DISTRIBUTED SYSTEMS

SYLLABUS

UNIT - I

Introduction: Definition of Distributed Systems, Goals: Connecting Users and Resources, Transparency, Openness, Scalability, Hardware Concepts: Multiprocessors, Homogeneous Multicomputer systems, Heterogeneous Multicomputer systems, Software Concepts: Distributed Operating Systems, Network Operating Systems, Middleware, The client-server model: Clients and Servers, Application Layering, Client-Server Architectures.

UNIT - II

Communication: Layered Protocols, Lower-Level Protocols, Transport Protocols, Higher-Level Protocols, Remote Procedure Call: Basic RPC Operation, Parameter Passing. Extended RPC Models, Remote Object Invocation: Distributed Objects, Binding a Client to an Object; Static verses Dynamic Remote Method Invocations, Parameter Passing, Message Oriented Communication: Persistence and synchronicity in Communication, Message-Oriented Transient Communication, Message-Oriented' Persistent Communication, Stream Oriented Communication: Support for Continuous Media, Streams and Quality of Service, Stream Synchronization.

UNIT - III

Process: Threads: Introduction to Threads, Threads in Distributed Systems, Clients: user Interfaces, Client-Side Software for Distribution Transparency, Servers: General Design Issues, Object Servers, Software Agents: Software Agents in Distributed Systems, Agent Technology, Naming: Naming Entities: Names, Identifiers, and Address, Name Resolution, The Implementation of a Name System, Locating Mobile Entities: Naming verses Locating Entities, Simple Solutions, Home-Based Approaches, Hierarchical Approaches

UNIT - IV

Distributed Object based Systems: CORBA: Overview of CORBA, Communication, Processes, Naming, Synchronization, Caching and Replication, Fault Tolerance, Security, Distributed COM: Overview of DCOM, Communication, Processes, Naming, Synchronization, Replication, Fault Tolerance, Security, GLOBE: Overview of GLOBE, Communication, Process, Naming, Synchronization, Replication, Fault Tolerance, Security, Comparison of COREA, DCOM, and Globe: Philosophy. Communication. Processes. Naming. Synchronization. Caching and Replication, Fault Tolerance. Security.

UNIT - V

Distributed Multimedia Systems: Introduction. Characteristics of Multimedia Data. Quality of Service Management: Quality of Service negotiation. Admission Control. Resource Management: Resource Scheduling.

Contents

FACULTY OF INFORMATICS

M.C.A II Year - IV Semester Examination

MODEL PAPER - I

DISTRIBUTED SYSTEMS

Time : 3 Hours] *[Max. Marks : 70*

ANSWERS

1. a) What is a distributed system? Explain its importance with examples. **(Unit - I, Q.No.1,2)**
   b) Write a short note on Distribution Transparency and Degree of transparency. **(Unit - I, Q.No.5)**

   (Or)

   c) Write about multi processor and multi computer architectures. **(Unit - I, Q.No. 9,10)**
   d) Write about application layering in client server model. **(Unit - I, Q.No. 18)**
2. a) Write a short note on Layered protocols in OSI Model. **(Unit - II, Q.No. 2)**
   b) What is RPC? Explain basic RPC Operations. **(Unit - II, Q.No. 7,8)**

   (Or)

   c) Write about message oriented transient communication. **(Unit - II, Q.No. 18)**
   d) Explain about Streams and Quality of service (QOS). **(Unit - II, Q.No. 23)**
3. a) What is a Thread? Explain about threads in non distributed systems. **(Unit - III, Q.No. 1,2)**
   b) What is server? Explain general design issues of a server. **(Unit - III, Q.No. 6)**

   Or

   c) Write a short note on object servers. **(Unit - III, Q.No. 7)**
   d) Explain about the role of Software Agents in networks. **(Unit - III, Q.No. 8)**
4. a) Give examples for distributed object systems. **(Unit - IV, Q.No. 1)**
   b) What is CORBA? Explain about naming, processes and synchronization in CORBA. **(Unit - IV, Q.No. 2,10,11)**

   Or

   c) Explain about fault tolerance and security in CORBA. **(Unit - IV, Q.No. 16)**
   d) What is DCOM? Explain about naming, fault tolerance in DCOM. **(Unit - IV, Q.No. 15)**
5. a) Write a short note on distributed multimedia sytems. **(Unit - V, Q.No. 1)**
   b) Write about Quality service management. **(Unit - V, Q.No. 4)**

   Or

   c) Explain about admission control in multimedia systems. **(Unit - V, Q.No. 8)**
   d) Explain Fair scheduling and Real time scheduling. **(Unit - V, Q.No. 9)**

FACULTY OF INFORMATICS

M.C.A II Year - IV Semester Examination

MODEL PAPER - II

DISTRIBUTED SYSTEMS

Time : 3 Hours] [Max. Marks : 70

ANSWERS

1. a) Explain the role of distributed systