, the previous write operation at L1 must have been propagated to L2. o This explains operation W (x1) at L2, and why it takes place before W (x2).	o Situation in which monotonic-write consistency is not guaranteed. o Missing is the propagation of W(x1) to copy L2. o No guarantees can be given that the copy of x on which the second write is being performed has the same or more recent value at the time W(x1) completed at L1.

Read Your Writes

A client-centric consistency model that is closely related to monotonic reads is as follows. A data store is said to provide read-your-writes consistency, if the following condition holds:

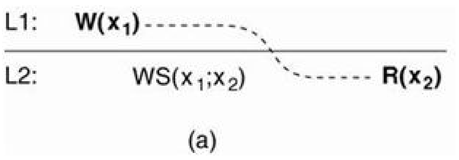
The effect of a write operation by a process on data item x will always be seen by a successive read operation on x by the same process.

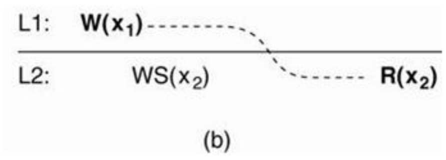
Hence : A write operation is always completed before a successive read operation by the same process, no matter where that read operation takes place.

Example

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

(b) A data store that does not.





(a)	(b)
o Process P performed a write operation W(x1) and later a read operation at a different local copy. o Read-your-writes consistency guarantees that the effects of the write operation can be seen by the succeeding read operation. o This is expressed by WS (x1;x2), which states that W (x1) is part of WS (x2).	o W (x1) has been left out of WS (x2), meaning that the effects of the previous write operation by process P have not been propagated to L2.

Writes Follow Reads

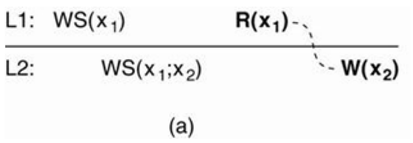
A data store is said to provide writes-follow-reads consistency, if the following holds:

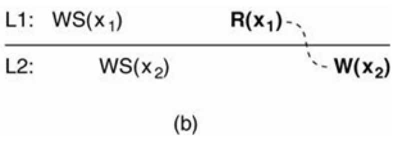
A write operation by a process on a data item x following a previous read operation on x by the same process is guaranteed to take place on the same or a more recent value of x that was read.

Hence: any successive write operation by a process on a data item x will be performed on a copy of x that is up to date with the value most recently read by that process.

Example

(a) A writes-follow-reads consistent data store. (b) A data store that does not provide writes-follow-reads consistency.





(a)	(b)
o A process reads x at local copy L1. o The write operations that led to the value just read, also appear in the write set at L2, where the same process later performs a write operation. o (Note that other processes at L2 see those write operations as well.)	o No guarantees are given that the operation performed at L2, o They are performed on a copy that is consistent with the one just read at L1.

3.1.4 Replica Management

Q4. Explain about Replica Management in Distributed Systems.

Ans :

The issue in replica management is to decide where, when, and by whom replicas should be placedand Which mechanisms to use for keeping the replicas consistent.

And replica servers find the placement issues inorder to place the content.

Placement Problem

- Placing replica servers
  - o Replica-server placement is concerned with finding the best locations to place a server that can host (part of) a data store.
- Placing content.
  - o Content placement deals with finding the best servers for placing content.

Replica-Server Placement

We need to figure out which is the best K places are out of N possible locations.In order to find the place for replica server. It follows the following:

1. Select best location out of N - k for which the average distance to clients is minimal. Then choose the next best server.
2. Select the k-th largest autonomous system and place a server at the best-connected host.
   - o An autonomous system (AS) can best be viewed as a network in which the nodes all run the same routing protocol and which is managed by a single organization.
3. Position nodes in a d-dimensional geometric space, where distance reflects latency. Identify the K regions with highest density and place a server in every one.

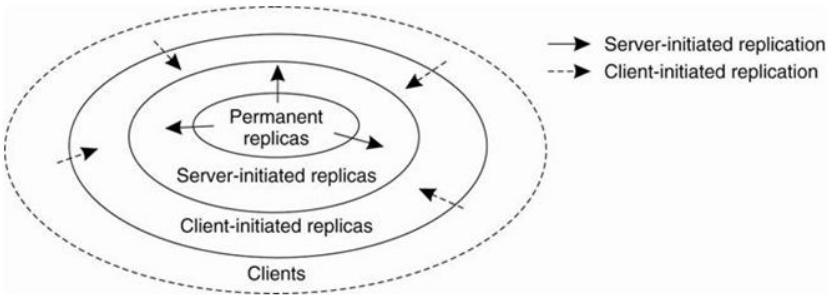
Content Replication and Placement

For Content Replication and placement, We considerobjects and **Distinguish different processes.**

A process is capable of hosting a replica of an object or data:

- **Permanent replicas:** Process/machine always having a replica
- **Server-initiated replica:** Process that can dynamically host a replica on request of another server in the data store
- **Client-initiated replica:** Process that can dynamically host a replica on request of a client (client cache)

The logical organization of different kinds of copies of a data store into three concentric rings.


Server-initiated replication
Client-initiated replication
Permanent replicas
Server-initiated replicas
Client-initiated replicas
Clients

Permanent Replicas

Examples:

- The initial set of replicas that constitute a distributed data store.
- The number of permanent replicas is small.
- The files that constitute a site are replicated across a limited number of servers at a single location.

- o Whenever a request comes in, it is forwarded to one of the servers, for instance, using a round-robin strategy.

➢ Alternatively, a database is distributed and possibly replicated across a number of geographically dispersed sites.

- o This architecture is generally deployed in federated databases..

Server-Initiated Replicas

Copies of a data store are created to enhance performance

Example

➢ Placement of content with replica servers in place

➢ Algorithm is designed to support Web pages for which reason it assumes that updates are relatively rare compared to read requests.

- o Algorithm considers:
  - Replication take used to reduce the load on a server.
  - Specific files on a server can be migrated or replicated to servers placed in the proximity of clients that issue many requests for those files.

Each server keeps track of:

- o access counts per file
- o where access requests originate

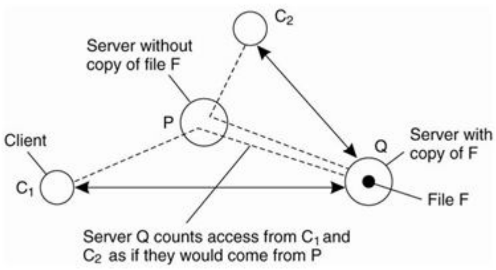
Given a client C, each server can determine which of the servers in the Web hosting service is closest to C. If client C1 and client C2 share the same "closest" server P, all access requests for file F at server Q from C1 and C2 are jointly registered at Q as a single access count cntQ(P,F).

Removed from S - when the number of requests for a file F at server S drops below a deletion threshold del (S,F)

➢ Replicate F – when replication threshold rep (S,F) is surpassed

➢ Migrate F – when the number of requests lies between the deletion and replication thresholds

Counting access requests from different clients.



C_2
Server without copy of file F
P
Client
C_1
Q
Server with copy of F
File F
Server Q counts access from C_1 and C_2 as if they would come from P

Client-Initiated Replicas

➢ Client-initiated replicas aka (client) caches.

➢ Cache is a local storage facility that is used by a client to temporarily store a copy of the data it has just requested.

➢ Managing cache is left to the client.

➢ Used to improve access times to data.

Approaches to Cache Placement

➢ **Traditional file systems:** data files are rarely shared at allrendering a shared cache useless.

➢ **LAN caches:** Machine shared by clients on the same local-area network.

➢ **WAN caches:** Place (cache) servers at specific points in a wide-area network and let a client locate the nearest server.

- o When the server is located, it can be requested to hold copies of the data the client was previously fetching from somewhere else.

3.1.5 Consistency Protocols

Q5. Explain about consistency protocols.

Ans :

A consistency protocol describes an implementation of a specific consistency model.

The most widely implemented models are:

1. Sequential Consistency

   Those in which operations can be grouped through locking or transactions

2. Weak Consistency (with sync variables).
3. Atomic Transactions

Sequential Consistency

Primary-Based Protocols

- ➢ Use 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:

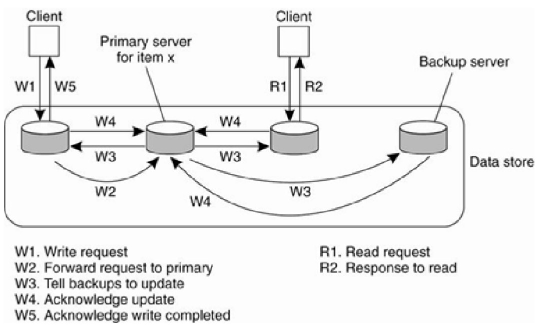
1. Remote-Write.
2. Local-Write.

Remote-Write Protocols

- ➢ AKA primary backup protocols (Budhiraja et al., 1993)
- ➢ 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:

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.



The Bad and Good of Primary-Backup

- ➢ Bad: Performance!
  - o All of those writes can take a long time (especially when a "blocking write protocol" is used).
  - o Using a non-blocking write protocol to handle the updates can lead to fault tolerant problems (which is our next topic).
- ➢ Good: 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.

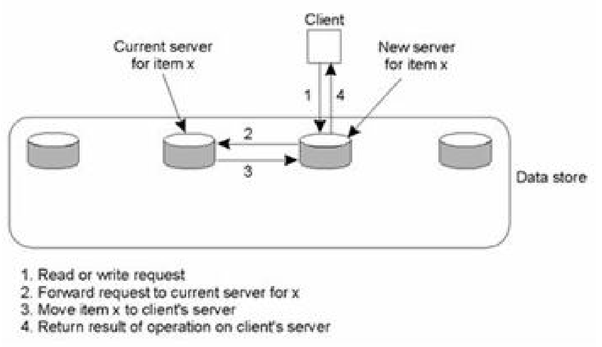
Local-Write Protocols

- ➢ AkA 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.
  - o 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:

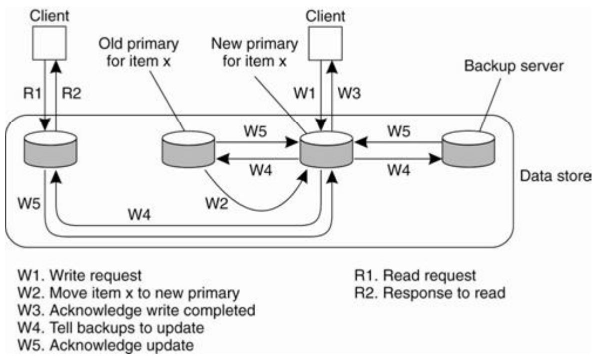
Primary-based local-write protocol in which a single copy is migrated between processes (prior to the read/write).


Client
Current server for item x
New server for item x
1
4
2
3
Data store
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

Local-Write Issues

Processes can spend more time actually locating a data item than using it!

Primary-backup protocol in which the primary migrates to the process wanting to perform an update.



Advantage

- Multiple, successive write operations can be carried out locally, while reading processes can still access their local copy.
- Can be achieved only if a nonblocking protocol is followed by which updates are propagated to the replicas after the primary has finished with locally performing the updates.

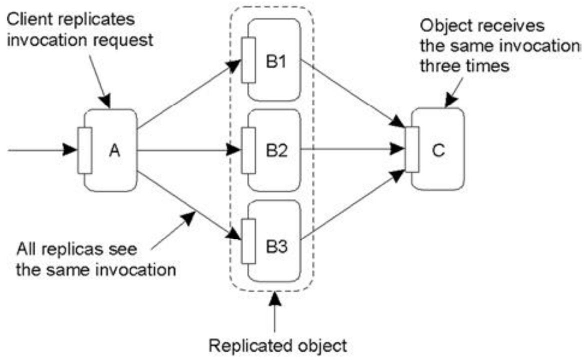
Replicated-Write Protocols

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

Active Replication

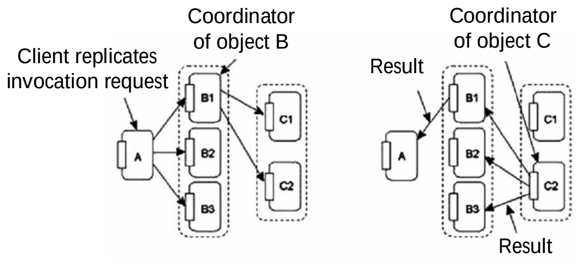
- A special process carries out the update operations at each replica.
- Lamport'stimsestamps can be used to achieve total ordering, but this does not scale well within Distributed Systems.
- An alternative/variation is to use a sequencer, which is a process that assigns a unique ID# to each update, which is then propagated to all replicas.

  Issue: replicated invocations


Client replicates invocation request
A
B1
B2
B3
C
Object receives the same invocation three times
All replicas see the same invocation
Replicated object

The problem of replicated invocations –

- 'B' is a replicated object (which itself calls 'C').
- When 'A' calls 'B', how do we ensure 'C' isn't invoked three times?



(a) (b)

(a) Using a coordinator for 'B', which is responsible for forwarding an invocation request from the replicated object to 'C'.

(b) Returning results from 'C' using the same idea: a coordinator is responsible for returning the result to all 'B's. Note the single result returned to 'A'.

Quorum-Based Protocols

Clients must request and acquire permissions from multiple replicas before either reading/writing a replicated data item.

Example:

- A file is replicated within a distributed file system.
- To update a file, a process must get approval from a majority of the replicas to perform a write.
  - o The replicas need to agree to also perform the write.

- After the update, the file has a new version # associated with it (and it is set at all the updated replicas).
- To read, a process contacts a majority of the replicas and asks for the version # of the files.
  - o If the version # is the same, then the file must be the most recent version, and the read can proceed.

Gifford's Method

- To read a file of which N replicas exist a client needs to assemble a read quorum, an arbitrary collection of any NR servers, or more.
- To modify a file, a write quorum of at least NW servers is required.
- The values of NR and NW are subject to the following two constraints:

  NR + NW > N

  NW > N/2
  - o First constraint prevents read-write conflicts
  - o Second constraint prevents write-write conflicts.

Only after the appropriate number of servers has agreed to participate can a file be read or written.

Example

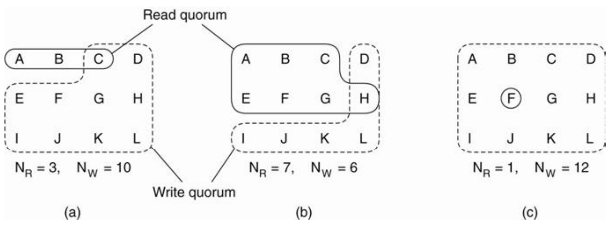
- NR = 3 and NW = 10
- Most recent write quorum consisted of the 10 servers C through L.
- All get the new version and the new version number.
- Any subsequent read quorum of three servers will have to contain at least one member of this set.
- When the client looks at the version numbers, it will know which is most recent and take that one.

  Three examples of the voting algorithm:

  (a) A correct choice of read and write set.

  (b) A choice that may lead to write-write conflicts.

  (c) A correct choice, known as ROWA (read one, write all).



(b) a write-write conflict may occur because NW? N/2.
  - o If one client chooses {A,B,C,E,F,G} as its write set and another client chooses {D,H,I,J,K,L} as its write set, then the two updates will both be accepted without detecting that they actually conflict.

(c) NR = 1, making it possible to read a replicated file by finding any copy and using it.
  - o poor performance, becausewrite updates need to acquire all copies. (aka Read-One, Write-All (ROWA)). T

Cache-Coherence Protocols

These are a special case, as the cache is typically controlled by the client not the server.

Coherence Detection Strategy:

- o When are inconsistencies actually detected?
- o Statically at compile time: extra instructions inserted.
- o Dynamically at runtime: code to check with the server.

3.2 Fault Tolerance

3.2.1 Introduction

Q6. What is dependability?

Ans :

To understand the role of fault tolerance in distributed systems we first need to take a closer look at what it actually means for a distributed system to tolerate faults. Being fault tolerant is strongly related to what are called dependable systems. Dependability is a term that covers a number of useful requirements for distributed systems including the following :

1. Availability
2. Reliability
3. Safety
4. Maintainability

1. Availability

Availability is defined as the property that a system is ready to be used immediately. In general, it refers to the probability that the system is operating correctly at any given moment and is available to perform its functions on behalf of its users. In other words, a highly available system is one that will most likely be working at a given instant in time.

2. Reliability

Reliability refers to the property that a system can run continuously without failure. In contrast to availability, reliability is defined in terms of a time interval instead of an instant in time. A highly-reliable system is one that will most likely continue to work without interruption during a relatively long period of time.

3. Safety

Safety refers to the situation that when a system temporarily fails to operate correctly, nothing catastrophic happens. For example, many process control systems, such as those used for controlling nuclear power plants or sending people into space, are required to provide a high degree of safety. If such control systems temporarily fail for only a very brief moment, the effects could be disastrous.

4. Maintainability

Finally, maintainability refers to how easy a failed system can be repaired. A highly maintainable system may also show a high degree of availability, especially if failures can be detected and repaired automatically. However, as we shall see later in this chapter, automatically recovering from failures is easier said than done.

A system is said to fail when it cannot meet its promises. In particular, 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 (completely) provided. An error is a part of a system's state that may lead to a failure.

For example, when transmitting packets across a network, it is to be expected that some packets have been damaged when they arrive at the receiver. Damaged in this context means that the receiver may incorrectly sense a bit value (e.g., reading a 1 instead of a 0), or may even be unable to detect that something has arrived.

Q7. What is failure model? Explain different types of failures in distributed systems

Ans :

Failure Models

A system that fails is not adequately providing the services it was designed for. A distributed system as a collection of servers that communicate with one another and with their clients. If the server fails by any means it cannot communicate with the cleints.

The failures of server can be different types which are discussed below:

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure	A server fails to respond to incoming requests
Receive omission	A server fails to receive incoming messages
Send omission	A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure	A server's response is incorrect
Value failure	The value of the response is wrong
State transition failure	The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

Fig.: Different types of failures

Crash Failure

A crash failure occurs when a server prematurely halts, but was working correctly until it stopped. A typical example of a crash failure is an operating system that comes to a grinding halt, and for which there is only one solution: reboot it..

Omission Failure

➢ An omission failure occurs when a server fails to respond to a request. In the case of a receive omission failure, possibly the server never got the request in the first place..

  o Also, a receive omission failure will generally not affect the current state of the server, as the server is unaware of any message sent to it.

  o A send omission failure happens when the server has done its work, but somehow fails in sending a response.

Timing Failures

Timing failures occur when the response lies outside a specified real-time interval..Ifa server responds too late, in which case a performance failure is said to occur.

Response Failure

➢ A serious type of failure is a response failure, by which the server's response is simply incorrect. Two kinds of response failures may happen.

  o A value failure, a server simply provides the wrong reply to a request. For example, a search engine that systematically returns Web pages not related to any of the search terms used, has failed.

  o A state transition failure. This kind of failure happens when the server reacts unexpectedly to an incoming request. For example, if a server receives a message it cannot recognize, a state transition failure

Arbitrary Failure

Arbitrary failures are closely related to crash failures. They are also referred to as fail-stop failures. In effect, a fail-stop server will simply stop producing output in such a way that its halting can be detected by other processes.

Q8. Write about Failure Masking by Redundancy

Ans :

If a system is to be fault tolerant, the best it can do is to try to hide the occurrence of failures from other processes.

The key technique for masking faults is to use redundancy.

Three kinds are possible:

- information redundancy $\rightarrow$ It refers to too much of information is stored in the servers. Error correction and error code techniques are used
- time redundancy $\rightarrow$ With time redundancy, an action is performed, and then, if need be, it is performed again. Transactions use this approach. If a transaction aborts, it can be redone with no harm.
- physical redundancy $\rightarrow$ With physical redundancy, extra equipment or processes are added to make it possible for the system as a whole to tolerate the loss or malfunctioning of some components. Physical redundancy can thus be done either in hardware or in software.

3.2.2 Process Resilience

Q9. Explain what is process resilience?and how fault tolerance can be achieved by using fault tolerance.

Ans :

The protection against process failures, can be achieved by replicating processes into groups

Design Issues

The key approach to tolerating a faulty process is to organize several identical processes into a group. The key property that all groups have is that when a message is sent to the group itself, all members of the group receive it.

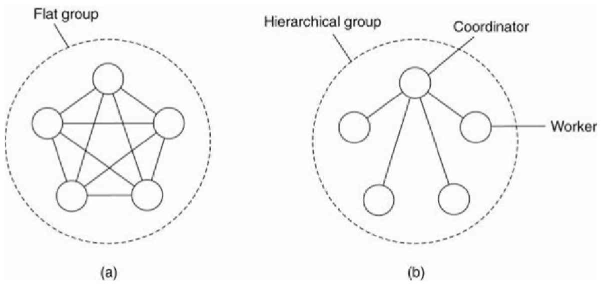
Process groups may be dynamic. New groups can be created and old groups can be destroyed. A process can join a group or leave one during system operation. A process can be a member of several groups at the same time. Consequently, mechanisms are needed for managing groups and group membership.

Flat Groups versus Hierarchical Groups

In flat groups all the processes are equal and all decisions are made collectively.

In hierarchical groups, some kind of hierarchy exists. For example, one process is the coordinator and all the others are workers.

Fig.: (a) Communication in a flat group. (b) Communication in a simple hierarchical group.



Advantages and Disadvantages

- The flat group is symmetrical and has no single point of failure. If one of the processes crashes, the group simply becomes smaller, but can otherwise continue.
- disadvantage is that decision making is more complicated. For example, to decide anything, a vote often has to be taken, incurring some delay and overhead.
- The hierarchical group has the opposite properties. Loss of the coordinator brings the entire group to a grinding halt, but as long as it is running, it can make decisions without bothering everyone else.

Group Membership

- When group communication is present, some method is needed for creating and deleting groups, as well as for allowing processes to join and leave groups.
- One possible approach is to have a group server to which all these requests can be sent. The group server can then maintain a complete data base of all the groups and their exact membership.
- The opposite approach is to manage group membership in a distributed way. For example, if (reliable) multicasting is available, an outsider can send a message to all group members announcing its wish to join the group .
- To leave a group, a member just sends a goodbye message to everyone. The other members have to discover this experimentally by noticing that the crashed member no longer responds to anything.
- Another issue is that leaving and joining have to be synchronous with data messages being sent
- final issue relating to group membership is what to do if so many machines go down that the group can no longer function at all. Some protocol is needed to rebuild the group.

Q10. What are the Agreements in Faulty Systems?

Ans :

Organizing replicated processes into a group helps to increase fault tolerance.

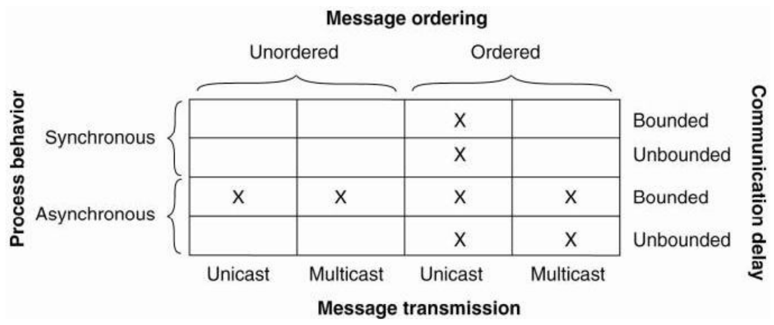
The general goal of distributed agreement algorithms is to have all thenonfaulty processes reach consensus on some issue, and to establish that consensus within a finite number of steps.

These are some cases to distinguish:

1. Synchronous versus asynchronous systems. A system is synchronous if and only if the processes are known to operate in a lock-step mode. Formally, this means that there should be some constant c 1, such that if any processor has taken c + 1 steps, every other process has taken at least 1 step. A system that is not synchronous is said to be asynchronous.

2. Communication delay is bounded or not. Delay is bounded if and only if we know that every message is delivered with a globally and predetermined maximum time.
3. Message delivery is ordered or not. In other words, we distinguish the situation where messages from the same sender are delivered in the order that they were sent, from the situation in which we do not have such guarantees.
4. Message transmission is done through unicasting or multicasting.

The following figure shows Circumstances under which distributed agreement can be reached.



Process behavior	Communication delay	Unordered Unicast	Unordered Multicast	Ordered Unicast	Ordered Multicast
Synchronous	Bounded			X	
Synchronous	Unbounded			X	
Asynchronous	Bounded	X	X	X	X
Asynchronous	Unbounded			X	X

The problem is also known as the Byzantine agreement problem, referring to the numerous wars in which several armies needed to reach agreement on,

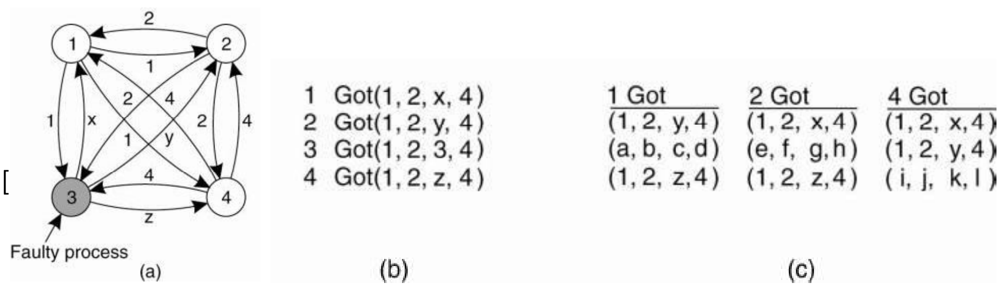
In this case, we assume that processes are synchronous, messages are unicast while preserving ordering, and communication delay is bounded.

We assume that there are N processes, where each process i will provide a value vi to the others. The goal is let each process construct a vector V of length N, such that if process i is nonfaulty, V [i] = vi. Otherwise, V [i] is undefined. We assume that there are at most k faulty processes.

The working of the algorithm for the case of N = 4 and k = 1.

For these parameters, the algorithm operates in four steps.

- In step 1, every nonfaulty process i sends vi to every other process using reliable unicasting. Faulty processes may send anything. Moreover, because we are using multicasting, they may send different values to different processes. Let vi =i.
  - o In following figure we see that process 1 reports 1, process 2 reports 2, process 3 lies to everyone, giving x, y, and z, respectively, and process 4 reports a value of 4.
- In step 2, the results of the announcements of step 1 are collected together in the form of the vectors of Fig. 8-5(b).
  - o The following figure shows , the Byzantine agreement problem for three nonfaulty and one faulty process. (a) Each process sends their value to the others. (b) The vectors that each process assembles based on (a). (c) The vectors that each process receives in step 3.



1 Got(1, 2, x, 4)
2 Got(1, 2, y, 4)
3 Got(1, 2, 3, 4)
4 Got(1, 2, z, 4)

(b)

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(c)

- Step 3 consists of every process passing its vector from Fig. (b) to every other process. In this way, every process gets three vectors, one from every other process.
  - Here, too, process 3 lies, inventing 12 new values, a through l.
- In step 4, each process examines the ith element of each of the newly received vectors. If any value has a majority, that value is put into the result vector. If no value has a majority, the corresponding element of the result vector is marked UNKNOWN.

Q11. Write about failure detection process.

Ans :

Failure Detection

Failure detection is one of the cornerstones of fault tolerance in distributed systems. What it all boils down to is that for a group of processes, nonfaulty members should be able to decide who is still a member, and who is not. In other words, we need to be able to detect when a member has failed.

When it comes to detecting process failures, there are essentially only two mechanisms. Either processes actively send "are you alive?" messages to each other and only when it can be guaranteed that there is enough communication between processes.

Due to unreliable networks, simply stating that a process has failed because it does not return an answer to a ping message may be wrong. In other words, it is quite easy to generate false positives. If a false positive has the effect that a perfectly healthy process is removed from a membership list, then clearly we are doing something wrong.

Another serious problem is that timeouts are just plain crude.

There are various issues that need to be taken into account when designing a failure detection subsystem. For example, failure detection can take place through gossiping in which each node regularly announces to its neighbors that it is still up and running.

Failure detection can also be done as a side-effect of regularly exchanging information with neighbors,

Another important issue is that a failure detection subsystem should ideally be able to distinguish network failures from node failures.

3.2.3 Reliable Clientserver Communication

Q12. Explain point to point communication in distributed systems.

Ans :

Point-to-Point Communication

In many distributed systems, reliable point-to-point communication is established by making use of a reliable transport protocol, such as TCP. TCP masks omission failures, which occur in the form of lost messages, by using acknowledgments and retransmissions. Such failures are completely hidden from a TCP client.

A crash failure may occur when a TCP connection is abruptly broken so that no more messages can be transmitted through the channel.

RPC Semantics in the Presence of Failures

The goal of RPC is to hide communication by making remote procedure calls look just like local ones.

To structure our discussion, let us distinguish between five different classes of failures that can occur in RPC systems, as follows:

1. The client is unable to locate the server.
2. The request message from the client to the server is lost.
3. The server crashes after receiving a request.
4. The reply message from the server to the client is lost.
5. The client crashes after sending a request.

Each of these categories poses different problems and requires different solutions.

1. Client Cannot Locate the Server

- It can happen that the client cannot locate a suitable server. All servers might be down, When the client is eventually run, the binder will be unable to match it up with a server and will report failure. While this mechanism is used to protect the client from accidentally trying to talk to a server that may not agree with it in terms of what parameters are required or what it is supposed to do, the problem remains of how should this failure be dealt with.

- One possible solution is to have the error raise an exception. In some languages, programmers can write special procedures that are invoked upon specific errors, such as division by zero.
- This approach, too, has drawbacks. not every language has exceptions or signals. Another point is that having to write an exception or signal handler destroys the transparency we have been trying to achieve.

2. Lost Request Messages

The second item on the list is dealing with lost request messages. This is the easiest one to deal with: just have the operating system or client stub start a timer when sending the request. If the timer expires before a reply or acknowledgment comes back, the message is sent again. If the message was truly lost, the server will not be able to tell the difference between the retransmission and the original, and everything will work fine.

3. Server Crashes

The next failure on the list is a server crash.. A request arrives, is carried out, and a reply is sent. Now consider , A request arrives and is carried out, just as before, but the server crashes before it can send the reply. Finally, Again a request arrives, but this time the server crashes before it can even be carried out. And, of course, no reply is sent back.

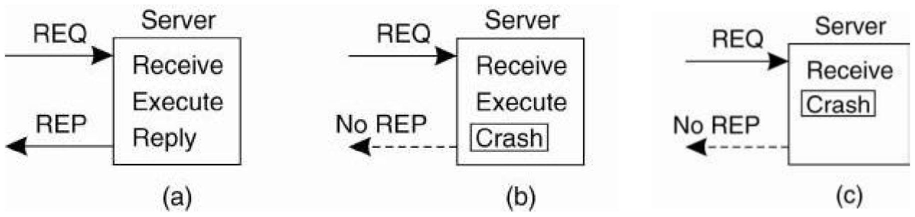


Figure: A server in client-server communication
(a) The normal case. (b) Crash after execution. (c) Crash before execution

Assume that the server crashes and subsequently recovers. It announces to all clients that it has just crashed but is now up and running again. The problem is that the client does not know whether its request to print some text will actually be carried out.

There are four strategies the client can follow.

- First, the client can decide to never reissue a request, at the risk that the text will not be printed.
- Second, it can decide to always reissue a request, but this may lead to its text being printed twice.
- Third, it can decide to reissue a request only if it did not yet receive an acknowledgment that its print request had been delivered to the server. In that case, the client is counting on the fact that the server crashed before the print request could be delivered.
- The fourth and last strategy is to reissue a request only if it has received an acknowledgment for the print request.

With two strategies for the server, and four for the client, there are a total of eight combinations to consider.

Note that there are three events that can happen at the server: send the completion message (M), print the text (P), and crash (C). These events can occur in six different orderings:

(i) MPC: A crash occurs after sending the completion message and printing the text.

(ii) MC (P): A crash happens after sending the completion message, but before the text could be printed.

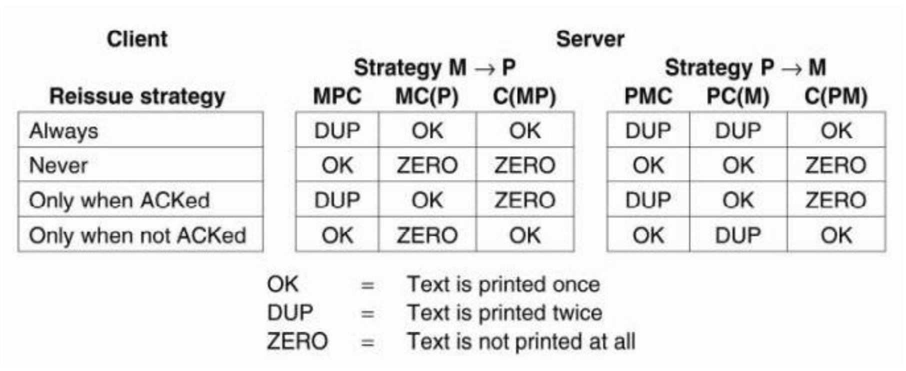
(ii) PMC: A crash occurs after sending the completion message and printing the text.

(iv) PC(M): The text printed, after which a crash occurs before the completion message could be sent.

(v) C (PM): A crash happens before the server could do anything.

(vi) C (MP): A crash happens before the server could do anything.

The parentheses indicate an event that can no longer happen because the server already crashed.



Client	Server					
	Strategy M → P			Strategy P → M		
Reissue strategy	MPC	MC(P)	C(MP)	PMC	PC(M)	C(PM)
Always	DUP	OK	OK	DUP	DUP	OK
Never	OK	ZERO	ZERO	OK	OK	ZERO
Only when ACKed	DUP	OK	ZERO	DUP	OK	ZERO
Only when not ACKed	OK	ZERO	OK	OK	DUP	OK

OK	=	Text is printed once
DUP	=	Text is printed twice
ZERO	=	Text is not printed at all

Fig.: Different combinations of client and server strategies in the presence of server crashes.

The possibility of server crashes radically changes the nature of RPC and clearly distinguishes single-processor systems from distributed systems.

4. Lost Reply Messages

Lost replies can also be difficult to deal with. The obvious solution is just to rely on a timer again that has been set by the client's operating system. If no reply is forthcoming within a reasonable period, just send the request once more.

In particular, some operations can safely be repeated as often as necessary with no damage being done. A request such as asking for the first 1024 bytes of a file has no side effects and can be executed as often as necessary without any harm being done.

5. Client Crashes

The final item on the list of failures is the client crash. At this point a computation is active and no parent is waiting for the result. Such an unwanted computation is called an orphan.

Orphans can cause a variety of problems that can interfere with normal operation of the system. As a bare minimum, they waste CPU cycles. They can also lock files or otherwise tie up valuable resources. Finally, if the client reboots and does the RPC again, but the reply from the orphan comes back immediately afterward, confusion can result.

The disadvantage of this scheme is the horrendous expense of writing a disk record for every RPC.

3.2.4 Reliable Group Communication

Q13. Explain basic reliable multicasting schemes.

Ans :

Most transport layers offer reliable point-to-point channels, they rarely offer reliable communication to a collection of processes. The best they can offer is to let each process set up a point-to-point connection to each other process it wants to communicate with.

Reliable multicasting is that a message that is sent to a process group should be delivered to each member of that group.

To cover such situations, a distinction should be made between reliable communication in the presence of faulty processes, and reliable communication when processes are assumed to operate correctly.

In the first case, multicasting is considered to be reliable when it can be guaranteed that all nonfaulty group members receive the message.

The situation becomes simpler if we assume agreement exists on who is a member of the group and who is not.

A simple solution is shown in following Fig. The sending process assigns a sequence number to each message it multicasts. We assume that messages are received in the order they are sent. In this way, it is easy for a receiver to detect it is missing a message. 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. Alternatively, the sender may automatically retransmit the message when it has not received all acknowledgments within a certain time.

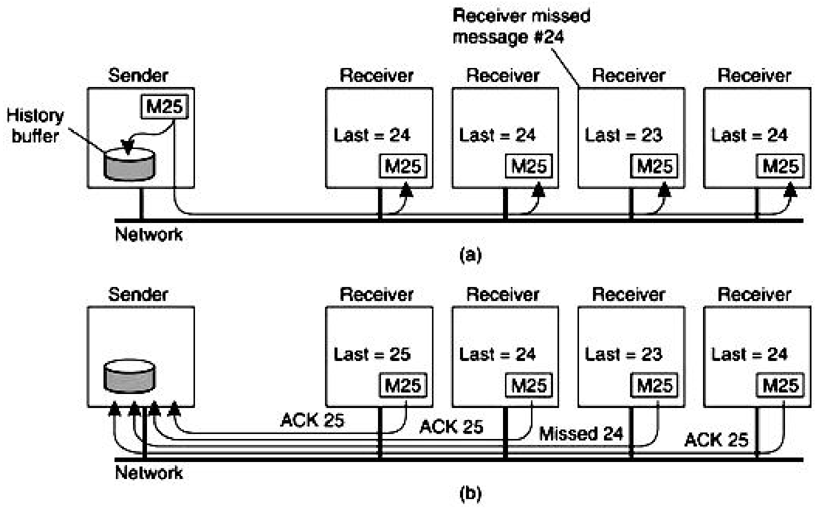


Fig.: L A simple solution to reliable multicasting when all receivers are known and are assumed not to fail. (a) Message transmission. (b) Reporting feedback.

Scalability in Reliable Multicasting

The main problem with the reliable multicast scheme just described is that it cannot support large numbers of receivers. If there are N receivers, the sender must be prepared to accept at least N acknowledgments. With many receivers, the sender may be swamped with such feedback messages, which is also referred to as a feedback implosion. In addition, we may also need to take into account that the receivers are spread across a wide-area network.

One solution to this problem is not to have receivers acknowledge the receipt of a message. Instead, a receiver returns a feedback message only to inform the sender it is missing a message.

Another problem with returning only negative acknowledgments is that the sender will, in theory, be forced to keep a message in its history buffer forever.

Several proposals for scalable reliable multicasting exist.

Non-hierarchical Feedback Control

The key issue to scalable solutions for reliable multicasting is to reduce the number of feedback messages that are returned to the sender. A popular model that has been applied to several wide-area applications is feedback suppression. This scheme underlies the Scalable Reliable Multicasting (SRM) protocol.

First, in SRM, receivers never acknowledge the successful delivery of a multicast message, but instead, report only when they are missing a message. How message loss is detected is left to the application. Only negative acknowledgments are returned as feedback. Whenever a receiver notices that it missed a message, it multicasts its feedback to the rest of the group.

Multicasting feedback allows another group member to suppress its own feed-back. Suppose several receivers missed message m. Each of them will need to return a negative acknowledgment to the sender, S, so that m can be retransmitted. However, if we assume that retransmissions are always multicast to the entire group, it is sufficient that only a single request for retransmission reaches S.

For this reason, a receiver R that did not receive message m schedules a feed-back message with some random delay. That is, the request for retransmission is not sent until some random time has elapsed. If, in the meantime, another request for retransmission for m reaches R, R will suppress its own feedback, knowing that m will be retransmitted shortly. In this way, ideally, only a single feedback message will reach S, which in turn subsequently retransmits m.

Feedback suppression has shown to scale reasonably well, and has been used as the underlying mechanism for a number of collaborative Internet applications, such as a shared whiteboard. However, the approach also introduces a number of serious problems.

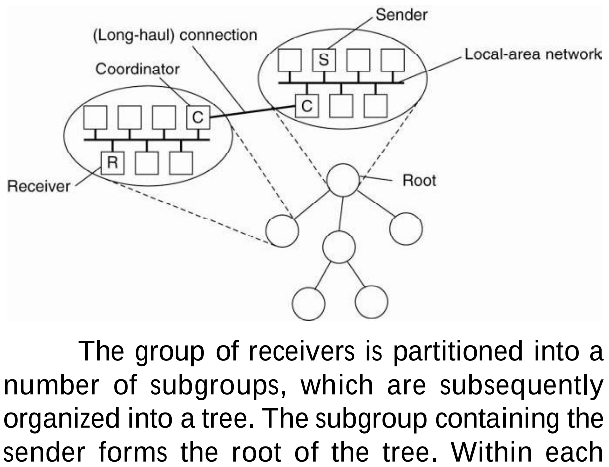
Another problem is that multicasting feedback also interrupts those processes to which the message has been successfully delivered. In other words, other receivers are forced to receive and process messages that are useless to them. The only solution to this problem is to let receivers that have not received message m join a separate multicast group for m,

To enhance the scalability of SRM, it is useful to let receivers assist in local recovery. In particular, if a receiver to which message m has been successfully delivered, receives a request for retransmission, it can decide to multicast m even before the retransmission request reaches the original sender.

Hierarchical Feedback Control

Feedback suppression as just described is basically a nonhierarchical solution. However, achieving scalability for very large groups of receivers requires that hierarchical approaches are adopted.

In Figure, The essence of hierarchical reliable multicasting. Each local coordinator forwards the message to its children and later handles retransmission requests.



The group of receivers is partitioned into a number of subgroups, which are subsequently organized into a tree. The subgroup containing the sender forms the root of the tree. Within each subgroup, any reliable multicasting scheme that works for small groups can be used.

Each subgroup appoints a local coordinator, which is responsible for handling retransmission requests of receivers contained in its subgroup. The local coordinator will thus have its own history buffer. If the coordinator itself has missed a message m, it asks the coordinator of the parent subgroup to retransmit m. In a scheme based on acknowledgments, a local coordinator sends an acknowledgment to its parent if it has received the message. If a coordinator has received acknowledgments for message m from all members in its subgroup, as well as from its children, it can remove m from its history buffer.

The main problem with hierarchical solutions is the construction of the tree. In many cases, a tree needs to be constructed dynamically. One approach is to make use of the multicast tree in the underlying network, if there is one.

Atomic Multicast

Let us now return to the situation in which we need to achieve reliable multicasting in the presence of process failures.

To see why atomicity is so important, consider a replicated database constructed as an application on top of a distributed system. The distributed system offers reliable multicasting facilities. It allows the construction of process groups to which messages can be reliably sent. The replicated database is therefore constructed as a group of processes, one process for each replica. Update operations are always multicast to all replicas and subsequently performed locally. In other words, we assume that an active-replication protocol is used.

Suppose that now that a series of updates is to be performed, but that during the execution of one of the updates, a replica crashes. Consequently, that update is lost for that replica but on the other hand, it is correctly performed at the other replicas.

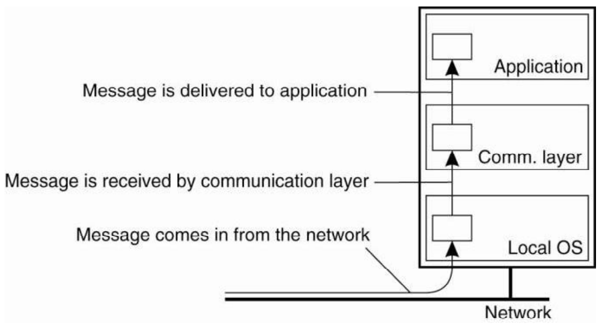
When the replica that just crashed recovers, at best it can recover to the same state it had before the crash.

Consequently, atomic multicasting ensures that nonfaulty processes maintain a consistent view of the database, and forces reconciliation when a replica recovers and rejoins the group.

Virtual Synchrony

Reliable multicast in the presence of process failures can be accurately defined in terms of process groups and changes to group membership.. Within this communication layer, messages are sent and received. A received message is locally buffered in the communication layer until it can be delivered to the application that is logically placed at a higher layer.

Figure shows the logical organization of a distributed system to distinguish between message receipt and message delivery.



Application
Message is delivered to application
Comm. layer
Message is received by communication layer
Message comes in from the network
Local OS
Network

The whole idea of atomic multicasting is that a multicast message m is uniquely associated with a list of processes to which it should be delivered. This delivery list corresponds to a group view, namely, the view on the set of processes contained in the group, which the sender had at the time message m was multicast.

Now suppose that the message m is multicast at the time its sender has group view G. Furthermore, assume that while the multicast is taking place, another process joins or leaves the group. This change in group membership is naturally announced to all processes in G. Stated somewhat differently, a view change takes place by multicasting a message vc announcing the joining or leaving of a process. We now have two multicast messages simultaneously in transit: m and vc. What we need to guarantee is that m is either delivered to all processes in G before each one of them is delivered message vc, or m is not delivered at all.

Message Ordering

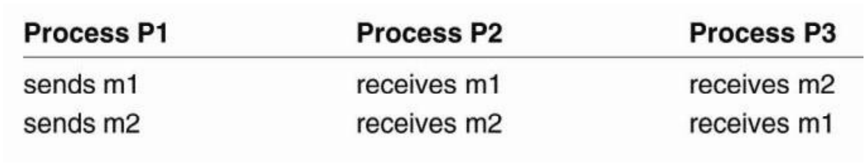
Virtual synchrony allows an application developer to think about multicasts as taking place in epochs that are separated by group membership changes. In general, four different orderings are distinguished:

1. Unordered multicasts
2. FIFO-ordered multicasts
3. Causally-ordered multicasts
4. Totally-ordered multicasts

A reliable, unordered multicast is a virtually synchronous multicast in which no guarantees are given concerning the order in which received messages are delivered by different processes. To explain, assume that reliable multicasting is supported by a library providing a send and a receive primitive. The receive operation blocks the calling process until a message is delivered to it.

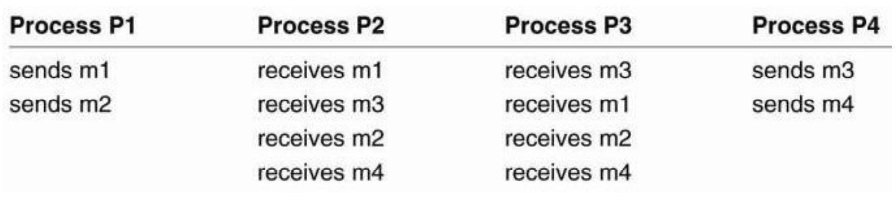
Now suppose a sender P1 multicasts two messages to a group while two other processes in that group are waiting for messages to arrive, as shown in Fig. Assuming that processes do not crash or leave the group during these multicasts, it is possible that the communication layer at P2 first receives message m1 and then m 2. Because there are no message-ordering constraints, the messages may be delivered to P2 in the order that they are received. In contrast, the communication layer at P3 may first receive message m 2 followed by m 1, and delivers these two in this same order to P3.

Figure : Three communicating processes in the same group. The ordering of events per process is shown along the vertical axis.



Process P1	Process P2	Process P3
sends m1	receives m1	receives m2
sends m2	receives m2	receives m1

Figure: Four processes in the same group with two different senders, and a possible delivery order of messages under FIFO-ordered multicasting.



Process P1	Process P2	Process P3	Process P4
sends m1	receives m1	receives m3	sends m3
sends m2	receives m3	receives m1	sends m4
	receives m2	receives m2	
	receives m4	receives m4	

If process P2 receives m 1 before m 3, it may deliver the two messages in that order. Meanwhile, process P3 may have received m 3 before receiving m 1. FIFO ordering states that P3 may deliver m 3 before m 1, although this delivery order is different from that of P2.

3.2.5 Distributed Commit

Q14. Write about two phase commit protocol.

Ans :

Two-Phase Commit

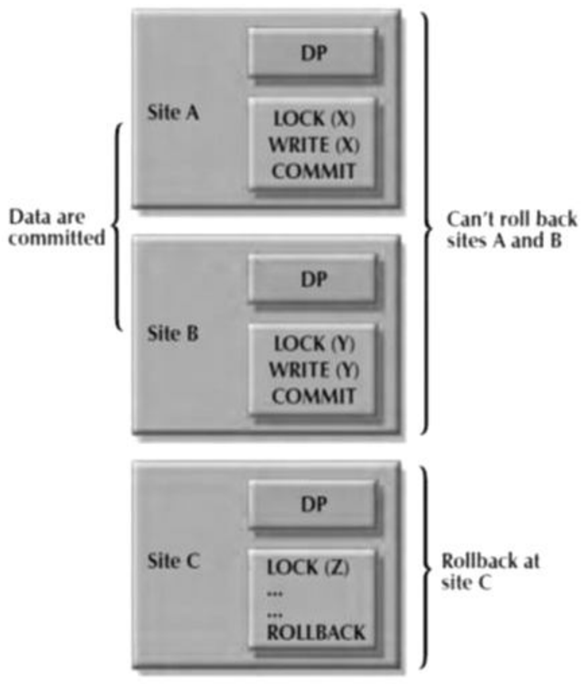
The two-phase commit protocol (2PC) is consider a distributed transaction involving the participation of a number of processes each running on a different machine. Assuming that no failures occur, the protocol consists of the following two phases, each consisting of two steps

1. The coordinator sends a VOTE_REQUEST message to all participants.
2. When a participant receives a VOTE_REQUEST message, it returns either a VOTE_COMMIT message to the coordinator telling the coordinator that it is prepared to locally commit its part of the transaction, or otherwise a VOTE_ABORT message.

3. The coordinator collects all votes from the participants. If all participants have voted to commit the transaction, then so will the coordinator. In that case, it sends a GLOBAL_COMMIT message to all participants. However, if one participant had voted to abort the transaction, the coordinator will also decide to abort the transaction and multicasts a GLOBAL_ABORT message.

4. Each participant that voted for a commit waits for the final reaction by the coordinator. If a participant receives a GLOBAL_COMMIT message, it locally commits the transaction. Otherwise, when receiving a GLOBAL_ABORT message, the transaction is locally aborted as well.

The first phase is the voting phase, and consists of steps 1 and 2. The second phase is the decision phase, and consists of steps 3 and 4.



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.

Q15. Explain about three phase commit protocol.

Ans :

Three-Phase Commit

The 3PC protocol is designed as a non-blocking protocol. In this context, it is necessary to mention that 2PC protocol is not a non-blocking protocol, as in certain circumstances it is possible for sites to become blocked. The coordinator can be blocked in the wait state, whereas the participants can be blocked in the ready state. The requirements for the 3PC protocol are as follows:

(i) Network partition should not occur.

(ii) All sites should not fail simultaneously, that is, at least one site must be available always.

(iii) At the most k sites can fail simultaneously, where k is less than total number of sites in the distributed system.

In 3PC, a new phase is introduced, known as pre-commit phase, in between the voting phase and the global decision phase, for eliminating the uncertainly period for participants that voted commit and are waiting for the global decision from the coordinator. In 3PC, if all participants vote for commit, the coordinator sends a "global pre-commit" message to all participants. A participant who has received a "pre-commit" message to the coordinator will definitely commit by itself, if it has not failed. Each participant's acknowledgements the coordinator sends a "global commit" message to all participants.

Termination protocols for 3PC

Like in 2PC, the termination protocol is used in 3PC to handle timeouts.

Coordinator In 3PC, the coordinator may timeout in four different states: wait, precommit, abort and commit. Timeouts during the abort and the commit states are handled in the same manner as in 2PC, therefore, only three cases are considered here.

(i) Timeout in the wait state - The action taken here is identical to that in the coordinator timeout in the wait state for the 2PC protocol. In this state, the coordinator can decide to globally abort the transaction. Therefore, the coordinator writes an "abort" record in the log and sends a "global_abort" message to all participants.

(ii) Timeout in the precommit state - In this case, the coordinator does not know whether the non-responding participants have already moved to the precommit state or not, but the coordinator can decide to commit the transaction globally as all participants have voted to commit. Hence, the coordinator sends a "prepare-to-commit" message to all participants to move them into the commit state, and then globally commits the transaction by writing a commit record in the log and sending "global_commit" message to all participants.

(iii) Timeout in the commit or abort state - In this state, the coordinator is waiting for all par-ticipants to acknowledge whether they have successfully committed or aborted and timeout occurs. Hence, the participants are at least in the precommit state and can invoke the termina-tion protocol as listed in case (ii) and case (iii) in the following section. Therefore, the coordina-tor is not required to take any special action in this case.

Participant

A participant may timeout in three different states; initial, ready and precommit.

(i) Timeout in the initial state- In this case, the action taken is identical to that in the termination protocol of 2PC.

(ii) Timeout in the ready state- in this state, the participant has voted to commit and is waiting for the global decision from the coordinator. As the communication with the coordinator is lost, the termination protocol continues by electing a new coordinator.

3.2.6 Recovery

Q16. Write about backward and forward recovery.

Ans :

Backward and Forward Error Recovery

There are two approaches for restoring an erroneous state to an error free state.

1. **Forward Error Recovery:** If the nature of the errors and damages caused by the fault can be completely and accurately accessed, then it is possible to remove those errors in the process's state or system's state and enable the process or system to move forward. This technique is known as forward error recovery.

2. **Backward Error Recovery:** If the nature of the errors and damages caused by the fault cannot be completely and accurately accessed, then it is not possible to remove

those errors in the process's state or system's state. To remove the errors or fault, the system state can be restored to the previous error-free stable state of the system. This technique is known as backward error recovery.

Backward Error Recovery Vs Forward Error Recovery

Advantages

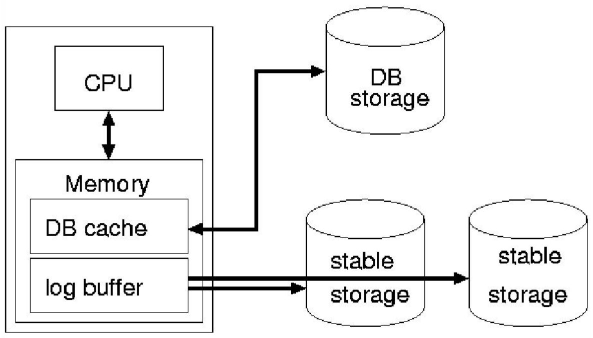
1. Backward error recovery is simpler than forward error recovery.
2. Backward error recovery is independent of an arbitrary fault.

Disadvantages

1. Performance Penalty: The overhead to restore a process or system state to a prior state can be quite high.
2. There is no guarantee that faults will not occur again when processing begins.
3. Some component of the process or system may not be recoverable.

Backward Error Recovery - System Model

The system is assumed to consist of a single machine, connected to aecondary storage system and a stable storage system. A stable storage does not lose information in the event of system failure and used to keep logs & recovery points.



There are two ways to implement backward error recovery:

1. The operation based approach
2. The State based approach

The Operation based Approach:

In this approach, the sufficient details of all the modifications made in the transaction are recorded so that the previous stable state of the system can be restored by reversing all the changes made in that state. The information in the logs contains the name of object, the old state of the object and the new state of the object.

This can be done in two ways:

1. **Updating-in-place:** In Update-in-Place, every update or write operation to anobject updates the object and create a log in the stable storage. The recoverableupdate operation can be implemented as a collection of operations as follows
   (a) A 'do' operation, which does the action or update and write a log record.
   (b) An 'undo' operation, which undoes the action performed by 'do' operation.
   (c) A 'redo' operation, which redoes the action specified by 'do' operation.
   (d) An optional 'display' operation, which displays the log record.

   The major problem with update-in-place that a 'do' operation cannot be undoneif the system crashes after the update operation but before a log record is stored.
2. **The Write-Ahead-Log Protocol:** In Write-Ahead-Log Protocol, The Recoverableupdate operation can be implemented as a collection of operations as follows
   (a) Update an object only after the 'undo' log is recorded.
   (b) Before committing the update, 'redo' and 'undo' logs are recorded.

State Based Recovery

➢ Establish a recovery point where the process state is saved.

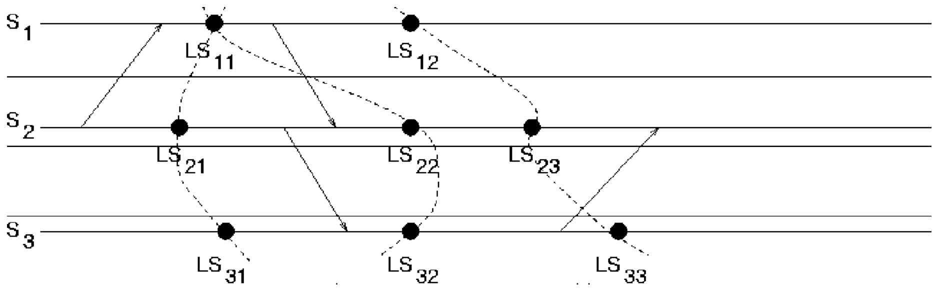
➢ Recovery done by restoring the process state at the recovery, called a checkpoint. This process is called rollback.

- Process of saving called checkpointing or taking a check point.
- Rollback normally done to the most recent checkpoint, hence many checkpoints are done over the execution of a process.
- Shadow pages technique can be used for checkpointing. Page containing the object to be updated is duplicated and maintained as a checkpoint in stable storage.
  - Actual update done on page in secondary storage. Copy in stable storage used for rollback.

Q17. Explain about checkpoints Recovery.

Ans :

A process takes a checkpoint from time to time by saving its state in stable storage need consistent global state. The state of channels corresponding to a global state is the set of messages sent but not yet received. A check point is saved as a local state of a process. A set of check points one per process in the system, is consistent if the saved state form a consistent global state.



Two approaches to create check points

1. **State based or operation based**: processes take checkpoints independently and save all checkpoints in stable storage (asynchronous)
2. **Global checkpointing**: processes coordinate their checkpointing actions such that each process saves only its most recent checkpoints, and the set of checkpoints in the system is guaranteed to be consistent

Independent Checkpointing

Each process independently takes checkpoints, with the risk of a cascaded rollback to system startup:

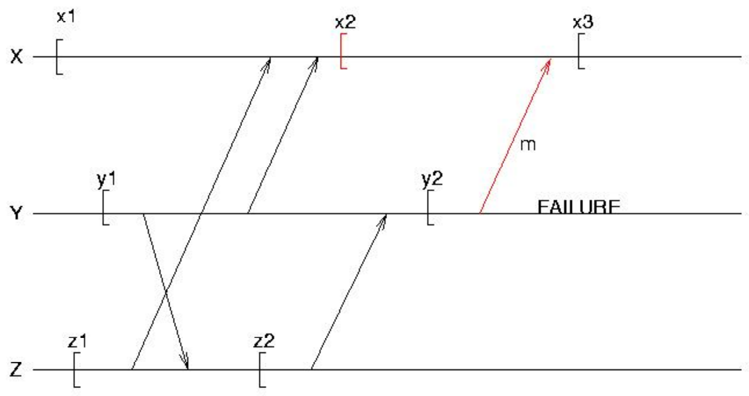
1. Let CP[i](m) denote mth checkpoint of process P_i and INT[i](m) the interval between CP[i](m-1) and CP[i](m)
2. When process P_i sends a message in interval INT[i](m), it piggybacks (i,m)
3. When process P_j receives a message in interval INT[j](n), it records the dependency INT[i](m) '! INT[j](n)
4. The dependency INT[i](m)'! INT[j](n) is saved to stable storage when taking checkpoint CP[j](n)

Observation

If process P_i rolls back to CP[i](m1), P_j must roll back to CP[j](n1).

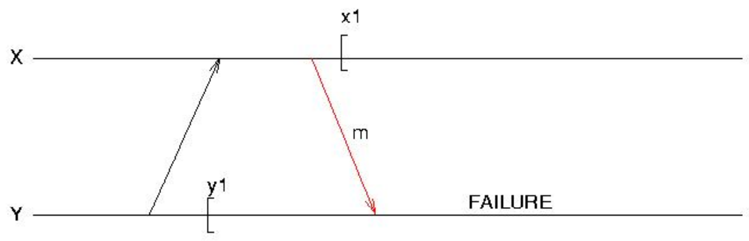
Orphan messages and Domino effect

May lead to unacceptable delays.



- After failure, Y has to roll back to y_2.
- If X only rolls back to x_3, an extra message m will be recorded.
- So it must roll back to x_2.

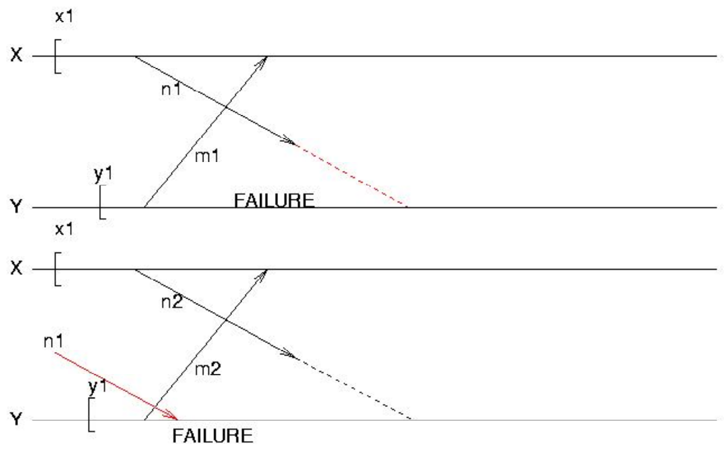
Lost messages



- If the system is restored to state $\{x_1, y_1\}$, message m is lost as X has past the point where it sends m as X has past the point where it sends m.

Livelock

A situation in which a single failure can cause an infinite number of rollbacks, preventing the system from making progress.



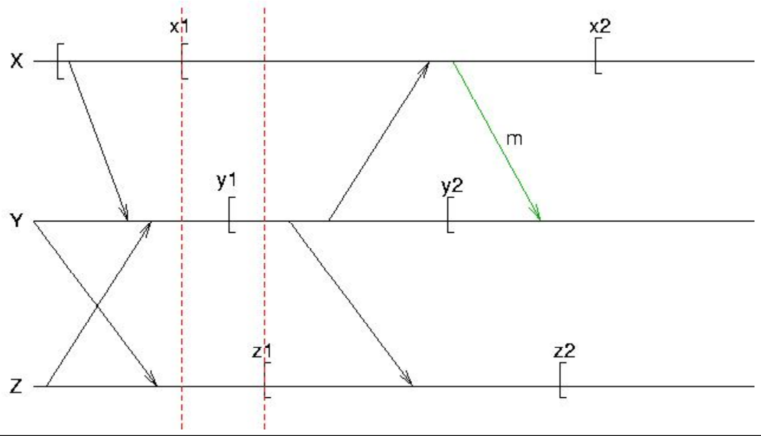
- Y fails before receiving message n_1 (upper figure).
- When Y rolls back to y_1, there's no record of sending m_1. So X has to roll back to x_1.
- When Y recovers, it sends out m_2 and receives n_1 (lower figure).
- X sends out n_2 and receives m_2.
- But X has no record of sending out n_1, so Y has to roll back,
- which forces X to roll back also ....

So, state-based or operation-based are not adequate in many cases.

Coordinated Checkpointing

- Each process takes a checkpoint after a globally coordinated action. Overcoming domino effect and livelocks: checkpoints should not have messages in transit.
- Consistent checkpoints: no message exchange between any pair of processes in the set as well as outside the set during the interval spanned by checkpoints. {x1,y1,z1} is a strongly consistent checkpoint.

Strongly Consistent Set of Checkpoints



Q18. What is message logging ? When should we use message logging?

Ans :

Message logging and consistency

Message logging is used to avoid orphans.

- Process Q has just received and subsequently delivered messages m_1 and m_2
- Assume that m_2 is never logged.
- After delivering m_1 and m_2, Q sends message m_3 to process R
- Process R receives and subsequently delivers m_3.

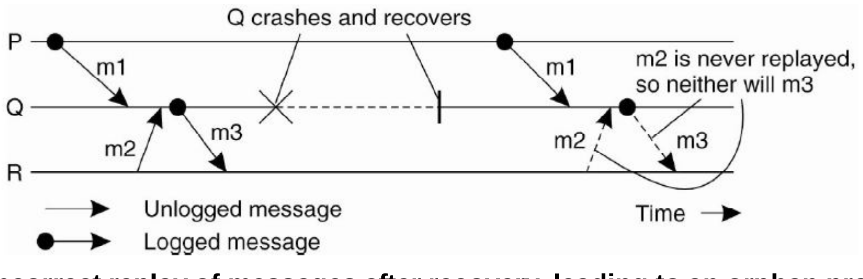


Fig. : Incorrect replay of messages after recovery, leading to an orphan process.

Message-logging schemes

Notations

- **HDR[m]:** The header of message m containing its source, destination, sequence number, and delivery number

  The header contains all information for resending a message and delivering it in the correct order (assume data is reproduced by the application)

  A message m is stable if HDR[m] cannot be lost (e.g., because it has been written to stable storage)

- **DEP[m]:** The set of processes to which message m has been delivered, as well as any message that causally depends on delivery of m

- **COPY[m]:** The set of processes that have a copy of HDR[m] in their volatile memory

Characterization

If C is a collection of crashed processes, then $Q \notin C$ is an orphan if there is a message m such that $Q \in$ DEP[m] and COPY[m] $\subseteq$ C (i.e. All processes that have a copy of m are crashed and Q depends on m!)

Note: We want $\forall$ m $\forall$ C :: COPY[m] $\subseteq$ C => DEP[m] $\subseteq$ C. This is the same as saying that $\forall$ m :: DEP[m] $\subseteq$ COPY[m].

Goal: No orphans means that for each message m,

DEP[m] $\subseteq$ COPY[m]

Pessimistic protocol

For each nonstable message m, there is at most one process dependent on m, that is |DEP[m]| $\leq$ 1.

Consequence

An unstable message in a pessimistic protocol must be made stable before sending a next message.

Optimistic protocol

For each unstable message m, we ensure that if COPY[m] $\subseteq$ C, then eventually also DEP[m] $\subseteq$ C, where C denotes a set of processes that have been marked as faulty

Consequence

To guarantee that DEP[m] $\subseteq$ C, we generally rollback each orphan process Q until $Q \notin$ DEP[m]

3.3 Security

3.3.1 Introduction

Q19. Write about Security in distributed systems.

Ans :

Strategies for securing Distributed Systems

Generally very similar to techniques used in a non-distributed system, only much more difficult to implement …

1. Providing a secure communications channel – authentication, confidentiality and integrity.
2. Handling authorization – who is entitled to use what in the system?
3. Providing effective Security Management.
4. Example systems: SESAME and e-payment systems.

Types of Threats

- **Interception** – unauthorized access to data.
- **Interruption** – a service becomes unavailable.
- **Modification** – unauthorized changes to, and tampering of, data.
- **Fabrication** – non-normal, additional activity.

Security Mechanisms

- **Encryption** – fundamental technique: used to implement confidentiality and integrity.
- **Authentication** – verifying identities.
- **Authorization** – verifying allowable operations.
- **Auditing** – who did what to what and when/ how did they do it?

Matching security mechanisms to threats is only possible when a Policy on security and security issues exists.

20. Write about the Globus Security Policy and architecture.

Ans :

Globus is a system supporting largescale distributed computations in which many hosts, files, and other resources are simultaneously used for doing a computation.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.

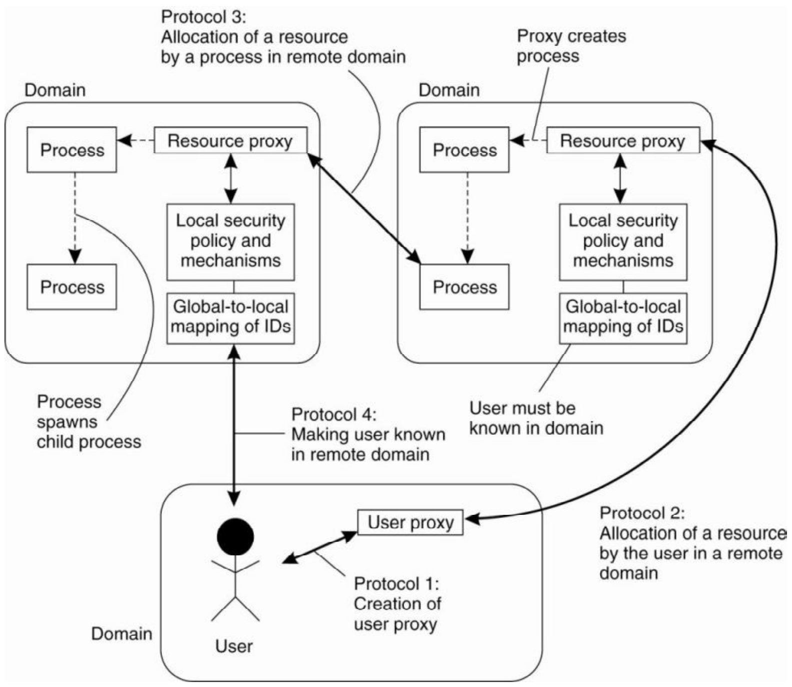
Globus Security Policy

1. The environment consists of multiple administrative domains.
2. Local operations 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:

It consists of entities such as: users, user proxies, resource proxies, and general processes.

- Entities are located in domains and interact with each other.
- Security architecture defines four different protocols
- The Globus security architecture. (See diagram)



Q21. Write about the design issues when considering the security in distributed systems.

Ans :

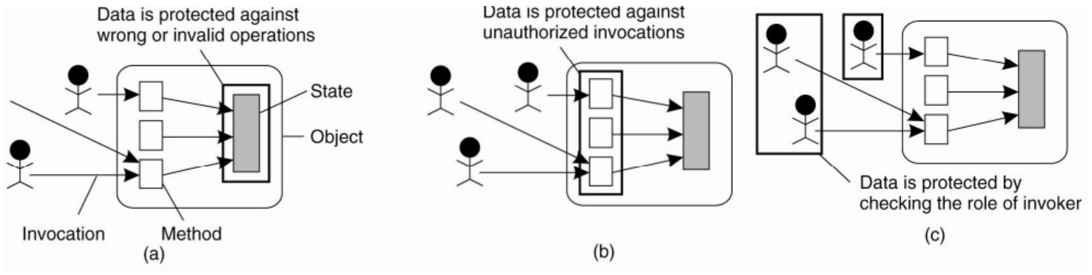
Design Issues

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.


Data is protected against wrong or invalid operations
State
Object
Invocation
Method
(a)
Data is protected against unauthorized invocations
(b)
Data is protected by checking the role of invoker
(c)

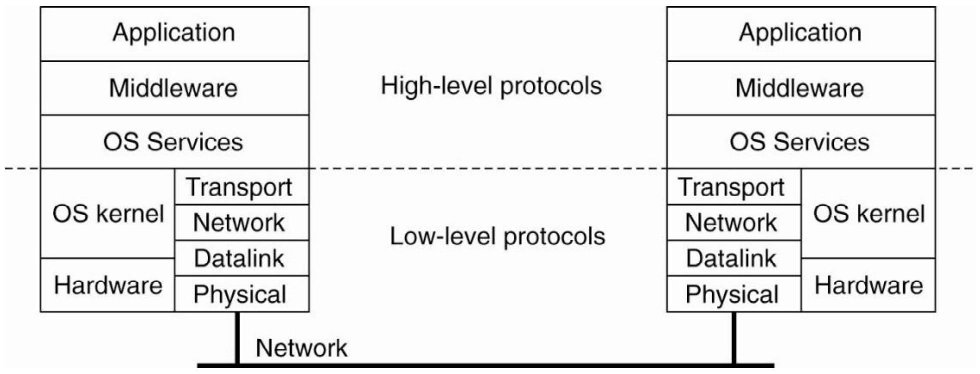
(a) Protection against invalid operations

(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.

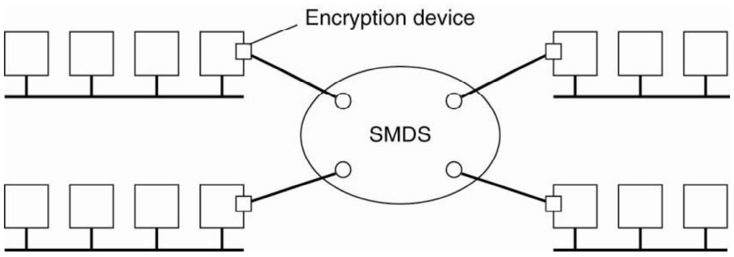


Trust and Security

- 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.

Example

Several sites connected through a wide-area backbone service.



Switched Multi-megabit Data Service (SMDS)

- If the encryption mechanisms cannot be trusted, additional mechanisms can be employed by clients to provide the level of trust required (e.g., using SSL (Secure Sockets Layer).
- Secure Sockets Layer and can be used to securely send messages across a TCP connection.

Q22. Explain various Security Mechanisms

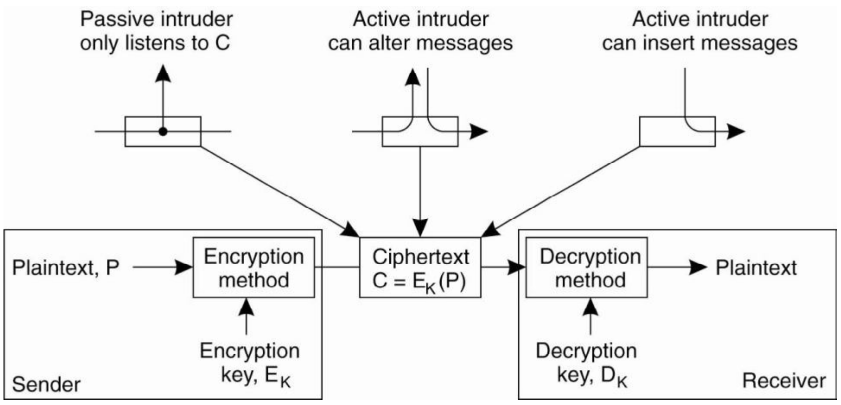
Ans :

Security Mechanisms

Fundamental technique within any distributed systems security environment: Cryptography.

- A sender S wanting to transmit message m to a receiver R.
- The sender encrypts its message into an unintelligible message m', and sends m' to R.
- R must decrypt the received message into its original form m.

Three kinds of Intruders and eavesdroppers in communication.



- Original form of the message that is sent is called the plaintext, P
- Encrypted form is referred to as the ciphertext, illustrated as C.

Notation to relate plaintext, ciphertext, and keys.

$C = E_K(P)$ denotes that the ciphertext C is obtained by encrypting the plaintext P using key K.

$P = D_K(C)$ is used to express the decryption of the ciphertext C using key K, resulting in the plaintext P.

Q23. Explain various Cryptosystems.

Ans :

Types of Cryptosystems

Symmetric: often referred to as conventional cryptography, defined as:

$$P = D_k(E_k(P))$$

Symmetric cryptosystems => referred to as secret-key or shared-key systems

1. because the sender and receiver are required to share the same key,
2. this shared key must be kept secret; no one else is allowed to see the key.
3. Use the notation $K_{A,B}$ to denote a key shared by A and B.

Asymmetric: often referred to as public-key cryptography, defined as:

➢ The keys for encryption and decryption are different, but together form a unique pair.

$$P = D_{k_d}(E_{k_e}(P))$$

➢ There is a separate key K_E for encryption and one for decryption, K_D, such that One of the keys in an asymmetric cryptosystem is kept private; the other is made public.

➢ Notation K_A^+ to denote a public key belonging to A, and K_A^- as its corresponding private key.

Notation used in cryptography

Notation	Description
$K_{A,B}$	Secret key shared by A and B
K_A^+	Public key of A
K_A^-	Private key of A

Hash (One-Way) Functions

A hash function H takes a message m of arbitrary length as input and produces a bit string h having a fixed length as output:

h = H(m)

➢ Given H and m, h is easy to compute.

➢ However, given H and h, it is computationally infeasible to compute m.

➢ That is, the function only works **One-Way**.

Weak Collision Resistence: given m and h = H(m), it is hard to find another m, say, m2, such that m and m2 produce the same h.

Strong Collision Resistence: Given H, it is infeasible to find an m and an m2 such that H(m) and H(m2) produce the same value.

Similarly:

➢ For any encryption function E, it should be computationally infeasible to find the key K when given the plaintext P and associated ciphertext $C = E_K(P)$

➢ Analogous to collision resistance, when given a plaintext P and a key K, it should be effectively impossible to find another key K' such that $E_K(P) = E_{K'}(P)$.

Symmetric Cryptosystems: DES

Example of a cryptographic algorithm: Data Encryption Standard (DES)Used for symmetric cryptosystems.

➢ DES is designed to operate on 64-bit blocks of data.

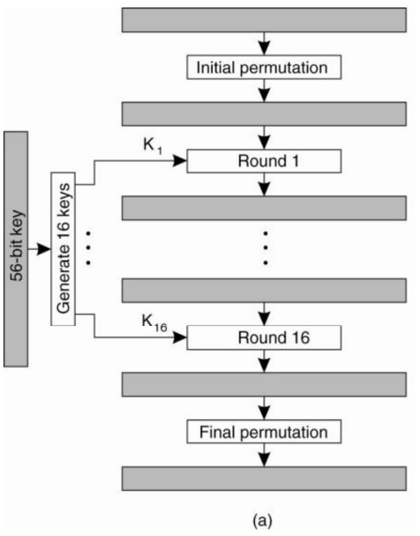
➢ A block is transformed into an encrypted (64 bit) block of output in 16 rounds, where each round uses a different 48-bit key for encryption.

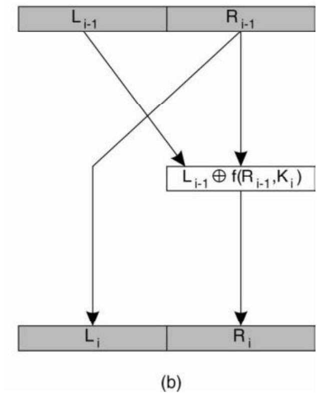
➢ Each of these 16 keys is derived from a 56-bit master key (see figure).

➢ Before an input block starts its 16 rounds of encryption, it is first subject to an initial permutation, of which the inverse is later applied to the encrypted output leading to the final output block.

(a) The principle of DES.

(b) Outline of one encryption round.





Each encryption round i takes the 64-bit block produced by the previous round i - 1 as its input.

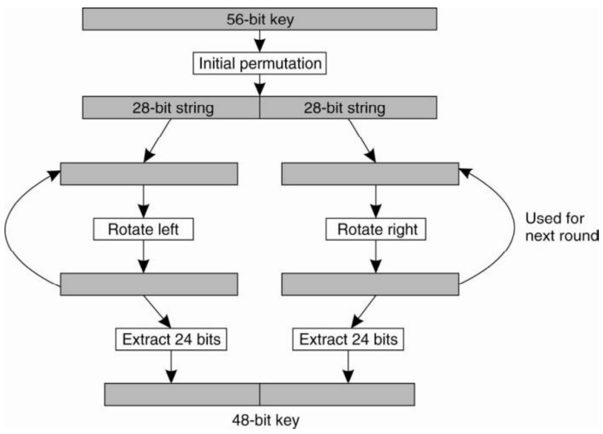
- ➢ The 64 bits are split into a left part Li-1 and a right part Ri-1, each containing 32 bits.
- ➢ The right part is used for the left part in the next round, that is, Li = Ri-1.

  Work is done in the mangler function f.
- ➢ This function takes a 32-bit block Ri-1 as input, together with a 48-bit key Ki, and produces a 32-bit block that is XORed with Li-1 to produce Ri.
  - o (XOR is an abbreviation for the exclusive or operation.)
- ➢ The mangler function first expands Ri-1 to a 48-bit block and XORs it with Ki.
- ➢ The result is partitioned into eight chunks of six bits each.
- ➢ Each chunk is then fed into a different S-box, which is an operation that substitutes each of the 64 possible 6-bit inputs into one of 16 possible 4-bit outputs.
- ➢ The eight output chunks of four bits each are then combined into a 32-bit value and permuted again.

The 48-bit key Ki for round i is derived from the 56-bit master key as follows

- ➢ First, the master key is permuted and divided into two 28-bit halves.
- ➢ For each round, each half is first rotated one or two bits to the left, after which 24 bits are extracted.
- ➢ Together with 24 bits from the other rotated half, a 48-bit key is constructed.

Details of per-round key generation in DES.



Principle of DES is quite simple:

- ➢ Algorithm is difficult to break using analytical methods.
- ➢ Brute-force attack by simply searching for a key that will do the job has become easy as has been demonstrated a number of times.
- ➢ DES three times in a special encrypt-decrypt-encrypt mode with different keys, also known as Triple DES is much more safe and is still often used.

Public-Key Cryptosystems: RSA

- ➢ Public-key systems: RSA : named after its inventors: Rivest, Shamir, and Adleman 1978).

- Security of RSA - no methods are known to efficiently find the prime factors of large numbers.
- It can be shown that each integer can be written as the product of prime numbers.
- For example, 2100 can be written as

  2100 = 2 x 2 x 3 x 5 x 5 x 7

  making 2, 3, 5, and 7 the prime factors in 2100.
- Private and public keys are constructed from very large prime numbers (consisting of hundreds of decimal digits).
- Breaking RSA is equivalent to finding those two prime numbers.
- So far, this has shown to be computationally infeasible despite mathematicians working on the problem for centuries.

Generating the private and public key requires four steps:

1. Choose two very large prime numbers, p and q.
2. Compute n = p x q and z = (p – 1) x (q – 1).
3. Choose a number d that is relatively prime to z.
4. Compute the number e such that e x d = 1 mod z.
   - One of the numbers, say d, can subsequently be used for decryption, whereas e is used for encryption.
   - Only one of these two is made public, depending on what the algorithm is being used for.

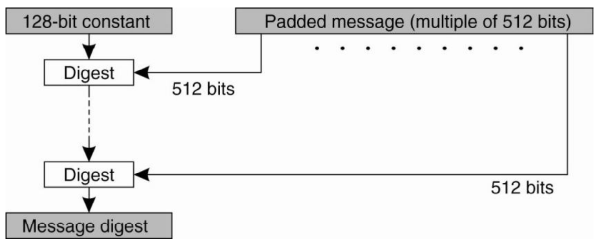
Example 3: Bob and Alice

- Alice wants to keep the messages she sends to Bob confidential.
- She wants to ensure that no one but Bob can intercept and read her messages to him.
- RSA considers each message m to be just a string of bits.
- Each message is first divided into fixed-length blocks, where each block m_i, interpreted as a binary number that should lie in the interval $0 \leq mi < n$.
- To encrypt message m, the sender calculates for each block m_i the value $c_i = m_i^e \ (mod\ n)$, which is then sent to the receiver.
- Decryption at the receiver's side takes place by computing $m_i = c_i^d \ (mod\ n)$.
- **Note** for the encryption, both e and n are needed, whereas decryption requires knowing the values d and n.
- RSA has the drawback of being computationally more complex.
- encrypting messages using RSA is approximately 100–1000 times slower than DES, depending on the implementation technique used.
- Many cryptographic systems use RSA to exchange only shared keys in a secure way, but much less for actually encrypting "normal" data.

Hash Functions: MD5

- MD5 is a hash function for computing a 128-bit, fixed length message digest from an arbitrary length binary input string.
- The input string is first padded to a total length of 448 bits (modulo 512), after which the length of the original bit string is added as a 64-bit integer.
- In effect, the input is converted to a series of 512-bit blocks.

The structure of MD5 algorithm.



- Starting with some constant 128-bit value, the algorithm proceeds in k phases, where k is the number of 512-bit blocks comprising the padded message.
- During each phase, a 128-bit digest is computed out of a 512-bit block of data coming from the padded message, and the 128-bit digest computed in the preceding phase.

A phase in MD5 consists of four rounds of computations, where each round uses one of the following four functions:

F (x,y,z) = (x AND y) OR ((NOT x) AND z)

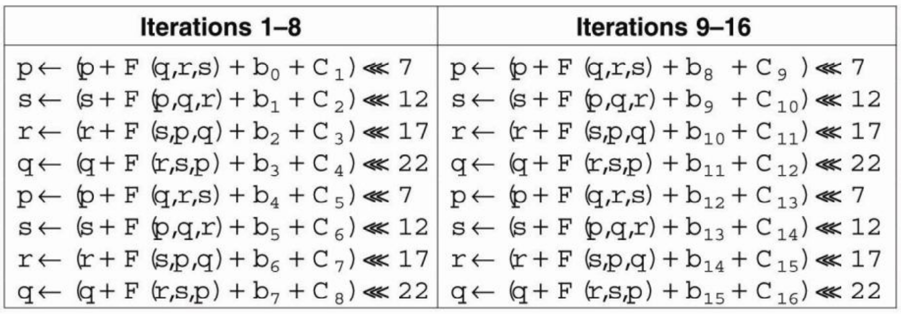
G (x,y,z) = (x AND z) OR (y AND (NOT z))

H (x,y,z) = x XOR y XOR z

I (x,y,z) = y XOR (x OR (NOT z))

- Each of these functions operates on 32-bit variables x, y, and z.
- Consider a 512-bit block b from the padded message that is being processed during phase k.
- Block b is divided into 16 32-bit subblocks b0,b1,...,b15.
- During the first round, function F is used to change four variables (denoted as p, q, r, and s, respectively) in 16 iterations
- These variables are carried to each next round, and after a phase has finished, passed on to the next phase.
- There are a total of 64 predefined constants Ci.
- The notation x < < < n is used to denote a left rotate: the bits in x are shifted n positions to the left, where the bit shifted off the left is placed in the rightmost position.

The 16 iterations during the first round in a phase in MD5.



Iterations 1–8	Iterations 9–16
$p \leftarrow (p + F(q,r,s) + b_0 + C_1) \lll 7$	$p \leftarrow (p + F(q,r,s) + b_8 + C_9) \lll 7$
$s \leftarrow (s + F(p,q,r) + b_1 + C_2) \lll 12$	$s \leftarrow (s + F(p,q,r) + b_9 + C_{10}) \lll 12$
$r \leftarrow (r + F(s,p,q) + b_2 + C_3) \lll 17$	$r \leftarrow (r + F(s,p,q) + b_{10} + C_{11}) \lll 17$
$q \leftarrow (q + F(r,s,p) + b_3 + C_4) \lll 22$	$q \leftarrow (q + F(r,s,p) + b_{11} + C_{12}) \lll 22$
$p \leftarrow (p + F(q,r,s) + b_4 + C_5) \lll 7$	$p \leftarrow (p + F(q,r,s) + b_{12} + C_{13}) \lll 7$
$s \leftarrow (s + F(p,q,r) + b_5 + C_6) \lll 12$	$s \leftarrow (s + F(p,q,r) + b_{13} + C_{14}) \lll 12$
$r \leftarrow (r + F(s,p,q) + b_6 + C_7) \lll 17$	$r \leftarrow (r + F(s,p,q) + b_{14} + C_{15}) \lll 17$
$q \leftarrow (q + F(r,s,p) + b_7 + C_8) \lll 22$	$q \leftarrow (q + F(r,s,p) + b_{15} + C_{16}) \lll 22$

The second round uses the function G in a similar fashion, whereas H and I are used in the third and fourth round, respectively.

Each step thus consists of 64 iterations, after which the next phase is started, but now with the values that p, q, r, and s have at that point.

3.3.2 Secure Channels

Q24. What are secure channels? Write about them.

Ans :

Secure Channels

Secure communications between parties. They are.Authorized

Secure channels protect against (protected by):

- ➢ **Interception** (confidentiality).
- ➢ **Modification** (auth. and integrity).
- ➢ **Fabrication** (auth. and integrity).

Applications of Cryptography

1. Authentication.
2. Message Integrity.
3. Confidentiality.

- ➢ Common practice to use secret-key cryptography by means of session keys.
- ➢ A session key is a shared (secret) key that is used to encrypt messages for integrity and possibly also confidentiality.
- ➢ This key is used only for as long as the channel exists.
- ➢ When the channel is closed, its associated session key is securely destroyed.

  Authentication Based on a Shared Secret Key

- ➢ Ensure data message integrity exchanged after authentication use secret-key cryptography with session keys.
- ➢ Session key - is a shared (secret) key that is used to encrypt messages for integrity
- ➢ Generally used only for as long as the channel exists.

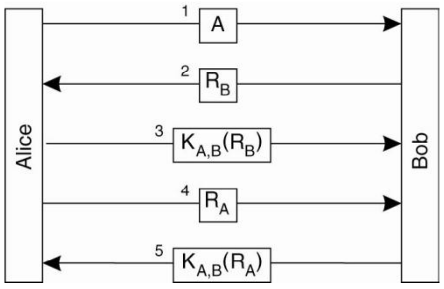
Example : Bob and Alice Again

- ➢ Alice and Bob are abbreviated by A and B, respectively, and their shared key is denoted as $\mathbf{K}_{A,B}$.
- ➢ One party challenges the other to a response that can be correct only if the other knows the shared secret key.
- ➢ also known as challenge-response protocols

1. Alice sends her identity to Bob (message 1), indicating that she wants to set up a communication channel between the two.
2. Bob sends a challenge R_B to Alice, shown as message 2.
   - ➢ Such a challenge could take the form of a random number.
   - ➢ Alice must encrypt the challenge with the secret key $\mathbf{K}_{A,B}$. that she shares with Bob, and return the encrypted challenge to Bob. This response is shown as message 3 containing $\mathbf{K}_{A,B}.(R_B)$.

Authentication based on a shared secret key, using a 'challenge response' protocol.

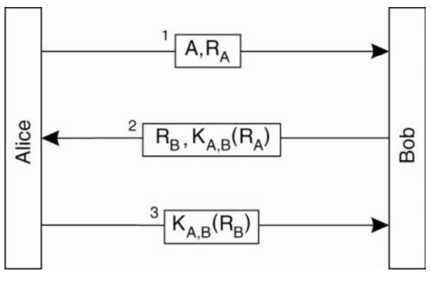
Note: R is a random number.



3. When Bob receives the response $\mathbf{K}_{A,B}.(R_B)$ to his challenge R_B, he can decrypt the message using the shared key again to see if it contains R_B.
   - ➢ If so, he then knows that Alice is on the other side, for who else could have encrypted R_B with $\mathbf{K}_{A,B}$ in the first place?
4. Alice has not yet verified that it is Bob is on the other side of the channel.
   - ➢ She sends a challenge R_A (message 4), which Bob responds to by returning $\mathbf{K}_{A,B}.(R_A)$, shown as message 5.
5. When Alice decrypts it with KA,B and sees her RA, she knows she is talking to Bob.

Example: Bob and Alice – Taking Shortcuts

Optimization of the Authentication based on a shared secret key, but using three instead of five messages.



Idea:

- If Alice eventually wants to challenge Bob anyway, she might as well send a challenge along with her identity when setting up the channel.
- Bob returns his response to that challenge, along with his own challenge in a single message

This protocol can easily be defeated by a reflection attack.

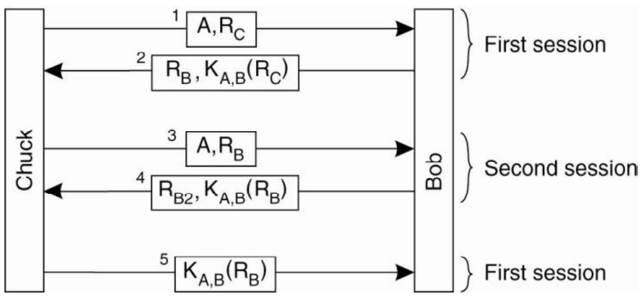
A **reflection attack** is a potential way of attacking a challenge-response authentication system which uses the same protocol in both directions.

The basic idea is to trick the target into providing the answer to its own challenge.

The general attack outline is as follows:

1. The attacker initiates a connection to a target.
2. The target attempts to authenticate the attacker by sending it a challenge.
3. The attacker opens another connection to the target, and sends the target this challenge as its own.
4. The target responds to that challenge.
5. The attacker sends that response back to the target ("reflects" it) on the first connection.

The reflection attack.



5. Bob not recognizing that he, himself, had used R_B before as a challenge, responds with $\mathbf{K}_{A,B}(R_B)$ and another challenge R_{B2}, shown as message 4.
6. At that point, Chuck has $\mathbf{K}_{A,B}(R_B)$ and finishes setting up the first session by returning message 5 containing the response $\mathbf{K}_{A,B}(R_B)$, which was originally requested from the challenge sent in message 2.

Mistake 1: the two parties in the new version of the protocol were using the same challenge in two different runs of the protocol.

Better Design: always use different challenges for the initiator and for the responder.

- In general, letting the two parties setting up a secure channel do a number of things identically is not a good idea.

Mistake 2: Bob gave away valuable information in the form of the response $\mathbf{K}_{A,B}(R_C)$ without knowing for sure to whom he was giving it.

- Not violated in the original protocol, in which Alice first needed to prove her identity, after which Bob was willing to pass her encrypted information.

Authentication Using a Key Distribution Center

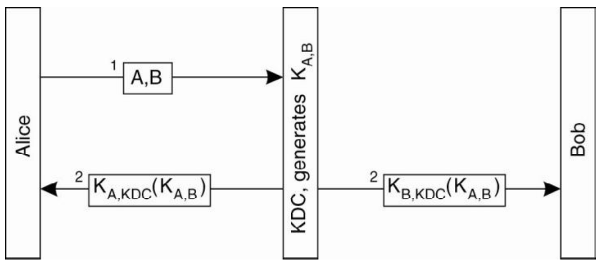
Problem with a shared secret key for authentication: **scalability**.

- Given N hosts - each host is required to share a secret key with each of the other **N-1** hosts
- System as a whole must manage **N (N - 1)/2** keyes
- Each host has to manage N - 1 keys.

Alternative: use a centralized approach by means of a Key Distribution Center (KDC).

- This KDC shares a secret key with each of the hosts
- No pair of hosts is required to have a shared secret key as well.
- Using a KDC requires that we manage N keys instead of N (N - 1)/2
- If Alice wants to set up a secure channel with Bob, she can do so with the help of a (trusted) KDC.
- The KDC hands out a key to both Alice and Bob that they can use for communication,

Principle of using a KDC



1. Alice sends a message to the KDC, telling it that she wants to talk to Bob.
2. The KDC returns a message containing a shared secret key $K_{A,B}$ that she can use.
   - The message is encrypted with the secret key $K_{A,KDC}$ that Alice shares with the KDC.
3. The KDC sends $K_{A,B}$ to Bob, but now encrypted with the secret key $K_{B,KDC}$ it shares with Bob.

Drawbacks :

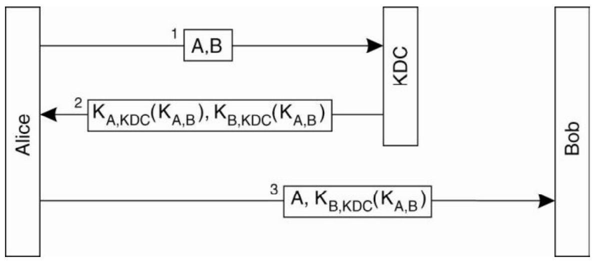
- Alice may want to set up a secure channel with Bob even before Bob had received the shared key from the KDC.
- The KDC is required to pass Bob the key.

Solution: ticket

- KDC passes $K_{B,KDC}(K_{A,B})$ back to Alice and lets her connect to Bob.
- The message $K_{B,KDC}(K_{A,B})$ is also known as a ticket.
- It is Alice's job to pass this ticket to Bob.

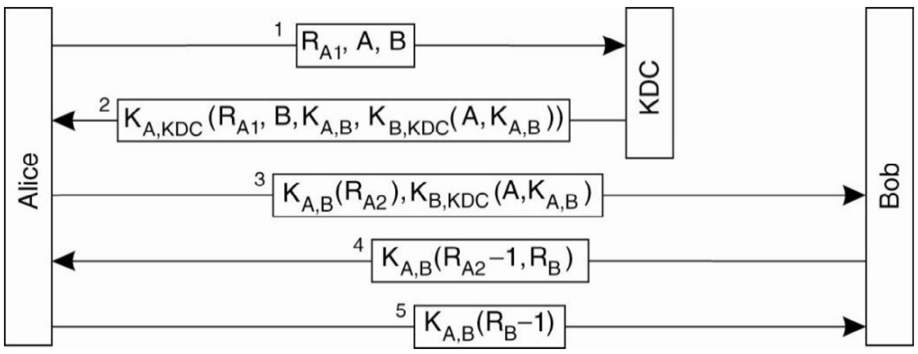
Note that Bob is still the only one that can make sensible use of the ticket, as he is the only one besides the KDC who knows how to decrypt the information it contains.

Using a ticket and letting Alice set up a connection to Bob.


Alice
KDC
Bob
1 A,B
2 $K_{A,KDC}(K_{A,B}), K_{B,KDC}(K_{A,B})$
3 A, $K_{B,KDC}(K_{A,B})$

- Protocol is a variant of a well-known example of an authentication protocol using a KDC, known as the Needham-Schroeder authentication protocol, named after its inventors (Needham and Schroeder, 1978).

The Needham-Schroeder protocol is a multiway challenge-response protocol and works as follows.


Alice
KDC
Bob
1 R_{A1}, A, B
2 $K_{A,KDC}(R_{A1}, B, K_{A,B}, K_{B,KDC}(A, K_{A,B}))$
3 $K_{A,B}(R_{A2}), K_{B,KDC}(A, K_{A,B})$
4 $K_{A,B}(R_{A2}-1, R_B)$
5 $K_{A,B}(R_B-1)$

1. When Alice wants to set up a secure channel with Bob, she sends a request to the KDC containing a challenge R_A, along with her identity A and that of Bob.
2. The KDC responds by giving her the ticket $\mathbf{K_{B,KDC}(K_{A,B})}$, along with the secret key $\mathbf{K_{A,B}}$ that she can subsequently share with Bob.
   - The challenge R_{A1} that Alice sends to the KDC along with her request to set up a channel to Bob is also known as a nonce.
   - A nonce is a random number that is used only once, such as one chosen from a very large set.
   - Purpose of a nonce is to uniquely relate two messages to each other
   - e.g. message 1 and message 2.
   - by including R_{A1} again in message 2, Alice will know for sure that message 2 is sent as a response to message 1, and that it is not a replay of an older message.
   - Message 2 also contains B, the identity of Bob.
   - By including B, the KDC protects Alice against the following attack.
3. After the KDC has passed the ticket to Alice, the secure channel between Alice and Bob can be set up.
   - Alice sends message 3, which contains the ticket to Bob, and a challenge R_{A2} encrypted with the shared key $\mathbf{K_{A,B}}$ that the KDC had just generated.

4. Bob then decrypts the ticket to find the shared key, and returns a response R_{A2} - 1 along with a challenge R_B for Alice.

   ➢ This ties message 4 to message 3 in the same way that the nonce R_A tied message 2 to message 1.

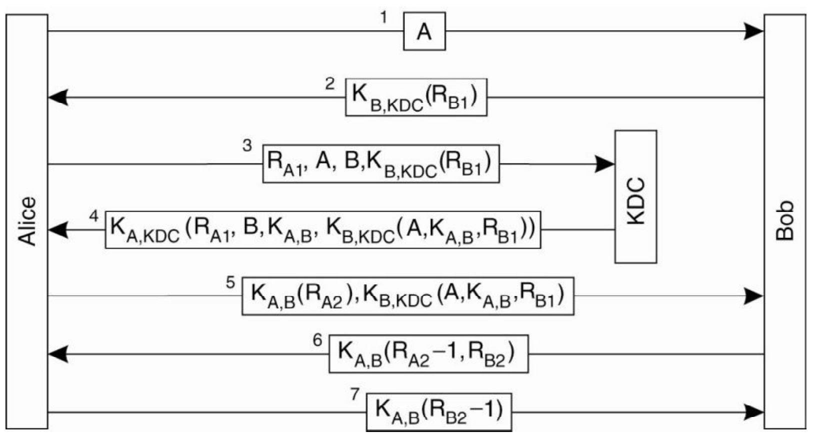
Weak Point

➢ If Chuck got an old key $K_{A,B}$, he could replay message 3 and get Bob to set up a channel.

➢ Bob will then believe he is talking to Alice, while, in fact, Chuck is at the other end.

➢ Need to relate message 3 to message 1 - make the key dependent on the initial request from Alice to set up a channel with Bob.

Protection against malicious reuse of a previously generated session key in the Needham-Schroeder protocol.


Alice
Bob
KDC
1 A
2 $K_{B,KDC}(R_{B1})$
3 R_{A1}, A, B, $K_{B,KDC}(R_{B1})$
4 $K_{A,KDC}(R_{A1}, B, K_{A,B}, K_{B,KDC}(A, K_{A,B}, R_{B1}))$
5 $K_{A,B}(R_{A2}), K_{B,KDC}(A, K_{A,B}, R_{B1})$
6 $K_{A,B}(R_{A2}-1, R_{B2})$
7 $K_{A,B}(R_{B2}-1)$

Solution: incorporate a nonce in the request sent by Alice to the KDC.

➢ The nonce has to come from Bob: this assures Bob that whoever wants to set up a secure channel with him, will have gotten the appropriate information from the KDC.

➢ Alice first requests Bob to send her a nonce R_{B1}, encrypted with the key shared between Bob and the KDC.

➢ Alice incorporates this nonce in her request to the KDC, which will then decrypt it and put the result in the generated ticket.

➢ Bob will know for sure that the session key is tied to the original request from Alice to talk to Bob.

Authentication Using Public-Key Cryptography

Mutual authentication in a public-key cryptosystem.

➢ Note that the KDC is missing ...

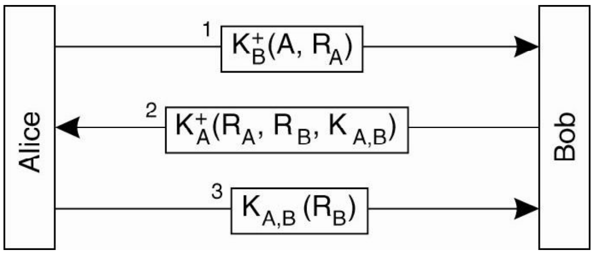
➢ But, this assumes that some mechanism exists to verify everyone's public key.

Example: Bob and Alice

➢ Alice wants to set up a secure channel to Bob

➢ Both are in the possession of each other's public key.

Mutual authentication in a public-key cryptosystem.



- Alice sends a challenge R_A to Bob encrypted with his public key K_B^+.
- Bob's must decrypt the message and return the challenge to Alice.
- Because Bob is the only person that can decrypt the message (using the private key that is associated with the public key Alice used), Alice will know that she is talking to Bob.
- When Bob receives Alice's request to set up a channel, he returns the decrypted challenge, along with his own challenge RB to authenticate Alice.
- He also generates a session key $\mathbf{K_{A,B}}$ that can be used for further communication.
- Bob's response to Alice's challenge, his own challenge, and the session key are put into a message encrypted with the public key K_A^+ belonging to Alice, shown as message 2.
- Only Alice will be capable of decrypting this message using the private key K_A^+ associated with K_A^+
- Alice, finally, returns her response to Bob's challenge using the session key $\mathbf{K_{A,B}}$ generated by Bob.

More on Secure Channels

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.
- 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.

Solution : Digital Signing

Q25. Write about Digital Signatures

Ans :

Digital Signatures

Digital signing a message using public-key cryptography. This is implemented in the RSA technology. Here the entire document is encrypted/signed - this can sometimes be a costly overkill.

Example

- Bob has sold Alice a collector's item of some phonograph record for $500.
- The whole deal was done through e-mail.
- Alice sends Bob a message confirming that she will buy the record for $500.

Issues

- Alice needs to be assured that Bob will not maliciously change the $500 mentioned in her message into something higher, and claim she promised more than $500.
- Bob needs to be assured that Alice cannot deny ever having sent the message because she had second thoughts.

Solution

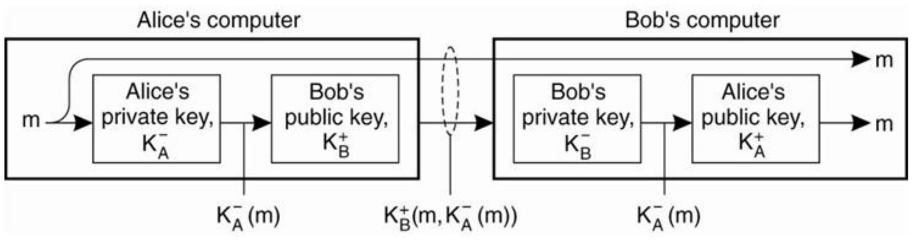
- Alice digitally signs the message in such a way that her signature is uniquely tied to its content.
- Association between a message and its signature prevents that modifications to the message will go unnoticed.
- Alice's signature can be verified to be genuine; she cannot later repudiate the fact that she signed the message.

Ways to Place Digital Signatures

1. Use a public-key cryptosystem such as RSA

   - When Alice sends a message m to Bob, she encrypts it with her private key K_A^-, and sends it off to Bob.
   - If she wants to keep the message content a secret, she can use Bob's public key and sen K_B^+ (m, K_A^- (m)) , which combines m and the version signed by Alice.

   Digital signing a message using public-key cryptography.


Alice's computer
Bob's computer
m
Alice's private key, K_A^-
Bob's public key, K_B^+
Bob's private key, K_B^-
Alice's public key, K_A^+
m
m
$K_A^-(m)$
$K_B^+(m, K_A^-(m))$
$K_A^-(m)$

- Message arrives at Bob => he can decrypt it using Alice's public key.
- If the public key is owned by Alice, then decrypting the signed version of m and successfully comparing it to m can mean only that it came from Alice.
- Alice is protected against any malicious modifications to m by Bob, because Bob will always have to prove that the modified version of m was also signed by Alice.

Problems with Scheme

1. Validity of Alice's signature holds only as long as Alice's private key remains a secret.
2. Alice decides to change her private key.
   - Once Alice has changed her key, her statement sent to Bob becomes worthless.
3. Alice encrypts the entire message with her private key.
   - Such an encryption may be costly in terms of processing requirements
   - A cheaper more elegant scheme is to use a message digest.

   Message digest => is a fixed-length bit string h that has been computed from an arbitrary-length message m by means of a cryptographic hash function H.

   - If m is changed to m', its hash H (m') will be different from h = H (m) so that it can easily be detected that a modification has taken place.

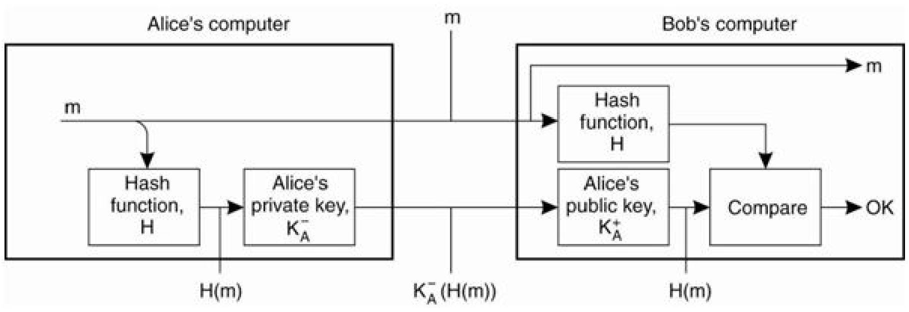
To Digitally Sign a Message

1. Alice first computes a message digest and encrypts the digest with her private key.
2. The encrypted digest is sent along with the message to Bob.

   Note that the message itself is sent as plaintext: everyone is allowed to read it.

   - If confidentiality is required, then the message should also be encrypted with Bob's public key.

   Digitally signing a message using a message digest.



3. When Bob receives the message and its encrypted digest, he decrypts the digest with Alice's public key, and separately calculates the message digest.

   ➢ If the digest calculated from the received message and the decrypted digest match, Bob knows the message has been signed by Alice.

3.3.3 Access Control

Q26. Write about various methods of access control.

Ans :

1. Access Rights

Access rights is referred to as access control, whereas authorization is about granting access rights

Authorization versus Authentication

Authentication: Verify the claim that a subject says it is S: verifying the **identity** of a subject

Authorization: Determining whether a subject is permitted certain services from an object

Note: Authorization makes sense only if the requesting subject has been authenticated

Access Control works mainly on Subjects and objects

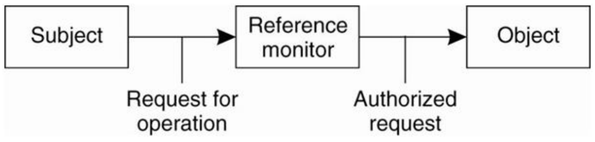
Subjects issue a request to access an object.

➢ 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.

General model of controlling access to objects.



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 (e.g., by the underlying trusted operating system) each time an object is invoked.

2. Access Control Matrix

Maintain an **access control matrix** ACM in which entry ACM[S,O] contains the permissible operations that subject S can perform on object O

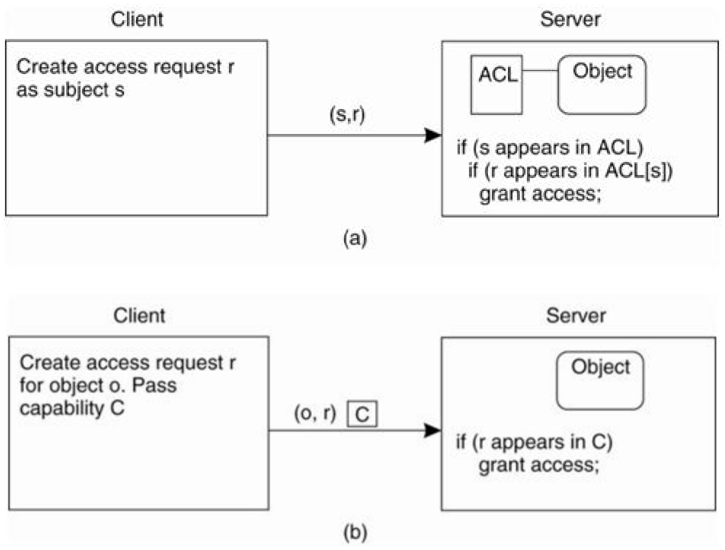
System may need to support thousands of users and millions of objects that require protection. Many entries in the matrix will be empty: a single subject will generally have access to relatively few objects.

Implementation (a): Each object O maintains an **access control list (ACL)**: ACM[*,O] describing the permissible operations per subject (or group of subjects)

Implementation (b): Each subject S has a **capability**: ACM[S,*] describing the permissible operations per object (or category of objects)

Comparison between ACLs and capabilities for protecting objects.

(a) Using an ACL. (b) Using capabilities.

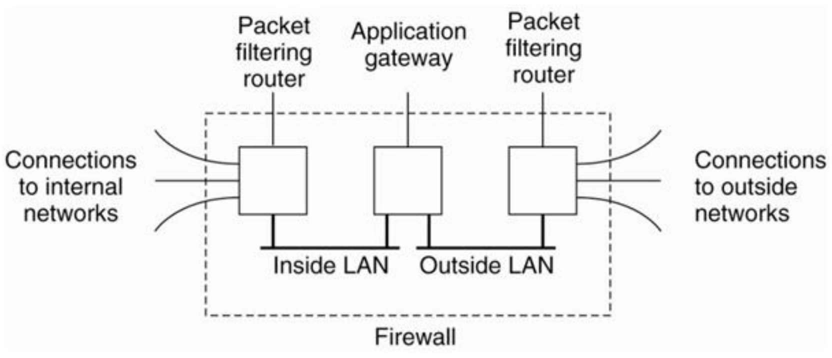


3. Firewalls

Sometimes it's better to select service requests at the lowest level: network packets. Packets that do not fit certain requirements are simply removed from the channel

Protect your company by a firewall: it implements access control

A common implementation of a firewall



Two Types of Firewalls:

Packet-filtering gateway - operates as a router and makes decisions as to whether or not to pass a network packet based on the source and destination address as contained in the packet's header.

- Typically, the packet-filtering gateway shown on the outside LAN above would protect against incoming packets, whereas the one on the inside LAN would filter outgoing packets.

  Application-level gateway - this type of firewall inspects the content of an incoming or outgoing message.
- e.g. mail gateway that discards incoming or outgoing mail exceeding a certain size.
- e.g. filtering spam e-mail.
- e.g. proxy gateway - works as a front end to a specific kind of application, and ensures that only those messages are passed that meet certain criteria.

4. 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.

Protecting an Agent

Etect that an agent has been tampered with while it was on the move.

Most important: append-only logs:

- Data can only be appended to the log; there is no way that data can be removed or modified without the owner being able to detect this
- There is always an associated checksum. Initially,
  - Cinit = K^+owner(N), with N a nonce.
- Adding data X by server S:
  - Cnew = K^+owner(Cold, sig(S,X),S)
- Removing data from the log:
- K^-owner(C) -> Cprev, sig(S,X),S allowing the owner to check integrity of X

Protecting a Host

Enforce a (very strict) single policy, and implement that by means of a few simple mechanisms

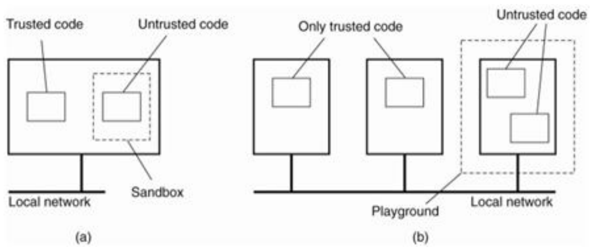
Sandbox model: Policy: Remote code is allowed access to only a pre-defined collection of resources and services.

- Check instructions for illegal memory access and service access

Playground model: Same policy, but mechanism is to run code on separate unprotected machine.

- We need to be able to distinguish local from remote code before being able to do anything
- We need to be able to assign a set of permissions to mobile code before its execution and check operations against those permissions at all times
- We need to be able to assign different sets of permissions to different units of mobile code => authenticate mobile code (e.g. through signatures)

(a) A sandbox. (b) A playground.



5. Denial of Service (DoS)

Maliciously preventing authorized processes from accessing resources.

- Huge collection of processes jointly attempt to bring down a networked service.
- Attackers succeed in hijacking a large group of machines which unknowingly participate in the attack.

- It is distinguish two types of attacks:
  1. those aimed at bandwidth depletion
  2. those aimed at resource depletion.

Solutions:

No single method to protect against DDoS attacks. BUT…

Continuously monitor network traffic

- Starting at the egress routers where packets leave an organization's network.
  - o Experience shows that by dropping packets whose source address does not belong to the organization's network we can prevent a lot of havoc.
  - o In general, the more packets can be filtered close to the sources, the better.
- Concentrate on ingress routers, that is, where traffic flows into an organization's network.
  - o detecting an attack at an ingress router is too late as the network will probably already be unreachable for regular traffic.
  - o Better to have routers further in the Internet, such as in the networks of ISPs, start dropping packets when they suspect that an attack is going on.

3.3.4 Security Management

Q27. Explain about security management in distributed systems.

Ans :

Security Management has 3 main categories:

- Key establishment and distribution
- Secure group management
- Authorization management

Key Establishment : Diffie-Hellman

We can construct secret keys in a safe way without having to trust a third party (i.e. a key distribution center (KDC)):

- Alice and Bob have to agree on two large numbers, n and g. Both numbers may be public.
- Alice chooses large number x, and keeps it to herself. Bob does the same, say y.

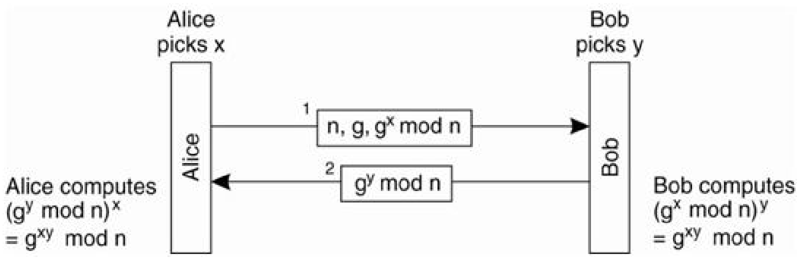
1: Alice sends (n, g, g^x mod n) to Bob

2: Bob sends (g^y mod n) to Alice

3: Alice computes $K_{A,B} = (g^y \bmod n)^x = g^{xy} \bmod n$

4: Bob computes $K_{A,B} = (g^x \bmod n)^y = g^{xy} \bmod n$

The principle of Diffie-Hellman key exchange


Alice picks x
Bob picks y
Alice
Bob
1
n, g, g^x mod n
2
g^y mod n
Alice computes (g^y mod n)^x = g^{xy} mod n
Bob computes (g^x mod n)^y = g^{xy} mod n

Key Distribution

If authentication is based on cryptographic protocols, and we need session keys to establish secure channels, who's responsible for handing out keys?

Secret Keys

Alice and Bob will have to get a shared key.

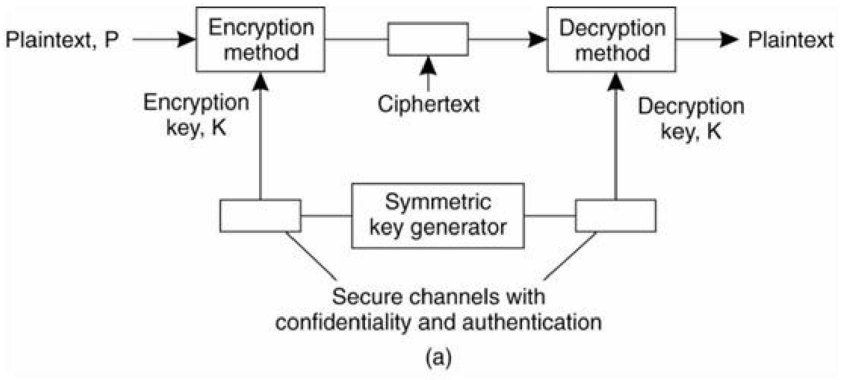
- They can invent their own and use it for data exchange.
- Alternatively, they can trust a **key distribution center** (KDC) and ask it for a key.

Public Keys

- Alice will need Bob's public key to decrypt (signed) messages from Bob, or to send private messages to Bob.
- But she'll have to be sure about actually having Bob's public key, or she may be in big trouble.
- Use a trusted **certification authority** (CA) to hand out public keys.
- A public key is put in a **certificate**, signed by a CA.

(a) Secret-key distribution.

(b) Public-key distribution.



Public-key cryptosystem (b), need to distribute the public key in such a way that the receivers can be sure that the key is paired to a claimed private key.

- although the public key itself may be sent as plaintext, it is necessary that the channel through which it is sent can provide authentication.
- The private key, needs to be sent across a secure channel providing authentication as well as confidentiality.

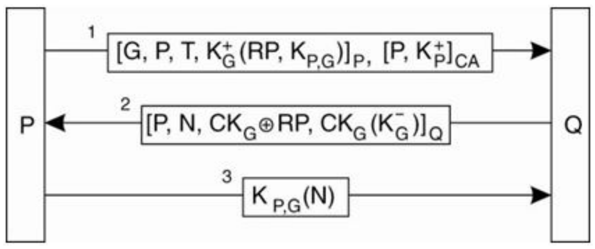
Secure Group Management

Group uses a key pair (K^+_G, K^-_G) for communication with nongroup members.

- There is a separate shared secret key CK_G for internal communication.
- 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 (notation: $[JR]_P$), along with a certificate containing its public key K^+_P .

Securely admitting a new group member.



P
Q
1
$[G, P, T, K^+_G(RP, K_{P,G})]_P, [P, K^+_P]_{CA}$
2
$[P, N, CK_G \oplus RP, CK_G(K^-_G)]_Q$
3
$K_{P,G}(N)$

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.

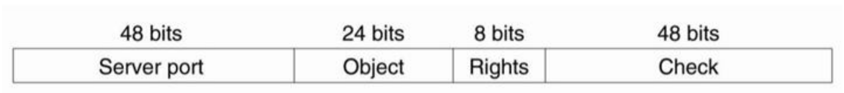
Authorization Management

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.
- 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.
  - o Amoeba uses broadcasting to locate the machine where the server is currently located.

A capability in Amoeba.

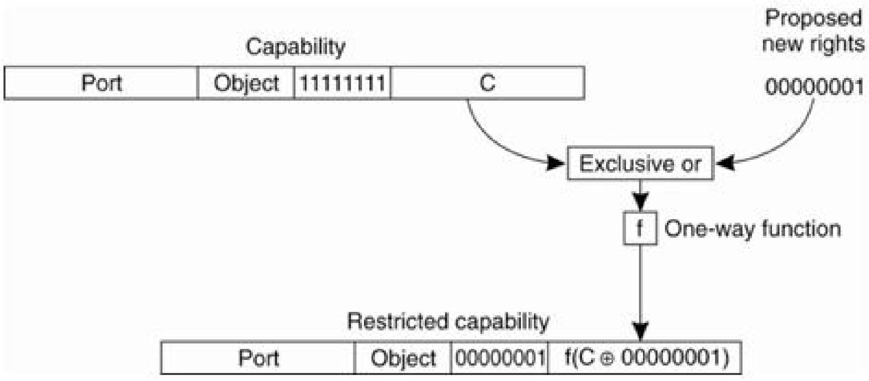


48 bits	24 bits	8 bits	48 bits
Server port	Object	Rights	Check

- Next 24 bits are used to identify the object at the given server.
- Note that the server port, along with the object identifier, form a 72-bit system wide unique identifier for every object in Amoeba.
- 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, as we explain in the following pages.

- 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.

Generation of a restricted capability from an owner capability



UNIT IV

Distributed Object - Based Systems: architecture, processes, communication, naming, synchronization, consistency and replication, fault tolerance, security. Distributed File Systems: architecture, process, communication, naming, synchronization, consistency and replication, fault tolerance, security. Distributed Webbased Systems:architecture, process, communication, naming, synchronization, consistency and replication, fault tolerance, security.

4.1 Distributed Object - Based Systems

4.1.1 Architecture

Q1. Explain about distributed objects.

Ans :

Remote Distributed Objects

The key feature of an object is that it encapsulates data, called the state, and the operations on those data, called the methods. Methods are made available through an interface..

This separation between interfaces and the objects implementing these interfaces is crucial for distributed systems.

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.

The actual object resides at a server machine, where it offers the same interface as it does on the client machine.

The server-side stub is often referred to as a skeleton as it provides the bare means for letting the server middleware access the user-defined objects.

The server stub is also responsible for marshaling replies and forwarding reply messages to the client-side proxy.

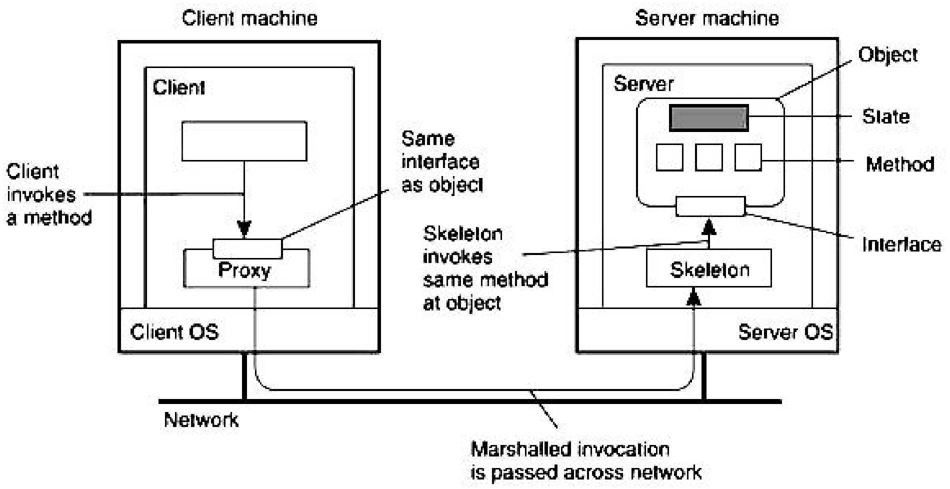


Fig.: Common Organization of a Remote Object with Client-side Proxy

Types of Objects I

➢ **Compile-time objects:** Language-level objects, from which proxy and skeletons are automatically generated.

➢ **Runtime objects:** Can be implemented in any language, but require use of an object adapter that makes the implementation appear as an object.

Types of objects II

➢ Transient objects: live only by virtue of a server: if the server exits, so will the object.

➢ Persistent objects: live independently from a server: if a server exits, the object's state and code remain (passively) on disk.

Example: Enterprise Java Beans (EJB)

EJB is a Java object hosted by special server that allows for different means of calling the object by remote clients.The following is the architecture of EJB.

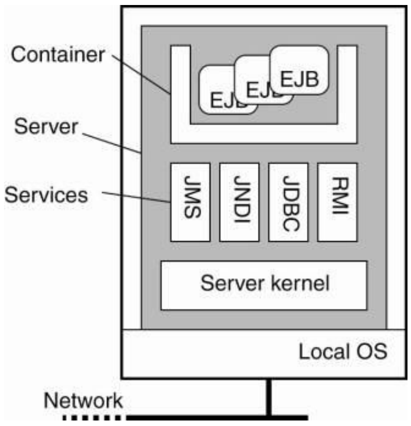


Fig.: General architecture of an EJB server

Types of EJBs

Four Different Types

➢ **Stateless Session Bean:** Transient object, called once, does its work and is done.

Example: execute an SQL query and return result to caller.

➢ Stateful session bean: Transient object, but maintains client-related state untilthe end of a session. Example: shopping cart.

➢ **Entity bean:** Persistent, statefulobject, can be invoked during different sessions.

Example: object maintaining client info on last number of sessions.

➢ **Message-driven bean:** Reactive objects, often triggered by message types. Usedto implement publish/subscribe forms of communication.

Globe distributed objects

Most distributed objects are not distributed at all: state is kept at a single node.Globe objects, which are physically distributed across multiple machines.

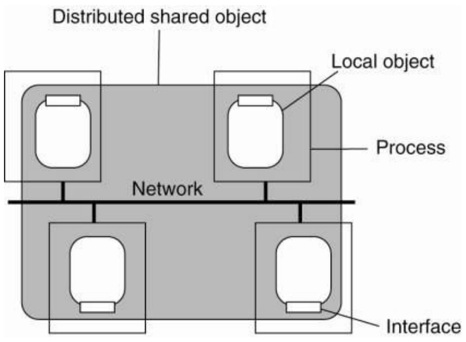


Fig.: The organization of a Globe distributed shared object

To make DSOs generic, we need to separate function from distribution support.

Globe local objects come in two flavors.

➢ A primitive local object is a local object that does not contain any other local object.

➢ A 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.

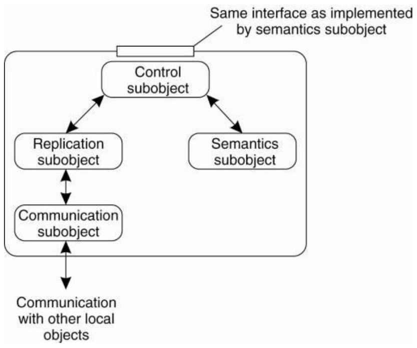

Same interface as implemented by semantics subobject
Control subobject
Replication subobject
Semantics subobject
Communication subobject
Communication with other local objects

Fig.: The general organization of a local object for distributed shared objects in Globe

This local object is shown in Figure and consists of at least four subobjects.

- The semantics subobject implements the functionality provided by a distributed shared object. In essence, it corresponds to ordinary remote objects, similar in flavor to EJBs.
- The communication subobject is used to provide a standard interface to the underlying network. This subobject offers a number of message-passing primitives for connection-oriented as well as connectionless communication.
- Crucial to virtually all distributed shared objects is the replication subobject. This subobject implements the actual distribution strategy for an object.. The replication subobject is responsible for deciding exactly when a method as provided by the semantics subobject is to be carried out.
- The control subobject is used as an intermediate between the user-defined interfaces of the semantics subobject and the standardized interfaces of the replication subobject.

4.1.2 Processes

Q2. Write about processes or object Servers.

Ans :

- 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.

Whether or not these parts are separated, or whether method implementations are shared by multiple objects, depends on the object server.

Alternatives for Invoking Objects

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.

A simple approach is to assume that all objects look alike and that there is only one way to invoke an object.

A transient object is an object that exists only as long as its server exists, but possibly for a shorter period of time.

The advantage of this approach is that a transient object will need a server's resources only as long as the object is really needed.

The drawback is that an invocation may take some time to complete, because the object needs to be created first.

Object Adapter

An object adapter has one or more objects under its control. Because a server should be capable of simultaneously supporting objects that require different activation policies, several object adapters may reside in the same server at the same time. When an invocation request is delivered to the server, that request is first dispatched to the appropriate object adapter, as shown in Fig.

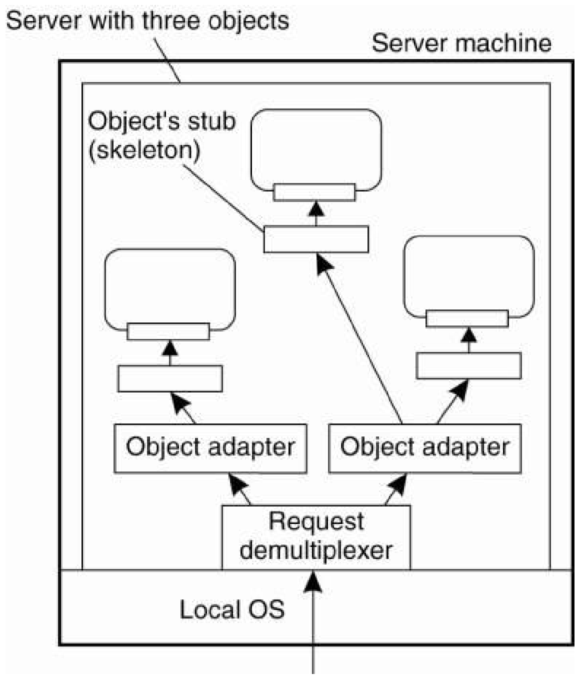


Fig.: Organization of an Object Server Supporting Different Activation Policies

The object adapters are unaware of the specific interfaces of the objects they control. Otherwise, they could never be generic.

The object adapter is that it can extract an object reference from an invocation request, and subsequently dispatch the request to the referenced object, but now following a specific activation policy.

An object adapter can support different activation policies by simply configuring it at runtime.

For example: CORBA-compliant systems

4.1.3 Communication

Q3. Explain, how to bind a client to an object.

Ans :

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, implementing an interface containing the methods the process can invoke. In many cases, binding is done automatically.

Two ways of binding:

- **Implicit:** Invoke methods directly on the referenced object.
- **Explicit:** Client must first explicitly bind to object before invoking it.

For example, C++ allows overloading the unary member selection operator ("$\rightarrow$").

Implementation of Object References

A simple object reference would include the network address of the machine where the actual object resides, along with an end point identifying the server that manages the object, plus an indication of which object.

First, if the server's machine crashes and the server is assigned a different end point after recovery, all object references have become invalid.

Static versus Dynamic Remote Method Invocations

After a client is bound to an object, it can invoke the object's methods through the proxy. Such a remote method invocation, or simply RMI, is very similar to an RPC when it comes to issues such as marshaling and parameter passing.

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.

Dynamic invocation isable to compose a method invocation at runtime, also referred to as a dynamic invocation. The essential difference with static invocation is that an application selects at runtime which method it will invoke at a remote object. Dynamic invocation generally takes a form such as

invoke(object, method, input_parameters, output_parameters);

Parameter Passing

When invoking a method with an object reference as parameter, that reference is copied and passed as a value parameter only when it refers to a remote object. In this case, the object is literally passed by reference.

These two situations are illustrated in Fig. which shows a client program running on machine A, and a server program on machine C. The client has a reference to a local object O 1 that it uses as a parameter when calling the server program on machine C. In addition, it holds a reference to a remote object O 2 residing at machine B, which is also used as a parameter. When calling the server, a copy of O 1 is passed to the server on machine C, along with only a copy of the reference to O 2.

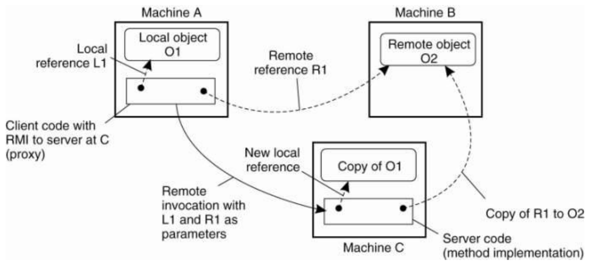


Fig.: The situation when passing an object by reference or by value

4.1.4 Naming

Q4. Explain RMI and its Architecture.

Ans :

Remote Method Invocation (RMI)

- The Java Remote Method Invocation (RMI) system allows an object running in one Java Virtual Machine (VM) to invoke methods on an object running in another Java VM.

- Java RMI provides applications with transparent and lightweight access to remote objects. RMI defines a high-level protocol and API.
- Programming distributed applications in Java RMI is simple.

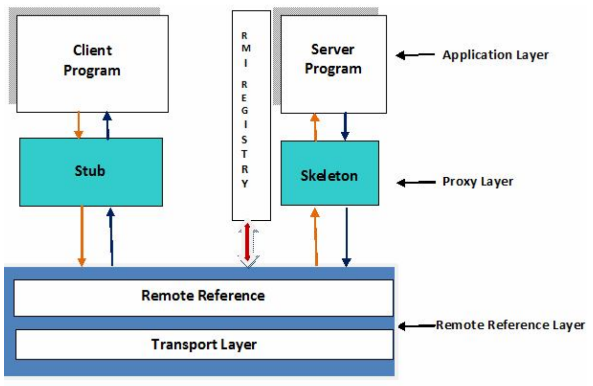
  It is a single-language system.

The programmer of a remote object must consider its behavior in a concurrent environment.

RMI Architecture

In RMI, the client and server do not communicate directly; instead communicates through stub and skeleton (a special concept of RMI and this is how designers achieved distributed computing in Java). They are nothing but special programs generated by RMI compiler. They can be treated as proxies for the client and server. Stub program resides on client side and skeleton program resides on server. That is, the client sends a method call with appropriate parameters to stub. The stub in turn calls the skeleton on the server. The skeleton passes the stub request to the server to execute the remote method. The return value of the method is sent to the skeleton by the server. The skeleton, as you expect, sends back to the stub. Finally the return value reaches client through stub.

As the figure illustrates, there comes three layers in the RMI Architecture – Application layer, Proxy layer and Remote reference layer (Transport layer is part of Remote reference layer).



1. Application Layer

This layer is nothing but the actual systems (client and server) involved in communication. A client Java program communicates with the other Java program on the server side. RMI is nothing but a communication between two JVMs placed on different systems.

2. Proxy Layer

The proxy layer consists of proxies (named as stub and skeleton by designers) for client and server. Stub is client side proxy and Skeleton is server side proxy. Stub and skeleton, as many people confuse, are not separate systems but they are after all programs placed on client and server. The stub and skeleton are not hard coded by the Programmer but they are generated by RMI compiler (this is shown in execution part later). Stub is placed on client side and skeleton is placed on server side. The client server communication goes through these proxies. Client sends its request of method invocation (to be executed on remote server) to stub. Stub inturn sends the request to skeleton. Skeleton passes the request to the server program. Server executes the method and sends the return value to the skeleton (to route to client). Skeleton sends to stub and stub to client program.

Marshaling and Unmarshaling

One more job of proxies is marshaling and unmarshaling. Marshaling is nothing but converting data into a special format suitable to pass through the distributed environment without loosing object persistence. For this reason, the RMI mechanism implicitly serializethe objects involved in communication. The stub marshals the data of client and then sends to the skeleton. As the format is not understood by the server program, it is unmarshaled by the skeleton into the original format and passed to server. Similarly from server to client also.

A marshal stream includes a stream of objects that are used to transport parameters, exceptions, and errors needed for these streams for communicating with each other. Marshaling and unmarshaling are done by the RMI runtime mechanism implicitly without any programmers extra coding.

3. Remote Reference Layer

Proxies are implicitly connected to RMI mechanism through Remote reference layer, the layer responsible for object communication and transfer of objects between client and server. It is responsible for dealing with semantics of remote invocations and implementation – specific tasks with remote objects. In this layer, actual implementation of communication protocols is handled.

Transport layer does not exist separately but is a part of Remote reference layer. Transport layer is responsible for actually setting up connections and handling the transport of data from one machine to another. It can be modified to handle encrypted streams, compression algorithms and a number of other security/performance related enhancements.

The marshaled stream is passed to RRL (Remote Reference Layer). Task of RRL is to identify the host, that is, remote machine. The marshaled stream is passed to Transport layer which performs routing.

Java RMI Applications

A Java RMI application needs to do the following:

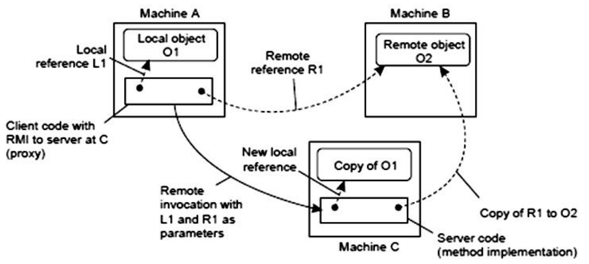
- Locate remote objects: Applications can use one of two mechanisms to obtainreferences to remote objects:
- An application can register its remote objects with RMI's simple namingfacility, the rmiregistry, orthe application can pass and return remote object references as part of itsnormal operation.
- Communicate with remote objects: Details of communication between remoteobjects are handled by RMI; to the programmer, remote communication lookslike a standard Java method invocation.
- Load class bytecodes for objects that are passed around: Because RMI allows acaller to pass objects to remote objects, RMI provides the necessary mechanismsfor loading an object's code, as well as for transmitting its data.
- RMI is supported by two java packages, java.rmi and java.rmi.server. An application that uses RMI has 3 components:
  - An interface that declares headers for remote methods;
  - A server class that implements the interface; and one or more clients that call the remote methods.

RMI: Parameter passing

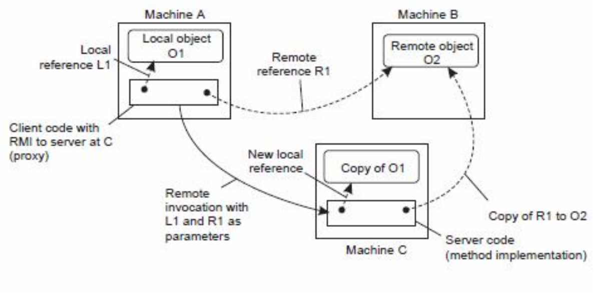
Object Reference

Much easier than in the case of RPC:

- Server can simply bind to referenced object, and invoke methods
- Unbind when referenced object is no longer neededObject-by-value
- A client may also pass a complete object as parameter value:
- An object has to be marshaled:
  - Marshall its state
- Marshall its methods, or give a reference to where an implementation canbe found
- Server unmarshals object. Note that we have now created a copy of the originalobject.
- Object-by-value passing tends to introduce nasty problems.



Systemwide object reference generally contains server address, port to which adapter listens, and local object ID. Extra: Information on protocol between client and server (TCP, UDP, SOAP, etc.).



RMI: Programming Applications

RMI Registry

- The RMI registry is a simple server-side bootstrap naming facility that allowsremote clients to get a reference to a remote object

- Servers name and register their objects to be accessed remotely with the RMIRegistry.
- Clients use the name to find server objects and obtain a remote reference to thoseobjects from the RMI Registry.
- A registry service is a background program that maintains a list of registeredserver names on a host. It is invoked by: rmiregistry port &
- The registry service is provided by a Naming object that provides two key methods:
  - Bind to register a name and server
  - Lookup to retrieve the server bound to a name

Q5. Explain about CORBA.

Ans :

- CORBA, or Common Object Request Broker Architecture, is a standard architecture for distributed object systems. It allows a distributed, heterogeneous collection of objects to interoperate.
- CORBA is a standard defined by the OMG (Object Management Group). It describes an architecture, interfaces, and protocols that distributed objects can use to interact with each other.
- Part of the CORBA standard is the Interface Definition Language (IDL), which is an implementation-independent language for describing the interfaces of remote objects.

The OMG comprises over 700 companies and organizations, including almost all the major vendors and developers of distributed object technology, including platform, database, and application vendors as well as software tool and corporate developers.

- Java IDL is an implementation of the standard IDL-to-Java mapping and is provided by Sun in version 1.3 of Java 2 and is compliant with CORBA 2.x specification. Java IDL provides an Object Request Broker, or ORB.

The ORB is a class library that enables low-level communication between Java-IDL applications and other CORBA-compliant applications.

Like RMI, Java IDL gives you a way to access remote objects over the network. It also provides tools you need to make your objects accessible to other CORBA clients. If you export a Java class using Java IDL, you can create an object from that class and publish it through a naming/directory service.

A remote object can: find this object, call methods on it, and receive data from it, just as if it were running on the client's local machine.

- Unlike RMI, objects that are exported using CORBA can be accessed by clients implemented in any language with an IDL binding (C, C++, Ada etc.).
- The CORBA standard is extensive and provides a rich set of services.

A few of the services are:

Service	Description
Object life cycle	Defines how CORBA objects are created, removed, moved, and copied
Naming Service	The CORBA naming service provides a naming structure for remote objects.
Event Service	Another COS that provides a supplier-consumer communication model that creates asynchronous communication between the objects via an Event Channel.
Persistent object service	
Transactions	Coordinates atomic access to CORBA objects
Concurrency Control	Provides a locking service for CORBA objects in order to ensure concurrent access.

CORBA Products

➢ Several vendors provide CORBA products for various programming languages. The CORBA products that support the Java programming language include:

ORB	Description
The Java 2 ORB	The Java 2 ORB comes with Sun's Java 2. It is missing several features. The main feature it has is the ability to look up an object by name.
Visi Broker for Java	A popular Java ORB from Inprise Corporation. VisiBroker is also embedded in other products. For example, it is the ORB that embedded in the Netscape Communicator browser.
Orbix	A popular Java ORB from Iona Technologies.

CORBA Architecture

The major components that make up the CORBA architecture include the:

➢ Interface Definition Language (IDL), which is how CORBA interfaces are defined,

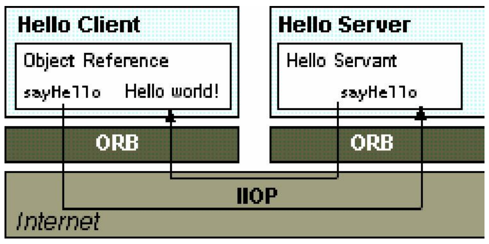
➢ Object Request Broker (ORB), which is responsible for all interactions between remote objects and the applications that use them,

➢ The Portable Object Adaptor (POA), which is responsible for object activation/deactivation, mapping object ids to actual object implementations.

➢ Naming Service, a standard service in CORBA that lets remote clients find remote objects on the networks, and

➢ Inter-ORB Protocol (IIOP).

This figure shows how a one-method distributed object is shared between a CORBA client and server to implement the classic "Hello World" application.



A one-method distributed object shared between a CORBA client and server

Any relationship between distributed objects has two sides: the client and the server. The server provides a remote interface, and the client calls a remote interface. These relationships are common to most distributed object standards, including RMI and CORBA.

The terms client and server define object-level rather than application-level interaction—any application could be a server for some objects and a client of others.

Q6. Explain about Interface Definition Language.

Ans :

Interface Definition Language (IDL)

➢ The services that an object provides are given by its interface. Interfaces are defined in OMG's Interface Definition Language (IDL). IDL is independent of any programming language.

➢ Mappings from IDL to specific programming languages are defined as part of the CORBA specification. Mappings for C, C++, Smalltalk, Ada, COBOL, and Java have been approved by OMG.

➢ The syntax of both Java and IDL were modeled to some extent on C++, so there are a lot of similarities between the two in terms of syntax. However, there are differences between IDL and Java.

In IDL, you declare only the names and types for interfaces, data members, methods, method arguments etc. You do not include the method implementations.

The method implementations are created in implementation language you choose after you've used an IDL compiler (idlj is the IDL compiler for Java) to convert your IDL interface to your target language. When you run the idlj compiler over your interface definition file, it generates the Java version of the interface, as well as the class code files for the stubs and skeletons that enable your applications to hook into the ORB.

IDL includes, like C++, non-class data structure definitions like structs, unions etc. (these are not part of Java).

Method parameters in IDL include modifiers that specify whether they are input, output, or input/output variables. In Java, all primitive data types are passed by value, and all objects are passed by reference.

An IDL file can include multiple public interfaces. Java allows multiple inner classes within a single public class definition and multiple nonpublic classes per file, but only a single public class can be defined in a given Java file.

Modules, which are similar to Java packages, can be nested within other modules in the same IDL file. In Java, you can define a class only within a single package in a single Java file.

Q7. Write a note on ORB.

Ans :

Object Request Broker (ORB)

The core of the CORBA architecture is the ORB. Each machine involved in a CORBA application must have an ORB running in order for processes on that machine to interact with CORBA objects running in remote processes.

➢ Object clients and servers make requests through their ORBs and the remote ORB locates the appropriate object and passes back the object reference to the requestor.

➢ The ORB provides the communication infrastructure needed to identify and locate objects, handles connection management, etc. The ORBs communicate with each other via the IIOP.

Q8. Explain CORBA Naming Services.

Ans :

Naming Service

Defines how CORBA objects can be looked up by a name. It is a Common Object Service (COS) andallows an object to be published using a symbolic name and allows clients to obtain references to the object using a standard API.

The CORBA naming service provides a naming structure for remote objects.

IIOP

- The CORBA standard includes specifications for inter-ORB communication protocols that transmit object requests between various ORBs running on the network.

  The protocols are independent of the particular ORB implementations running at either end. An ORB implemented in Java can talk to an ORB implemented in C, as long as they are both compliant with the CORBA standard. The inter-ORB protocol delivers messages between two ORBs. These messages might be method requests, return values, error messages etc. The inter-ORB protocol (IIOP) also deals with differences between two ORB implementations, like machine-level byte ordering etc. As a CORBA developer, you don't have to be concerned with the low-level communication protocol between ORBs. If you want two ORBs to talk, just make sure they both speak the same inter-ORB protocol (IIOP).

- The Inter-ORB Protocol (IIOP) is an inter-ORB protocol based on TCP/IP and so is extensively used on the Internet.

POA

CORBA Server Object
POA
ORB

The POA connects the server object implementation to the ORB. It extends the functionality of the ORB and some its services include: activation and deactivation of the object implementations, generation and management of object references, mapping of object references to their implementations, and dispatching of client requests to server objects through a skeleton.

4.1.5 Synchronization

Q9. What is Synchronization in Distributed systems.

Ans :

When a process invokes a (remote) object, it has no knowledge whether that invocation will lead to invoking other objects. As a consequence, if an object is protected against concurrent accesses, we may have a cascading set of locks that the invoking process is unaware of, as shown in Figure (a).

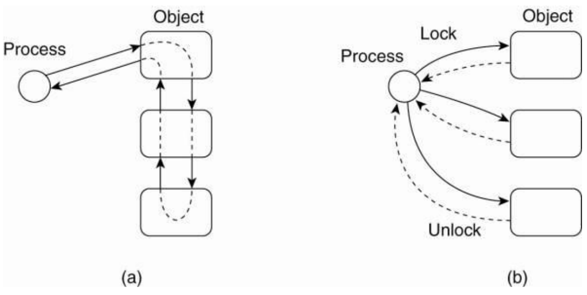


Fig.: Differences in control flow for locking objects

When dealing with data resources such as files or database tables that are protected by locks, the pattern for the control flow is actually visible to the process using those resources, as shown in Figure.

As a consequence, the process can also exert more control at runtime when things go wrong, such as giving up locks when it believes a deadlock has occurred. Note that transaction processing systems generally follow the pattern shown in Figure.

In object-based distributed systems it is therefore important to know where and when synchroni-zation takes place. An obvious location for synchronization is at the object server.

The object server maintain locks complicates matters in the case that invoking clients crash. For this reason, locking can also be done at the client side, an approach that has been adopted in Java. Unfortunately, this scheme has its own drawbacks.

4.1.6 Consistency and Replication

Q10. Write about Entry Consistency in distributed systems.

Ans :

Entry Consistency

Data-centric consistency for distributed objects comes naturally in the form of entry consistency. As objects naturally combine data and the operations on that data, locking objects during an invocation serializes access and keeps them consistent.

There are two issues that need to be solved for implementing entry consistency.

➢ The first one is that we need a means to prevent concurrent execution of multiple invocations on the same object. Simply using local locking mechanisms will ensure this serialization.

➢ The second issue is that in the case of a replicated object, we need to ensure that all changes to the replicated state of the object are the same.

No two independent method invocations take place on different replicas at the same time. This requirement implies that we need to order invocations such that each replica sees all invocations in the same order.

This requirement can generally be met in one of two ways: (1) using a primary-based approach or (2) using totally-ordered multicast to the replicas.

Designing replicated objects is done by first designing a single object, possibly protecting it against concurrent access through local locking, and subsequently replicating it.

If we were to use a primary-based scheme, then additional effort from the application developer is needed to serialize object invocations.

Its implementation may use a primary-based scheme, but it could equally well be based on Lamport clocks.

The problem is one of granularity: although all replicas of an object server may receive invocation requests in the same order, we need to ensure that all threads in those servers process those requests in the correct order as well. The problem is shown in Fig.

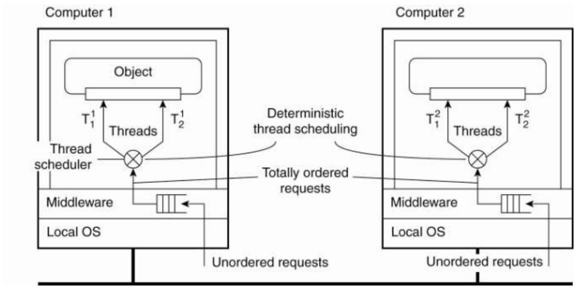


Fig.: Deterministic thread scheduling for replicated object servers

Multithreaded (object) servers simply pick up an incoming request, pass it on to an available thread, and wait for the next request to come in.

The server's thread scheduler subsequently allocates the CPU to runnable threads.

If threads and from Figure handle the same incoming (replicated) invocation request, they should both be scheduled before and , respectively.

One drawback of this scheme is that it operates at the level of the underlying operating system, meaning that every lock needs to be managed. By providing application-level information, a huge improvement in performance can be made by identifying only those locks that are needed for serializing access to replicated objects.

Q11. Explain Replication Framework.

Ans :

In most distributed object-based systems the object technology it is often possible to make a clean separation between devising functionality and handling extra-functional issues such as replication. A powerful mechanism to accomplish this separation is formed by interceptors.

Invocations to objects are intercepted at three different points, as also shown in figure.

➢ At the client side just before the invocation is passed to the stub.

➢ Inside the client's stub, where the interception forms part of the replication algorithm.

➢ At the server side, just before the object is about to be invoked.

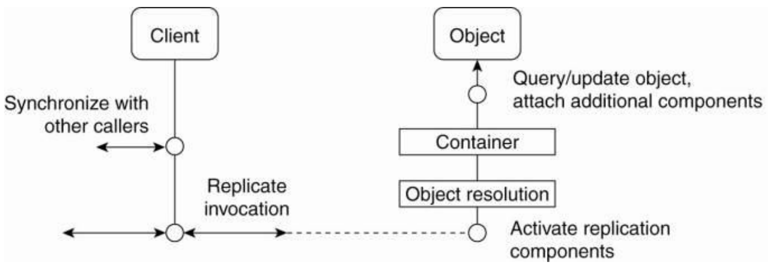

Client
Object
Query/update object, attach additional components
Synchronize with other callers
Container
Replicate invocation
Object resolution
Activate replication components

Fig.: A general framework for separating replication algorithms from objects in an EJB environment

The first interception is needed when it turns out that the caller is replicated. In that case, synchronization with the other callers may be needed as we may be dealing with a replicated invocation.

Invocation can be carried out, the interceptor in the client-side stub can take decisions on where to be forward the request to, or possibly implement a fail-over mechanism when a replica cannot be reached.

Finally, the server-side interceptor handles the invocation. This interceptor is split into two.

1. Just after the request has come in and before it is handed over to an adapter, the replication algorithm gets control. It can then analyze for whom the request is intended allowing it .
2. Just before the invocation, allowing the replication algorithm to, for example, get and set attribute values of the replicated object.

The interesting aspect is that the framework can be set up independent of any replication algorithm, thus leading to a complete separation of object functionality and replication of objects.

4.1.7 Fault Tolerance

Q12. Write about Fault Tolerant in CORBA.

Ans :

➢ The basic approach for dealing with failures in CORBA is to replicate objects into object groups. Such a group consists of one or more identical copies of the same object.

➢ To provide replication and failure transparency as much as possible, object groups should not be distinguishable from normal CORBA objects.

➢ Whenever a client passes an IOGR to its runtime system (RTS), that RTS attempts to bind to one of the referenced replicas. In the case of IIOP, the RTS may possibly use additional information it finds in one of the IIOP profiles of the IOGR.

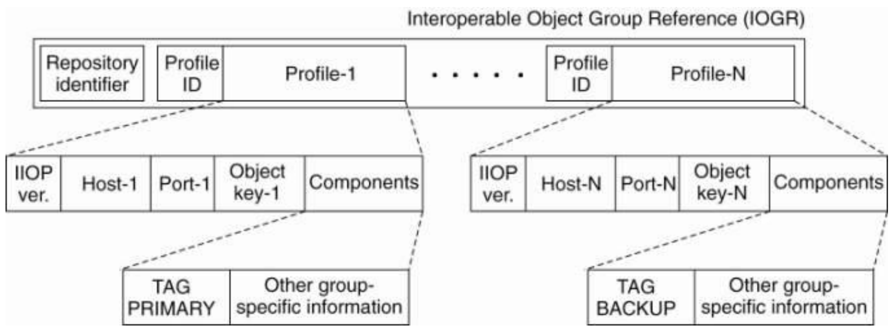


Fig.: A possible organization of an IOGR for an object group having a primary and backups

If binding to one of the replicas fails, the client RTS may continue by attempting to bind to another replica, thereby following any policy for next selecting a replica that it suits to best. To the client, the binding procedure is completely transparent; it appears as if the client is binding to a regular CORBA object.

An Example Architecture

To support object groups and to handle additional failure management, it is necessary to add components to CORBA. One possible architecture of a fault-tolerant version of CORBA is shown in Figure.

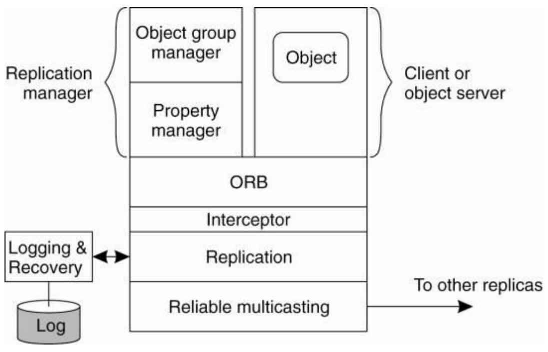


Fig.: An example architecture of a fault-tolerant CORBA system.

- There are several components that play an important role in this architecture. Tthe most important one is the replication manager, which is responsible for creating and managing a group of replicated objects.
- The number of replicas that are created when starting a new object group is normally determined by the system-dependent default value. The replica manager is also responsible for replacing a replica in the case of a failure, thereby ensuring that the number of replicas does not drop below a specified minimum.
- The architecture also shows the use of message-level interceptors. In the case of the Eternal system, each invocation is intercepted and passed to a separate replication component that maintains the required consistency for an object group and which ensures that messages are logged to enable recovery.
- This architecture is based on using interceptors. Alternative solutions exist as well, including those in which fault tolerance has been incorporated in the runtime system.

4.1.8 Security

Q13. Explain the security mechanism in Globe.

Ans :

Globe is one of the few distributed object-based systems in which an object's state can be physically distributed and replicated across multiple machines. This approach also introduces specific security problems .

Generally When invoking a method on a remote object, there are at least two issues that are important from a security perspective: (1) is the caller invoking the correct object and (2) is the caller allowed to invoke that method.

For Globe and other systems that support either replication or moving objects around, we have an additional problem, namely that of platform security. This kind of security comprises two issues. First, how can the platform to which a (local) object is copied be protected against any malicious code contained in the object, and secondly, how can the object be protected against a malicious replica server.

There are several mechanisms deployed in Globe to establish security.

First, every Globe object has an associated public/private key pair, referred to as the object key. The basic idea is that anyone who has knowledge about an object's private key can set the access policies for users and servers.

Every replica has an associated replica key, which is also constructed as a public/private key pair. This key pair is generated by the object server currently hosting the specific replica.. Finally, each user is also assumed to have a unique public/private key pair, known as the user key.

These keys are used to set the various access rights in the form of certificates. Certificates are handed out per object. There are three types, as shown in Fig. .The certificate contains a bit string U with the same length as the number of methods available for the object. U [i] = 1 if and only if the user is allowed to invoke method Mi. Likewise, there is also a replica certificate that specifies, for a given replica server, which methods it is allowed to execute. It also has an associated bit string R, where R [i] = i if and only if the server is allowed to execute method Mi.

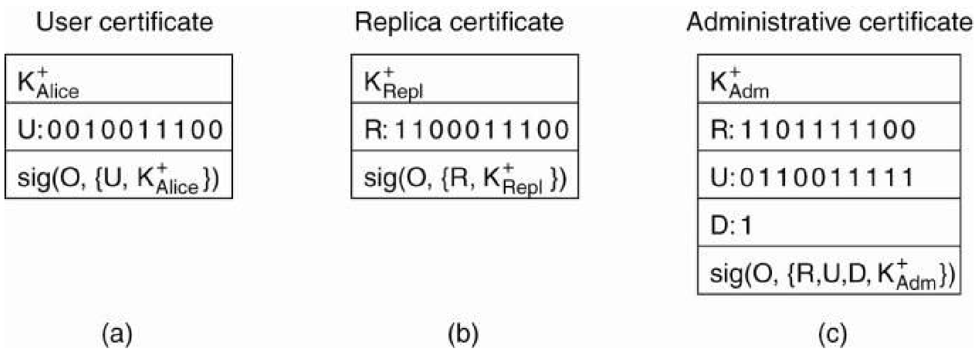


Fig.: Certificates in Globe: (a) a user certificate, (b) a replica certificate, (c) an administrative certificate

Secure Method Invocation

Let us now look into the details of securely invoking a method of a Globe object. The complete path from requesting an invocation to actually executing the operation at a replica is sketched in Figure.

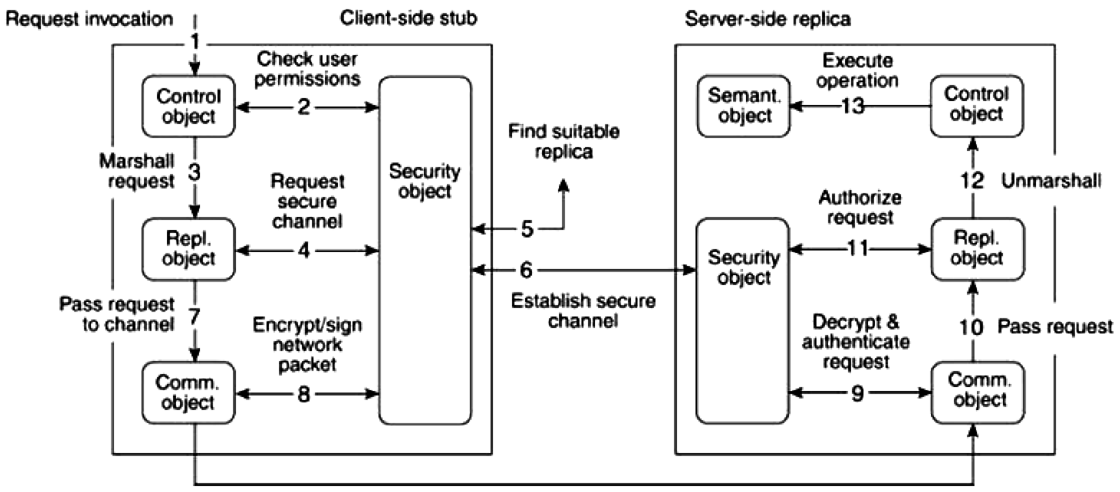


Fig.: Secure method invocation in Globe

1. First, an application issues a invocation request by locally calling the associated method, just like calling a procedure in an RPC.
2. The control subobject checks the user permissions with the information stored in the local security object. In this case, the security object should have a valid user certificate.
3. The request is marshaled and passed on.
4. The replication subobject requests the middleware to set up a secure channel to a suitable replica.
5. The security object first initiates a replica lookup. To achieve this goal, it could use any naming service that can look up replicas that have been specified to be able to execute certain methods. The Globe location service has been modified to handle such lookups (Ballintijn, 2003).
6. Once a suitable replica has been found, the security subobject can set up a secure channel with its peer, after which control is returned to the replication subobject. Note that part of this establishment requires that the replica proves it is allowed to carry out the requested invocation.

7. The request is now passed on to the communication subobject.
8. The subobject encrypts and signs the request so that it can pass through the channel.
9. After its receipt, the request is decrypted and authenticated.
10. The request is then simply passed on to the server-side replication subobject.
11. Authorization takes place: in this case the user certificate from the client-side stub has been passed to the replica so that we can verify that the request can indeed be carried out.
12. The request is then unmarshaled.
13. Finally, the operation can be executed.

4.2 Distributed File Systems

4.2.1 Architecture

Q14. What are distributed file systems? Explain.

Ans :

A Distributed File System (DFS)is simply a classical model of a file system distributed across multiple machines. The purpose is to promote sharing of dispersed files. There sources on a particular machine are **local** to itself. Resources on other machines are **remote**.

A file system provides a service for clients. These interface is the normal setoff the operations: create, read, etc. on files.

Clients, servers, and storage are dispersed across machines. Configuration and implementation may vary –

a) Servers may run on dedicated machines, OR
b) Servers and clients can be on the same machines.
c) The OS itself can be distributed.
d) Adistribution layer can be interposed between a conventional OS and the file system.

Clients should view a DFS the same way they would a centralized FS; the distribution is hidden at a lower level.

Performance is concerned with through put and response time.

Distributed file system support

➢ **Remote Information Sharing**

Allows a file to be transparently accessed by processes of any node of the system irrespective of the file's location.

➢ **User Mobility -** User have flexibility to work on different node at different time

➢ **Availability -** better fault tolerance

➢ **Diskless Workstations**

Distributed File System provide following type of services:

➢ Storage Service
➢ True File Service
➢ Name Service

Desirable features of a good distributed file system

➢ Transparency
➢ Structure transparency
➢ Access Transparency
➢ Naming Transparency
➢ Replication Transparency
➢ User Mobility
➢ Performance
➢ Simplicity and ease of use

Scalability

➢ High Availability
➢ High Reliability
➢ Data Integrity
➢ Security
➢ Heterogeneity

File Models

Criteria: Structure and Modifiability

Structured and Unstructured Files

Structured Files: A file appear to the file server as an ordered sequence of records.

- Files with Indexed Records
- Files With non-indexed records
- Unstructured files: No substructure known to the file server

Mutable and Immutable Files

Mutable

- An update performed on a file overwrites on its old contents
- A file is represented as a single stored sequence that is altered by each update operation.

Immutable Files

- A file cannot be modified once it has been created
- File versioning approach used to implement file updates
- It support consistent sharing therefore it is easier to support file caching and replication

File Accessing Models

File Accessing Models of DFS mainly depends on: Method used for accessing remote files and the unit of data access

Accessing Remote Files

- **Remote Service Model**
  - Client's request processed at server's node
  - In this case Packing and communication overhead can be significant
- **Data Caching Model**
  - Client's request processed on the client's node itself by using the cached data.
  - This model greatly reduces network traffic
  - Cache consistency problem may occur

  LOCUS and NFS use the remote service model but add caching for better performance

  Sprite use data caching model but employs the remote service model under certain circumstances.

Unit of Data Transfer

File Level Transfer Model(Ex. Amoeba, AFS)

- The whole file is moved when an operation requires file data
- It is simple, It has better scalability
- Disk access routines on the servers can be better optimized
- But it requires sufficient storage space on client's node

Block Level Transfer Model(Ex. LOCUS, Sprite)

- Data transferred in units of file blocks
- It does not require client node to have large storage space
- It can be used in diskless workstations
- Network traffic may be significant

Byte Level Transfer Model(Cambridge file server)

- Data transfers in units of bytes
- Low Storage requires but difficulty in cache management

Record Level Transfer Model(Research Storage System)

- Suitable for Structured model

Naming and Transparency

Naming is the mapping between logical and physical objects.

- **Example:** A use rfile name maps to <cylinder, sector>.
  - In a conventional file system, it's understood where the file actually resides; the system and disk are known.
  - In a transparent DFS, the location of a file, some where in the network, is hidden.
  - File replication means multiple copies of a file; mapping returns a SET of locations for the replicas.

Location transparency

a) The name of a file does no treveal any hint of the file's physicals to ragelocation.

b) File name still denotes as pecific, although hidden, set of physical disk blocks.

c) This is a convenient way to share data.

d) Can expose correspondence between component units and machines.

Location Independence

- The name of a file does n't need to be changed when the file's physical storage location changes. Dynamic, one-to-many mapping.
- Better file abstraction.
- Promotes sharing the storage space itself.
- Separates the naming hierarchy from the storage devices hierarchy.

Most DFS stoday

- Support location transparent systems.
- Do NOT support migration
- Files are permanently associated with specific disk blocks.

The ANDREWDFSASAN EXAMPLE

- Islocation independent.
- Supports file mobility.
- Separation of FS and OS allows fordisk-less systems. These have lower cost and conve-nient system up grades. The performance is not as good.

NAMING SCHEMES

There are three main approaches to naming files:

1. Files are named with a combination of host and local name.
   - This guarantees a unique name. NEITHER location transparent NOR location independent.
   - Same naming works on local and remotefiles. The DFS is a loose collection of independent file systems.
2. Remote directories are mounted to local directories.
   - Soalocal system seems to have a coherent directory structure.
   - There mote directories must be explicitly mounted. The files are location independent.
   - SUNNFS is a good example of this technique.
3. A single global name structures pansall the files in the system.
   - The DFS is built the same wayas alocal file system. Location independent.

Q15. Explain about Sun Network File Systmes.

Ans :

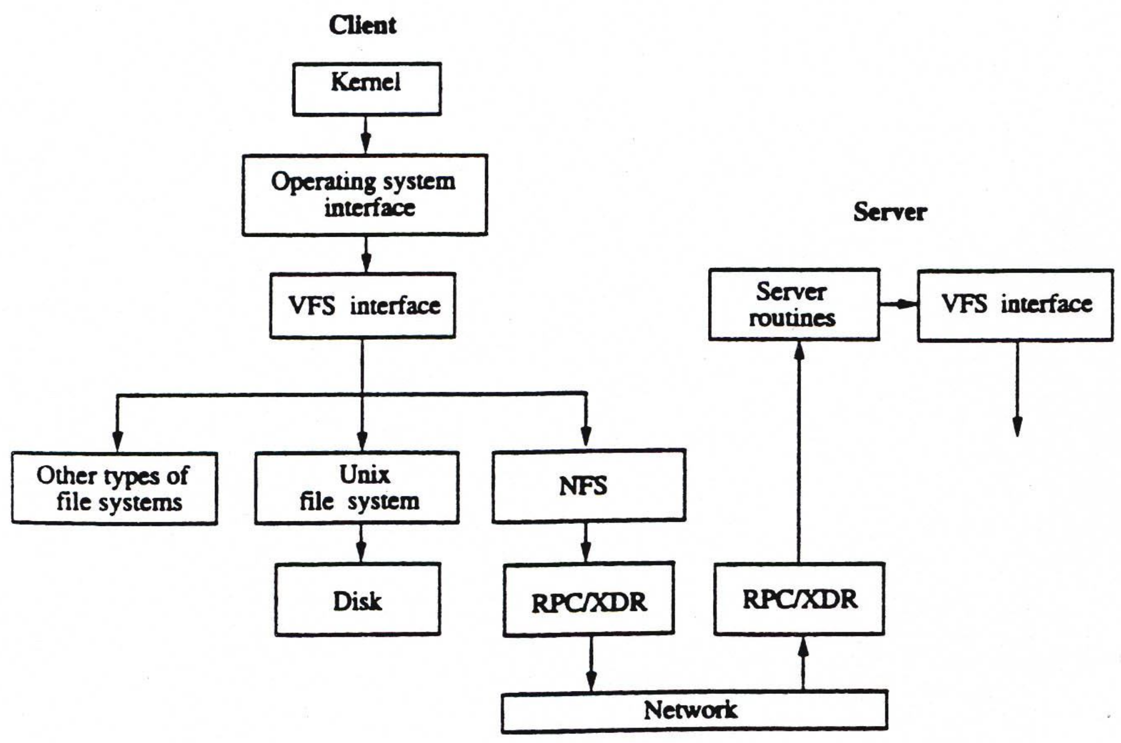
The Sun Network File System (NSF)

Developed by Sun Microsystems to provide a distributed file system independent of the hardware and operating system

Architecture

Virtual File System (VFS): File system interface that allows NSF to support different file systems. Requests for operation on remote files are routed by VFS to NFS. Requests are sent to the VFS on the remote using the remote procedure call (RPC), and the external data representation (XDR). VFS on the remote server initiates files system operation locally

Vnode (Virtual Node): There is a network-wide vnode for every object in the file system (file or directory)- equivalent of UNIX inode, vnode has a mount table, allowing any node to be a mount node.



- Naming and location:
  - Workstations are designated as clients or file servers
  - A client defines its own private file system by mounting a subdirectory of a remote file system on its local file system
  - Each client maintains a table which maps the remote file directories to servers
  - Mapping a filename to an object is done the first time a client references the field. Example:

    Filename: /A/B/C

    - Assume 'A' corresponds to 'vnode1'
    - Look up on 'vnode1/B' returns 'vnode2' for 'B' where'vnode2' indicates that object is on server 'X'
    - Client asks server 'X' to lookup 'vnode2/C'
    - 'file handle' returned to client by server storing that file

- Client uses 'file handle' for all subsequent operation on that fileNaming and location:
  - Workstations are designated as clients or file servers
  - A client defines its own private file system by mounting a subdirectory of a remote file system on its local file system
  - Each client maintains a table which maps the remote file directories to servers
  - Mapping a filename to an object is done the first time a client references the field. Example:

    Filename: /A/B/C
- Assume 'A' corresponds to 'vnode1'
- Look up on 'vnode1/B' returns 'vnode2' for 'B' where'vnode2' indicates that object is on server 'X'
- Client asks server 'X' to lookup 'vnode2/C'
- 'file handle' returned to client by server storing that file

Client uses 'file handle' for all subsequent operation on that file

- Caching:
  - Caching done in main memory of clients
  - Caching done for: file blocks, translation of filenames to vnodes, and attributes of files and directories

1) Caching of file blocks

- Cached on demand with time stamp of the file (when last modified on the server)
- Entire file cached, if under certain size, with timestamp when last modified
- After certain age, blocks have to be validated with server
- Delayed writing policy: Modified blocks flushed to the server after certain delay

2) Caching of filenames to vnodes for remote directory names

- Speeds up the lookup procedure

3) Caching of file and directory attributes

- Updated when new attributes received from the server, discarded after certain time
- Stateless Server
- Servers are stateless
- File access requests from clients contain all needed information (pointer position, etc)
- Servers have no record of past requests
- Simple recovery from crashes.

Q16. Write about Cluster Based File systems.

Ans :

Cluster-based or Clustered File System

- A distributed file system that consists of several servers that share the responsibilities of the system, as opposed to a single server (possibly replicated).

  The design decisions for a cluster-based systems are mostly related to how the data is distributed across the cluster and how it is managed.
- Some cluster-based systems organize the clusters in an application specific manner
- For file systems used primarily for parallel applications, the data in a file might be striped across several servers so it can be read in parallel.
- Or, it might make more sense to partition the file system itself – some portion of the total number of files are stored on each server.
- For systems that process huge numbers of requests; e.g., large data centers, reliability and management issues take precedence.

  e.g., Google File System

Google File System (GFS)

- GFS uses a cluster-based approach implemented on ordinary commodity Linux boxes (not high-end servers).
- Servers fail on a regular basis, just because there are so many of them, so the system is designed to be fault tolerant.

- There are a number of replicated clusters that map to www.google.com

DNS servers map requests to the clusters in a round-robin fashion, as a load-balancing mechanism; locality is also considered.

- GFS stores a huge number of files, totaling many terabytes of data
- Individual file characteristics
- Very large, multiple gigabytes per file
- Files are updated by appending new entries to the end (faster than overwriting existing data)
- Files are virtually never modified (other than by appends) and virtually never deleted.
- Files are mostly read-only

4.2.2 Process

Q17. What is the role of Processes in DFS.

Ans :

When it comes to processes, distributed file systems have no unusual properties

- Typical types of cooperating processes:
  - Servers, file managers, client software
  - Should servers be stateless
  - e.g., as in NFSv2 and v3 – but not NFSv4
- Advantage: Simplicity

Server crashes are easy to process since there is no state to recover

Disadvantages of Statelessness

- The server cannot inform the client whether or not a request has been processed.
- Consider implications for lost request/lost replies when operations are not idempotent
- File locking (to guarantee one writer at a time) is not possible

NFS got around this problem by supporting a separate lock manager.

NFSv4

- Maintains some minimal state about its clients; e.g., enough to execute authentication protocols Stateful servers are better equipped to run over wide area networks, because they are better able to manage consistency issues that arise when clients are allowed to cache portions of files locally

4.2.3 Communication

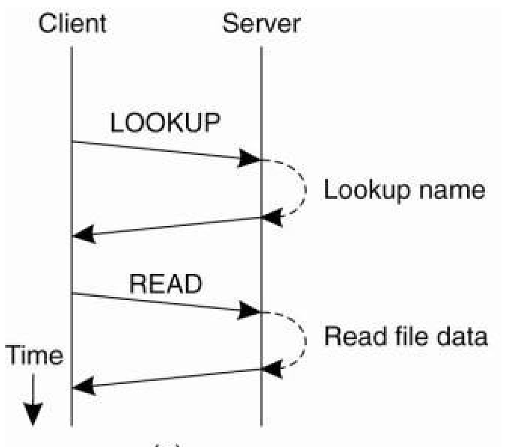
Q18. Write about communication in DFS.

Ans :

There is nothing particularly special or unusual about communication in distributed file systems. Many of them are based on remote procedure calls (RPCs). The main reason for choosing an RPC mechanism is to make the system independent from underlying operating systems, networks, and transport protocols.

RPCs in NFS

- Client-server communication in NFS is based on Open Network Computing RPC (ONC RPC) protocols.
- Each file system operation is represented as an RPC. Pre-version 4 NFS required one RPC at a time, so server didn't have to remember any state.
- NFSv4 supports compound procedures (several RPCs grouped together)



(a)

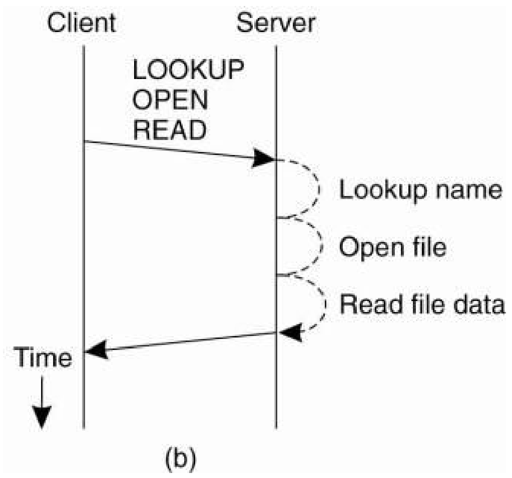


Fig.: (a) Reading data from a file in NFS version 3. (b) Reading data using a compound procedure in version 4.

4.2.4 Naming

Q19. Write a note on Naming service in DFS.

Ans :

Naming

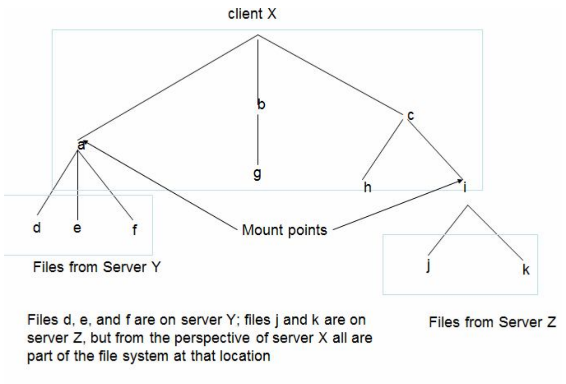
- NFS is used as a typical example of naming in a DFS.
- Virtually all support a hierarchical namespace organization.
- NFS naming model strives to provide transparent client access to remote file systems.

Goal

- Network (Access)Transparency
- Users should be able to access files over a network as easily as if the files were stored locally.
- Users should not have to know the location of a file to access it. Transparency can be addressed through naming and file mounting mechanisms

Mounting

- Servers export file systems; i.e, make them available to clients
- Client machines can attach a remote FS (directory or subdirectory) to the local FS at any point in its directory hierarchy.
- When a FS is mounted, the client can reference files by the local path name – no reference to remote host location, although files remain physically located at the remote site.
- Mount tables keep track of the actual physical location of the files.


client X
a
b
c
g
h
i
d
e
f
Mount points
Files from Server Y
j
k
Files d, e, and f are on server Y; files j and k are on server Z, but from the perspective of server X all are part of the file system at that location
Files from Server Z

Q20. What are File Handles in DFS.

Ans :

File Handles

A file handle is a reference to a file that is created by the server when the file is created.

- It is independent of the actual file name
- It is not known to the client (although the client must know the size)

It is used by the file system for all internal references to the file.

Benefits of File Handles

There is a uniform format for the file identifier inside the file system.

Clients can store the handle locally after an initial reference and avoid the lookup process on subsequent file operations.

4.2.5 Synchronization

Q21. Write about, how Synchronization takes place for the file systems.

Ans :

Synchronization for file systems would not be an issue if files were not shared.

Semantics of File Sharing

When two or more users share the same file at the same time, it is necessary to define the semantics of reading and writing precisely to avoid problems.

Example

- Single – processor systems
- Distributed system
- In single-processor systems that permit processes to share files, such as UNIX, the semantics normally state that when a read operation follows a write operation, the read returns the value just written, as shown in Figure (a).
- In a distributed system, if a client locally modifies a cached file and shortly thereafter another client reads the file from the server, the second client will get an obsolete file.

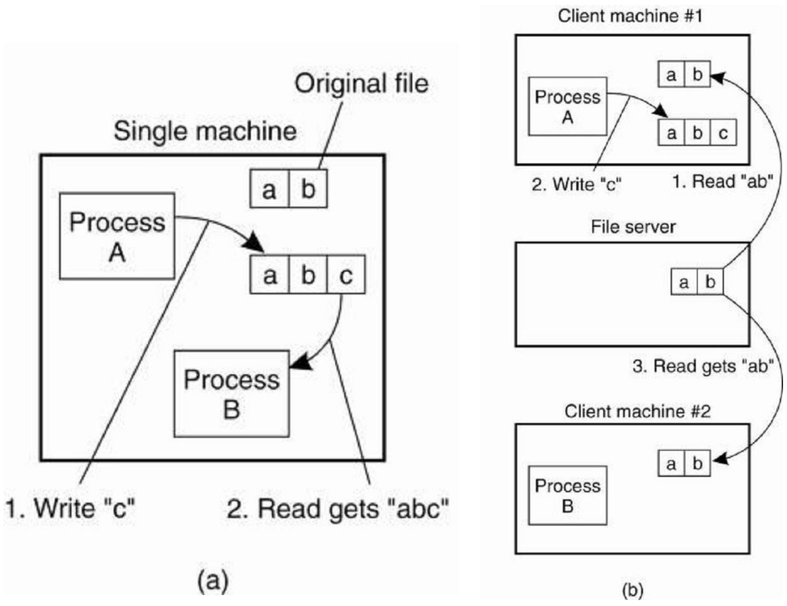
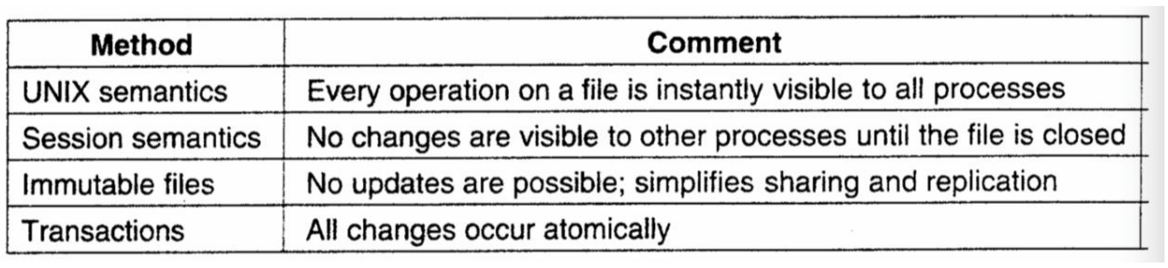


Fig.: (a) On a single processor, when a read follows a write, the value returned by the read is the value just written. (b) In a distributed system with caching, obsolete values may be returned.

In a distributed system, UNIX semantics can be achieved easily as long as there is only one file server and clients do not cache files. All reads and writes go directly to the file server, which processes them strictly sequentially.

Four approaches to deal with shared files in a distributed system.



Method	Comment
UNIX semantics	Every operation on a file is instantly visible to all processes
Session semantics	No changes are visible to other processes until the file is closed
Immutable files	No updates are possible; simplifies sharing and replication
Transactions	All changes occur atomically

UNIX Semantics

- UNIX semantics can be achieved easily as long as there is only one file server and client do not cache files.
- All reads and writes go directly to the file server, which processes them strictly sequentially.

Session Semantics

- Instead of requiring a read to see the effects of all previous writes, one can have a new rule:
  - Changes to an open file are initially visible only to the process that modified the file.
  - Only when the file is closed are the changes made visible to other processes.
- Most distributed file systems implement session semantics.

Immutable Files

- To create an entirely new file and enter it into the directory system under the name of a previous existing file, which now becomes inaccessible.
- Files cannot be updated, but directories can be.
- The only operations on files are create and read, no way to open and write

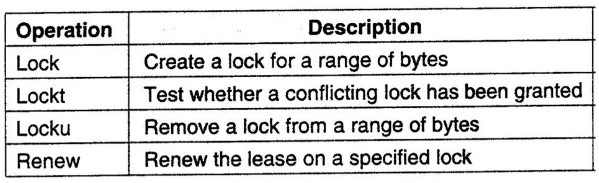
Transactions

- To access a file or a group of files, a process first executes some type of BEGIN_ TRANSACTION primitive to signal that what follows must be executed indivisibly.
- When the requested work has been completed, an END_TRANSACTION primitive is executed.
- The system guarantees that all the calls contained within the transaction will be carried out in order.

File Locking

- A central lock manager is generally deployed for synchronizing access to shared files.
- The complexity in locking comes from the need to allow concurrent access to the same file.
- A great number of different locks exist, and moreover, the granularity of locks may also differ.

NFSv4 operations related to file locking



Operation	Description
Lock	Create a lock for a range of bytes
Lockt	Test whether a conflicting lock has been granted
Locku	Remove a lock from a range of bytes
Renew	Renew the lease on a specified lock

Lock

- Operation lock is used to request a read or write lock on a consecutive range of bytes in a file.
- A client will get back an error message if the operation is conflicted with another lock.
- A FIFO – ordered list will grant the next lock to the client at the top of the list once the conflicting lock has been removed.

Lockt

- The lockt operation is used to test whether a conflicting lock exists.
- In the case of a conflict, the requesting client is informed exactly who is causing the conflict and on which range of bytes.
- It can be implemented more efficiently than lock, because there is no need to attempt to open a file.

Locku

- Locku operation removes a lock from a file.

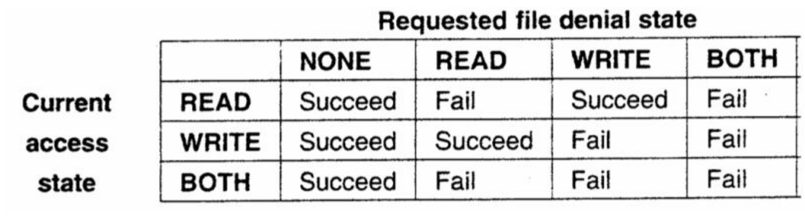
Renew

- A client requests the server to renew the lease on its lock. Otherwise, the server will automatically remove the lock after a granted specific time. Locku operation removes a lock from a file.

Share Reservation

- An implicit way to lock a file.
- Is independent from locking, can be used to implement NFS for Windows – based systems.

➢ When a client opens a file, it specifies the type of access it requires (READ, WRITE or BOTH), and which type of access the server should deny other clients. (NONE, READ, WRITE or BOTH).



Current access state	Requested file denial state			
	NONE	READ	WRITE	BOTH
READ	Succeed	Fail	Succeed	Fail
WRITE	Succeed	Succeed	Fail	Fail
BOTH	Succeed	Fail	Fail	Fail

Show the result of opening a file that is currently being accessed by anther client, but now requesting certain access types to be disallowed.

4.2.6 Consistency and Replication

Q22. How caching and replication is useful in DFS, Explain.

Ans :

Caching and replication play an important role in distributed file systems.

Client-Side Caching

Caching in NFS

Caching in NFSv3 hasled to the implementation of different caching policies, most of which never guaranteed consistency.

NFSv4 solves some consistency problems, but essentially still leaves cache consistency to be handled in an implementation-dependent way.

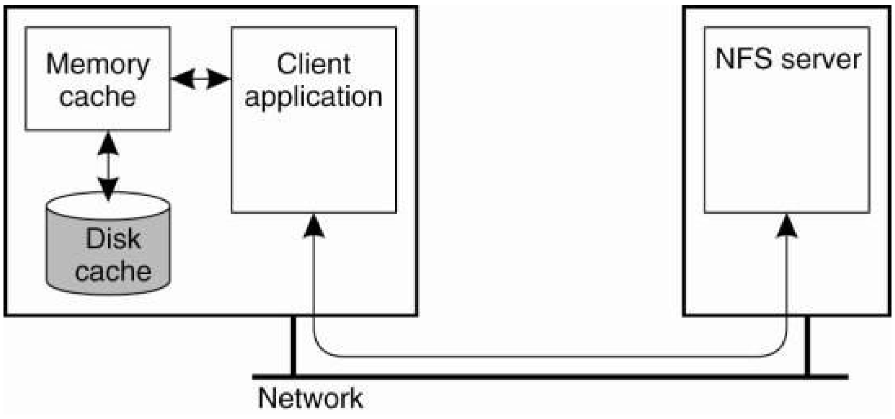


Fig.: Client-side caching in NFS

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.

NFSv4 supports two different approaches for caching file data.

➢ When a client opens a file and caches the data it obtains from the server as the result of various read operations.

➢ NFS requires that whenever a client opens a previously closed file that has been (partly) cached, the client must immediately revalidate the cached data.

In NFSv4 a server may delegate some of its rights to a client when a file is opened. Open delegation takes place when the client machine is allowed to locally handle open and close operations from other clients on the same machine.

An important consequence of delegating a file to a client is that the server needs to be able to recall the delegation Recalling a delegation requires that the server can do a callback to the client, as illustrated in Figure.

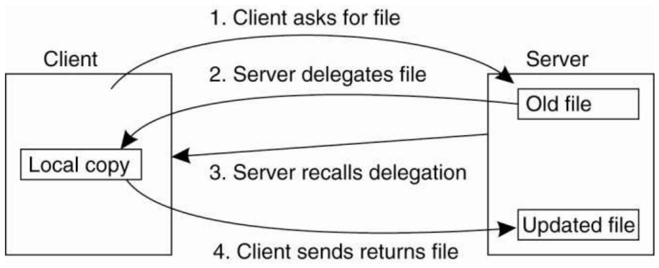


Fig.: Using the NFSv4 callback mechanism to recall file delegation

A similar approach is followed for caching file handles and directories.

4.2.7 Fault Tolerance

Q23. Explain how to handle Byzantine Failures in DFS.

Ans :

Handling Byzantine Failures

One of the problems that is often ignored when dealing with fault tolerance is that servers may exhibit arbitrary failures.

The basic idea is to deploy active replication by constructing a collection of finite state machines and to have the nonfaulty processes in this collection execute operations in the same order. Assuming that at most k processes fail at once, a client sends an operation to the entire group and accepts an answer that is returned by at least k + 1 different processes.

To achieve protection against Byzantine failures, the server group must consist of at least 3k + 1 processes. The difficult part in achieving this protection is to ensure that nonfaulty processes execute all operations in the same order.

A simple means to achieve this goal is to assign a coordinator that simply serializes all operations by attaching a sequence number to each request.

An important part of the protocol relies on the fact that requests can be correctly ordered.

The whole protocol consists of five phases, shown in Fig.

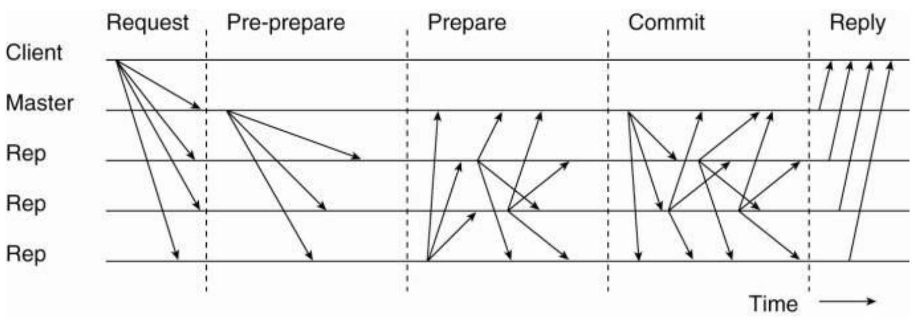


Fig.: The different Phases in Byzantine Fault Tolerance

Request Phase

During the first phase, a client sends a request to the entire server group. Once the master has received the request, it multicasts a sequence number in a pre-prepare phase so that the associated operation will be properly ordered.

Pre-Prepare phase

Here the slave replicas need to ensure that the master's sequence number is accepted by a quorum, provided that each of them accepts the master's proposal.

Prepare Pahse

The slave accepts the proposed sequence number, it multicasts this acceptance to the others.

Commit Phase

During the commit phase, agreement has been reached and all processes inform each other and execute the operation, after which the client can finally see the result.

Reply Phase

The final result has been seen

4.2.8 Security

Q24. Write about the security in NFS.

Ans :

Security in distributed file systems organized along a client-server architecture is to have the servers handle authentication and access control.

Security in NFS

The basic idea behind NFS is that a remote file system should be presented to clients as if it were a local file system.

In addition to secure RPCs, it is necessary to control file accesses. which are handled by means of access control file attributes in NFS. A file server is in charge of verifying the access rights of its clients, as we will explain below. Combined with secure RPCs, the NFS security architecture is shown in Fig:

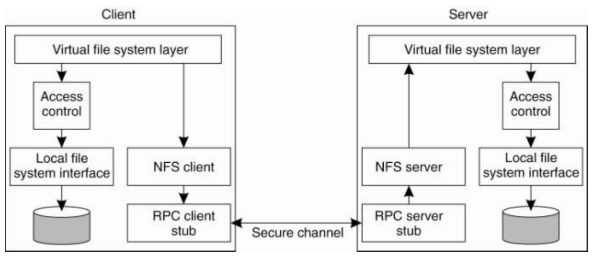

Client
Server
Virtual file system layer
Access control
Local file system interface
NFS client
RPC client stub
Secure channel
Virtual file system layer
Access control
NFS server
Local file system interface
RPC server stub

Fig.: The NFS security architecture

Secure RPCs

Because NFS is layered on top of an RPC system, setting up a secure channel in NFS is necessary.

There are three authentication methods used for secure RPCs:

➢ The first authentication method is In this UNIX-based method, a client simply passes its effective user ID and group ID to the server, along with a list of groups it claims to be a member of. This information is sent to the server as unsigned plaintext.

➢ The second authentication method in older NFS versions uses Diffie-Hellman key exchange to establish a session key, leading to what is called secure NFS.

➢ The third authentication protocol is Kerberos,.

NFS security is enhanced by the support for RPCSEC_GSS. RPCSEC_GSS is a general security framework that can support a myriad of security mechanism for setting up secure channels .

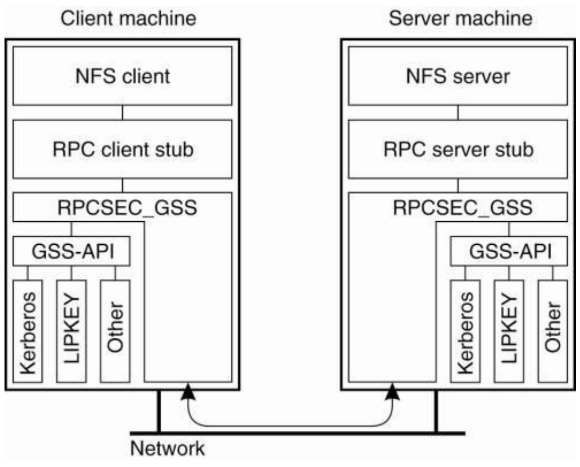


Fig.: Secure RPC in NFSv4

The important aspect of secure RPC in NFS is that the designers have chosen not to provide their own security mechanisms, but only to provide a standard way for handling security.

Access Control

Authorization in NFS provides the mechanisms but does not specify any particular policy. Access control is supported by means of the ACL file attribute.

This attribute is a list of access control entries, where each entry specifies the access rights for a specific user or group.

Type of user	Description
Owner	The owner of a file
Group	The group of users associated with a file
Everyone	Any user or process
Interactive	Any process accessing the file from an interactive terminal
Network	Any process accessing the file via the network
Dialup	Any process accessing the file through a dialup connection to the server
Batch	Any process accessing the file as part of batch job
Anonymous	Anyone accessing the file without authentication
Authenticated	Any authenticated user or process
Service	Any system-defined service process

Fig.: The various kinds of users and processes distinguished by NFS with respect to access control.

Decentralized Authentication

One of the main problems with systems such as NFS is that in order to properly handle authentication, it is necessary that users are registered through a central system administration. A solution to this problem is provided by using the Secure File Systems (SFS) in combination with decentralized authentication servers.

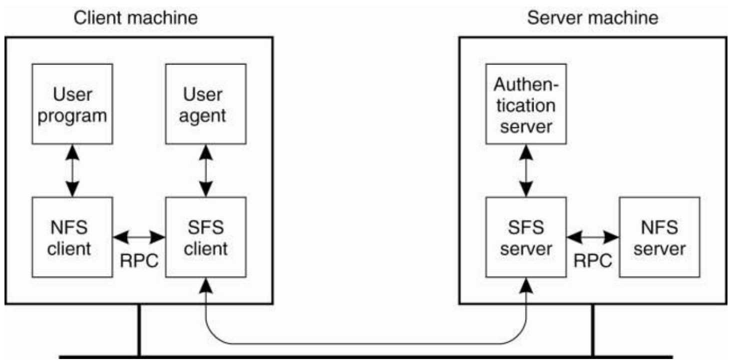


Fig.: The organization of SFS

The SFS client is responsible for setting up a secure channel with an SFS server. It is also responsible for communicating with a locally-available SFS user agent, which is a program that automatically handles user authentication.

The NFS server is again used for portability reasons. This server communicates with the SFS server which operates as an NFS client to the NFS server. The SFS server forms the core process of SFS. This process is responsible for handling file requests from SFS clients. Analogous to the SFS agent, an SFS server communicates with a separate authentication server to handle user authentication.

4.3 Distributed Webbased Systems

4.3.1 Architecture

Q25. Explain about distributed web based systems and its architecture

Ans :

1. The World Wide Web (WWW) can be viewed as a huge distributed system with millions of clients and servers for accessing linked documents.
2. Servers maintain collections of documents while clients provide users an easy-to-use interface for presenting and accessing those documents.
3. 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 document stored locally or in another part of the world.
4. Now, Web has become more than just a simple document based system.
5. With the emergence of Web services, it is becoming a system of distributed services rather than just documents offered to any user or machine.

Traditional Web-based Systems

Many Web-based systems are still organized as simple client-server architectures.

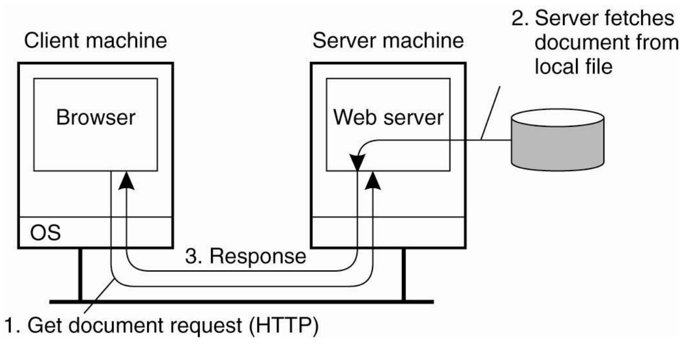
Relatively simple client-server architecture

Access to local file system

Simplest way to refer a document is by Uniform Resource Locator (URL)

URL specifies the application-level protocol for transfers across the network

Several different protocols


Client machine
Server machine
2. Server fetches document from local file
Browser
Web server
OS
3. Response
1. Get document request (HTTP)

The core of a Web site: a process that has access to a local file system storing documents.

Uniform Resource Locator

- A reference called Uniform Resource Locator (URL)is used to refer a document.

  The DNS name of its associated server along with a file name is specified.

- The URL also specifies the protocol for transferring the document across the network.

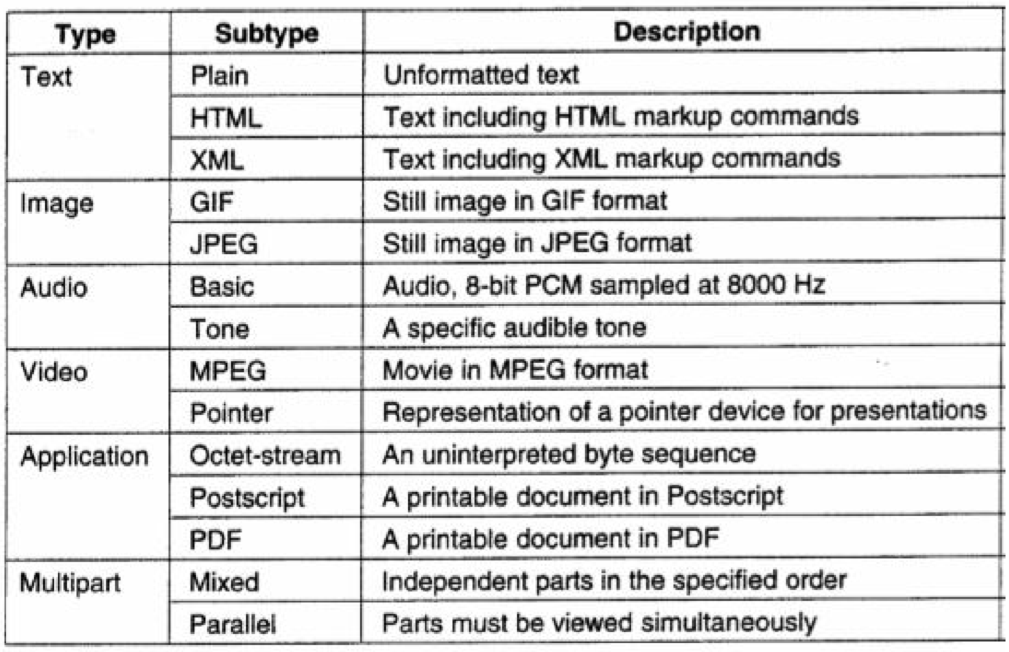
  Example: http://www.cse.unl.edu/~ylu/csce855/notes/web-system.ppt

Web Documents

A Web document does not only contain text, but it can include all kinds of dynamic features such as audio, video, animations, etc. In many cases special helper applications (interpreters) are needed, and they are integrated into the browser.

E.g., Windows Media Playerand QuickTime Playerfor playing streaming content

The variety of document types forces browser to be extensible. As a result, plug-ins are required to follow a standard interfaces so that they can be easily integrated with the browsers.



Type	Subtype	Description
Text	Plain	Unformatted text
	HTML	Text including HTML markup commands
	XML	Text including XML markup commands
Image	GIF	Still image in GIF format
	JPEG	Still image in JPEG format
Audio	Basic	Audio, 8-bit PCM sampled at 8000 Hz
	Tone	A specific audible tone
Video	MPEG	Movie in MPEG format
	Pointer	Representation of a pointer device for presentations
Application	Octet-stream	An uninterpreted byte sequence
	Postscript	A printable document in Postscript
	PDF	A printable document in PDF
Multipart	Mixed	Independent parts in the specified order
	Parallel	Parts must be viewed simultaneously

Fig.: Six top-level MIME types and some common subtypes

Q26. Explain multi tier architecture.

Ans :

Multitiered Architectures

Web documents can be built in two ways:

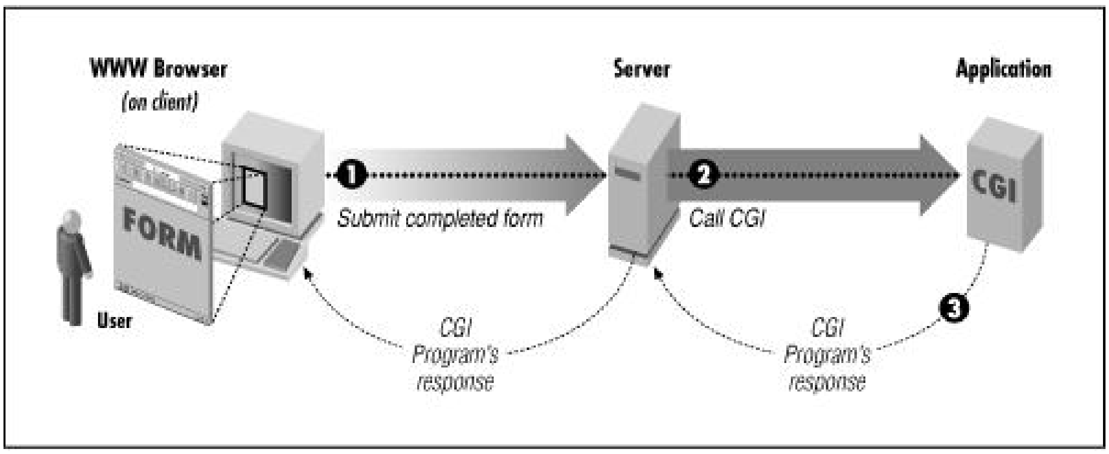
Static Web Pages

- Locates and returns the object identified in the request.
- Includes predefined HTML pages and JPEG ad GIF files
- Web servers do not require communication with any server side application

Dynamic Web Pages

- The request is forwarded to an application system where the response is generated dynamically on the server by server side program execution.
- The combination of HTML or XML with scripting provides powerful means of expressing documents.

Although Web started as simple two-tiered client-server architecture for static Web documents, this architecture has been extended to support advanced type of documents.



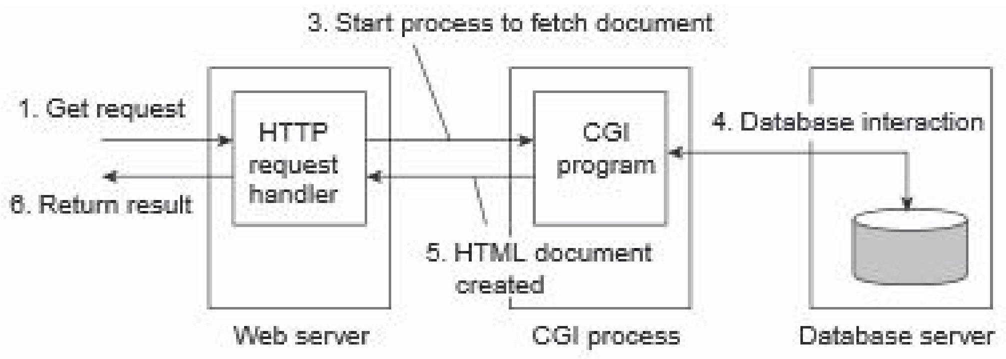
Server side CGI programs

Because of the server-side processing, many Web sites are now organized as three-tiered architectures consisting of a Web server, an application server, and a database server.

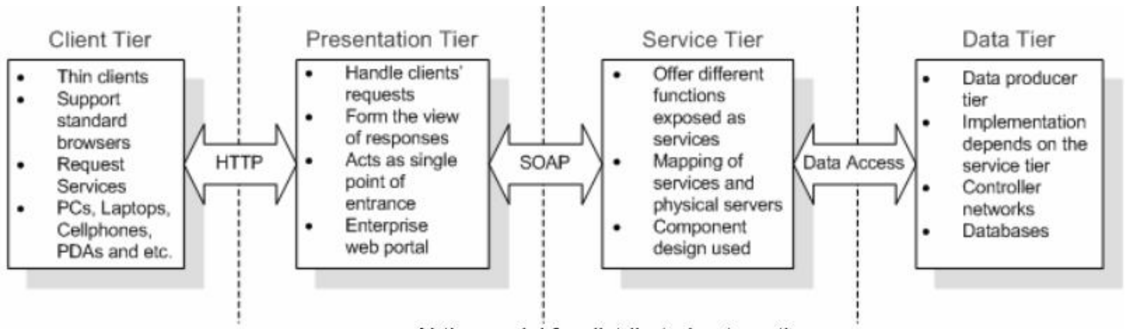
User data comes from an HTML form, specifying the program and parameters.

Server-side scripting technologies are used to generate dynamic content:

- Microsoft: Active Server Pages (ASP.NET)
- Sun: Java Server Pages (JSP)
- Netscape: JavaScript
- Free Software Foundation: PHP



The following figure shows the example of N-tier model.


Client Tier
Thin clients
Support standard browsers
Request Services
PCs, Laptops, Cellphones, PDAs and etc.
HTTP
Presentation Tier
Handle clients' requests
Form the view of responses
Acts as single point of entrance
Enterprise web portal
SOAP
Service Tier
Offer different functions exposed as services
Mapping of services and physical servers
Component design used
Data Access
Data Tier
Data producer tier
Implementation depends on the service tier
Controller networks
Databases

N-tier model for distributed automation.

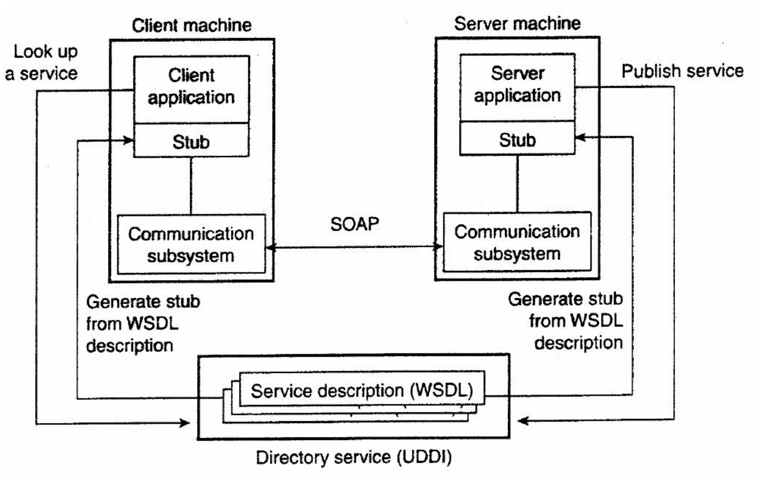
Q27. Explain about Web Services.

Ans :

Web Service

Principle of providing and using a Web service(similar to remote procedure call).

- Directory service storing service descriptions, it is adhere to Universal Description, Discovery and Integration standard (UDDI)
- UDDI prescribes the layout of a database containing service descriptions that will allow Web service clients to browse for relevant services.



Language

- Services are described by means of the Web Services Definition Language (WSDL)
- A WSDL description contains the precise definitions of the interfaces provided by a service, that is, procedure specification, data types, the (logical) location of services
- An important issue of a WSDL description is that can be automatically translated to client +side and server-side stubs.

Communication

- The Simple Object Access Protocol (SOAP) is used to determine how communication take place.
- SOAP is essentially a framework in which much of the communication between two processes can be standardized

4.3.2 Process

Q28. Write a note on Web Browser.

Ans :

Clients (Web Browsers)

The most important 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 user's screen.

Web browsers used to be simple programs, but that was long ago. Logically, they consist of several components, shown in figures [see also Grosskurth and Godfrey (2005)].

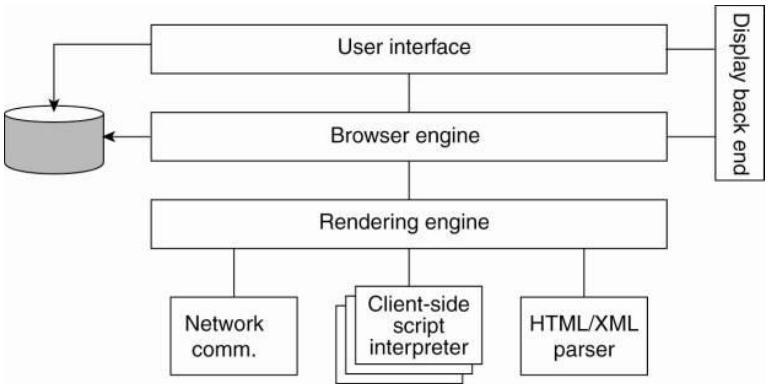
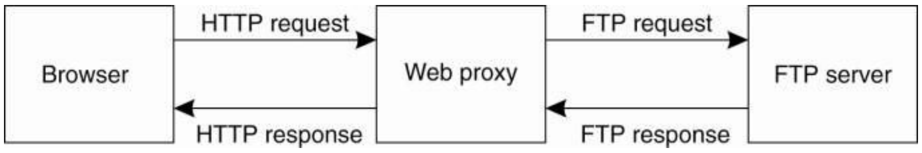


Fig.: Web browser

An important aspect of Web browsers is that they should (ideally) be platform independent. This goal is often achieved by making use of standard graphical libraries, shown as the display back end, along with standard networking libraries.

Originally, Web browsers were designed to support HTTP only. A Web proxy process was introduced to allow the browser to access FTP content:



Nowadays, Web browsers support a large range of document formats due to the availability of code migration methods. But proxies remain popular due to additional functions that they can perform: authentication, caching.

Q29. Explain about Apache Webserver.

Ans :

An HTTP web server is a web server that delivers content over HTTP, or the Hypertext Transfer Protocol, versus others like FTP. For example, when you go to Lifewire.comin your web browser, you're ultimately contacting the web server that hosts this website so that you can communicate with it to request web pages

Apache Web Server is designed to create web servers that have the ability to host one or more HTTP-based websites. Notable features include the ability to support multiple programming languages, server-side scripting, an authentication mechanism and database support. Apache Web Server can be enhanced by manipulating the code base or adding multiple extensions/add-ons.

It is also widely used by web hosting companies for the purpose of providing shared/virtual hosting, as by default, Apache Web Server supports and distinguishes between different hosts that reside on the same machine.

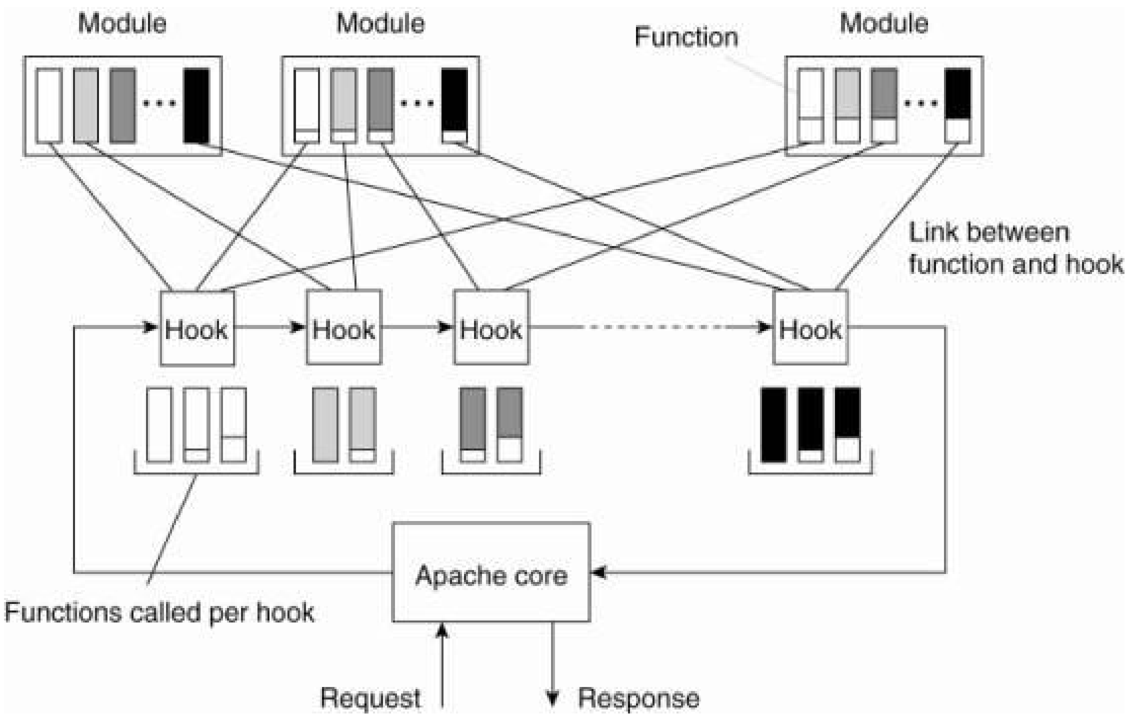


Fig.: The general organization of the Apache Web server

Web Server Clusters

An important problem related to the client-server nature of the Web is that a Web server can easily become overloaded.

To improve performance and availability, WWW servers are often clustered in a way that is transparent to clients.

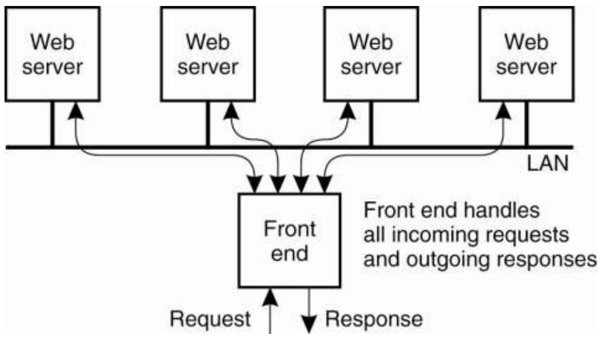


Fig.: The principle of using a server cluster in combination with a front end to implement a Web service.

Whenever a client issues an HTTP request, it sets up a TCP connection to the server. A transport-layer switch simply passes the data sent along the TCP connection to one of the servers, depending on some measurement of the server's load. The response from that server is returned to the switch, which will then forward it to the requesting client. As an optimization, the switch and servers can collaborate in implementing a TCP handoff The main drawback of a transport-layer switch is that the switch cannot take into account the content of the HTTP request that is sent along the TCP connection.

4.3.3 Communication

Q30. Explain about HTTP protocol.

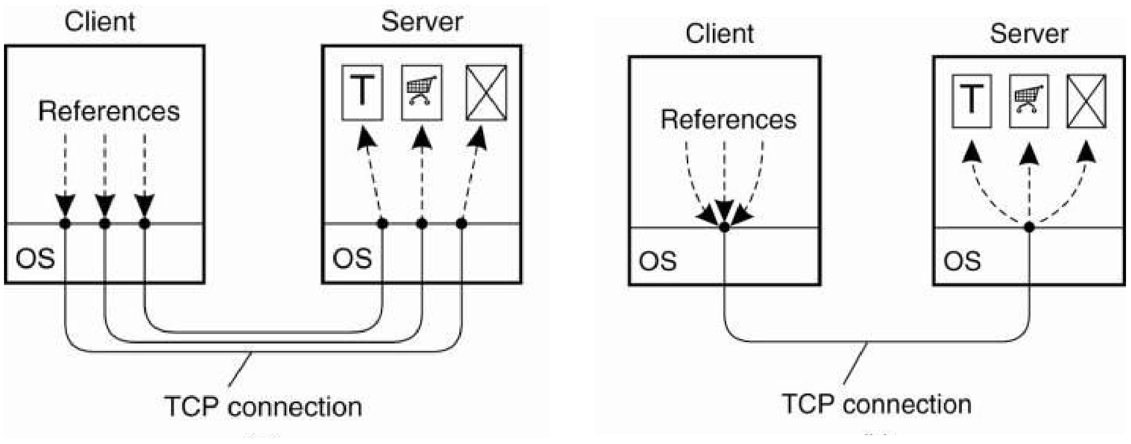
Ans :

Hypertext Transfer Protocol

- All communication in the Web between clients and servers is based on the Hypertext Transfer Protocol (HTTP).
- HTTP is a relatively simple client-server protocol
- A client sends a request message to a server and waits for a response message.
- HTTP is that it is stateless.

HTTP Connections

HTTP is based on TCP. Whenever a client issues a request to a server, it first sets up a TCP connection to the server and then sends its request message on that connection. The same connection is used for receiving the response.



(a) Using nonpersistent connections **(b) Using persistent connections**

Figure

HTTP does not preclude that a client sets up several connections simultaneously to the same server. Another approach that is followed in HTTP version 1.1 is to make use of a persistent connection, which can be used to issue several requests without the need for a separate connection for each (request, response)-pair.

HTTP Methods

HTTP has been designed as a general-purpose client-server protocol oriented toward the transfer of documents in both directions. A client can request each of these operations to be carried out at the server by sending a request message containing the operation desired to the server. A list of the most commonly-used request messages is given in Figure.

Operation	Description
Head	Request to return the header of a document
Get	Request to return a document to the client
Put	Request to store a document
Post	Provide data that are to be added to a document (collection)
Delete	Request to delete a document

Fig.: Operations supported by HTTP.

Head

The head operation is submitted to the server when a client does not want the actual document, but rather only its associated metadata..

GET

The most important operation is get. This operation is used to actually fetch a document from the server and return it to the requesting client. It is also possible to specify that a document should be returned only if it has been modified after a specific time.

PUT

The put operation is the opposite of the get operation. A client can request a server to store a document under a given name .

POST

The operation post is somewhat similar to storing a document, except that a client will request data to be added to a document or collection of documents.

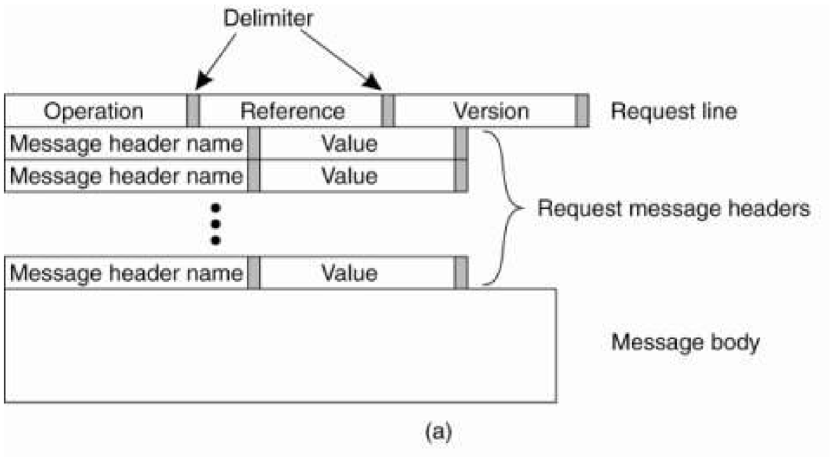
DELETE

Finally, the delete operation is used to request a server to remove the document that is named in the message sent to the server.

HTTP Messages

All communication between a client and server takes place through messages. HTTP recognizes only request and response messages.

A request message consists of three parts, as shown in Figure (a).



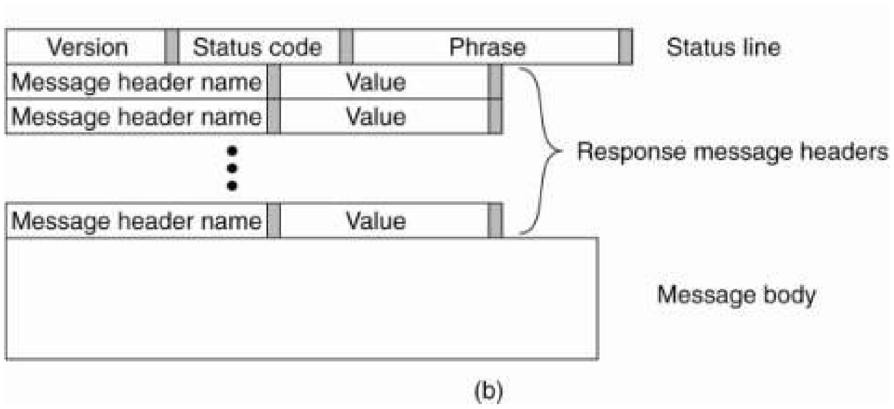


Fig.: (a) HTTP request message. (b) HTTP response message.

A response message starts with a status line containing a version number and also a three-digit status code, as shown in Figure (b).

A request or response message may contain additional headers.

Figure " Some HTTP message headers.

Header	Source	Contents
Accept	Client	The type of documents the client can handle
Accept-Charset	Client	The character sets are acceptable for the client
Accept-Encoding	Client	The document encodings the client can handle
Accept-Language	Client	The natural language the client can handle
Authorization	Client	A list of the client's credentials
WWW-Authenticate	Server	Security challenge the client should respond to
Date	Both	Date and time the message was sent
ETag	Server	The tags associated with the returned document
Expires	Server	The time for how long the response remains valid
From	Client	The client's e-mail address
Host	Client	The DNS name of the document's server
If-Match	Client	The tags the document should have
If-None-Match	Client	The tags the document should not have
If-Modified-Since	Client	Tells the server to return a document only if it has been modified since the specified time
If-Unmodified-Since	Client	Tells the server to return a document only if it has not been modified since the specified time
Last-Modified	Server	The time the returned document was last modified
Location	Server	A document reference to which the client should redirect its request
Referer	Client	Refers to client's most recently requested document
Upgrade	Both	The application protocol the sender wants to switch to
Warning	Both	Information about the status of the data in the message

There are various message headers that the client can send to the server explaining what it is able to accept as response.

Q31. Write a note on SOAP.

Ans :

Simple Object Access Protocol

Where HTTP is the standard communication protocol for traditional Webbased distributed systems, the Simple Object Access Protocol (SOAP) forms the standard for communication with Web services .

SOAP has made HTTP even more important than it already was: most SOAP communications are implemented through HTTP.

A SOAP message generally consists of two parts, header and body

- The header is optional,
- The body contains the actual message,

Everything in the envelope is expressed in XML, that is, the header and the body.

SOAP envelope does not contain the address of the recipient. Instead, SOAP explicitly assumes that the recipient is specified by the protocol that is used to transfer messages. To this end, SOAP specifies bindings to underlying transfer protocols.

4.3.4 Naming

Q32. What are URIs and write about them.

Ans :

The Web uses a single naming system to refer to documents. The names used are called Uniform Resource Identifiers or simply URIs.

URIs come in two forms.

- A Uniform Resource Locator (URL) is a URI that identifies a document by including information on how and where to access the document.
- Uniform Resource Name (URN) acts as true identifier. A URN is used as a globally unique, location-independent, and persistent reference to a document.

The actual syntax of a URI is determined by its associated scheme.

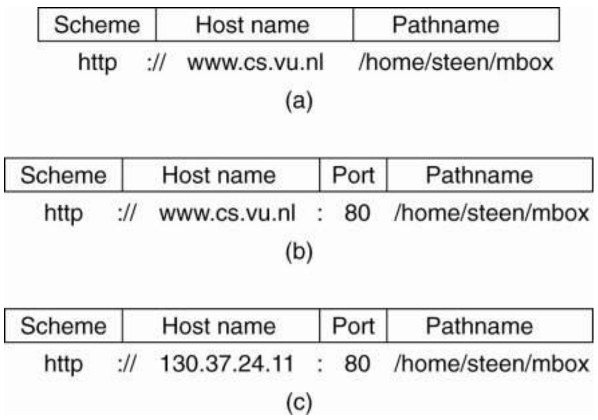

Scheme
Host name
Pathname
http :// www.cs.vu.nl /home/steen/mbox
(a)
Scheme
Host name
Port
Pathname
http :// www.cs.vu.nl : 80 /home/steen/mbox
(b)
Scheme
Host name
Port
Pathname
http :// 130.37.24.11 : 80 /home/steen/mbox
(c)

Fig. : Often-used structures for URLs. (a) Using only a DNS name. (b) Combining a DNS name with a port number. (c) Combining an IP address with a port number.

Although URLs are still commonplace in the Web, various separate URI name spaces have been proposed for other kinds of Web resources. Fig. shows a number of examples of URIs.

Name	Used for	Example
http	HTTP	http://www.cs.vu.nl:80/globe
mailto	E-mail	mailto:steen@cs.vu.nl
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
File	Local file	file:/edu/book/work/chp/11/11
Data	Inline data	data:text/plain;charset=iso-8859-7,%e1%e2%e3
telnet	Remote login	telnet://flits.cs.vu.nl
Tel	Telephone	tel:+31201234567
Modem	Modem	modem:+31201234567;type=v32

Fig.: Examples of URIs

URIs are often used as well for purposes other than referring to a document. For example, a telnet URI is used for setting up a telnet session to a server.

4.3.5 Synchronization

Q33. Write about synchronization in distributed web based systems.

Ans :

Synchronization has not been much of an issue for most traditional Webbased systems for two reasons.

- First, the strict client-server organization of the Web, in which servers never exchange information with other servers (or clients with other clients) means that there is nothing much to synchronize.
- Second, the Web can be considered as being a read-mostly system. Updates are generally done by a single person or entity, and hardly ever introduce write-write conflicts.

Distributed authoring of Web documents is handled through a separate protocol, namely WebDAV. WebDAV stands for Web Distributed Authoring and Versioning and provides a simple means to lock a shared document, and to create, delete, copy, and move documents from remote Web servers.

To synchronize concurrent access to a shared document, WebDAV supports a simple locking mechanism. There are two types of write locks. An exclusive write lock can be assigned to a single client, and will prevent any other client from modifying the shared document while it is locked.

4.3.6 Consistency and Replication

Q34. Write about Web Proxy Caching.

Ans:

Web Proxy Caching

Client-side caching generally occurs at two places.

- In the first place, most browsers are equipped with a simple caching facility. Whenever a document is fetched it is stored in the browser's cache from where it is loaded the next time.
- In the second place, a client's site often runs a Web proxy.

In addition to caching at browsers and proxies, it is also possible to place caches that cover a region, or even a country, thus leading to hierarchical caches. Such schemes are mainly used to reduce network traffic

As an alternative to building hierarchical caches, one can also organize caches for cooperative deployment as shown in Figure.

In cooperative caching or distributed caching, whenever a cache miss occurs at a Web proxy, the proxy first checks a number of neighboring proxies to see if one of them contains the requested document. If such a check fails, the proxy forwards the request to the Web server responsible for the document.

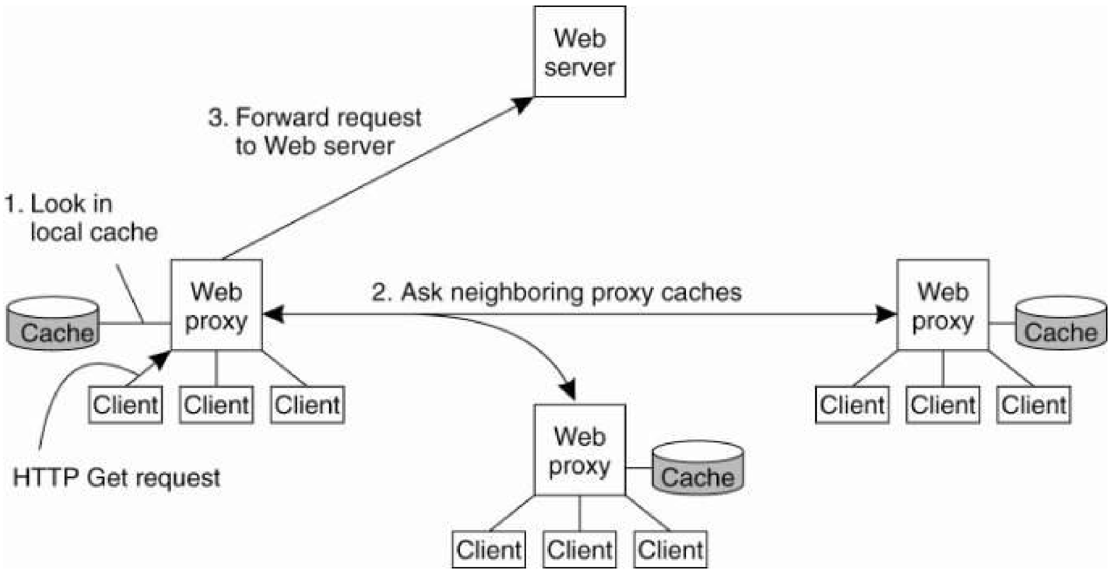


Fig.: The principle of cooperative caching

Different cache-consistency protocols have been deployed in the Web.

Note that documents that have not been modified for a long time will not be checked for modifications as soon as recently modified documents.

Q35. Write about Content Delivery Networks.

Ans :

A content delivery network (CDN) is a system of distributed servers (network) that deliver pages and other Web content to a user, based on the geographic locations of the user, the origin of the webpage and the content delivery server.

This service is effective in speeding the delivery of content of websites with high traffic and websites that have global reach. The closer the CDN server is to the user geographically, the faster the content will be delivered to the user. CDNs also provide protection from large surges in traffic.

How CDNs Work

Servers nearest to the website visitor respond to the request. The content delivery network copies the pages of a website to a network of servers that are dispersed at geographically different locations, caching the contents of the page. When a user requests a webpage that is part of a content delivery network, the CDN will redirect the request from the originating site's server to a server in the CDN that is closest to the user and deliver the cached content. CDNs will also communicate with the originating server to deliver any content that has not been previously cached.

The process of bouncing through CDNs is nearly transparent to the user. The only way a user would know if a CDN has been accessed is if the delivered URL is different than the URL that has been requested.

4.3.7 Fault Tolerance

Q36. Write about the fault tolerance in web based distributed systems.

Ans :

- Fault tolerance in the Web-based distributed systems is mainly achieved through client-side caching and server replication.
- No special methods are used .
- HTTP to assist fault tolerance or recovery.
- High availability in the Web is achieved through such as DNS.

In the case of Web services we may easily be dealing with complex calling graphs.

Many Web-based systems computing follows a simple two-tiered client-server calling convention. This means that a client calls a server, which then computes a response without the need of additional external services.

This situation no longer holds for Web services.

Byzantine fault-tolerant (BFT) serviceface three issues that need to be handled.

- First, clients of a BFT service should see that service as just another Web service.
- Second, a BFT service should guarantee internal consistency when acting as a client.
- Finally, external services should also treat a BFT service acting as a client, as a single entity.

These three situations lead to three different pieces of software that need to be integrated into toolkits for developing Web services.

4.3.8 Security

Q37. Explain about Security in web based distributed systems.

Ans :

- Most of the security issues in the Web deal with setting up a secure channel between a client and server.
- For setting up a secure channel in the Web is to use the Secure Socket Layer (SSL),
- An update of SSL now referred to as the Transport Layer Security (TLS) protocol

TLS is shown in Fig.

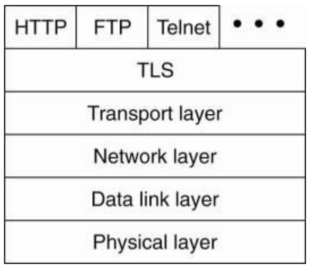


Fig.: The position of TLS in the Internet protocol stack

TLS itself is organized into two layers. The core of the protocol is formed by the TLS record protocol layer, which implements a secure channel between a client and server.

Setting up a secure channel proceeds in two phases.

First, the client informs the server of the cryptographic algorithms it can handle, as well as any compression methods it supports. The actual choice is always made by the server, which reports its choice back to the client. These first two messages shown in Figure.

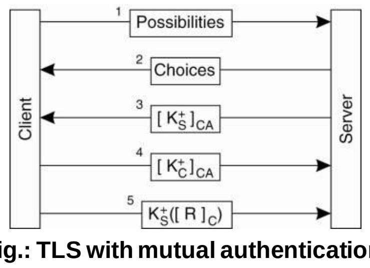


Fig.: TLS with mutual authentication

In the second phase, authentication takes place. The server is always required to authenticate itself, for which reason it passes the client a certificate containing its public key signed by a certification authority CA.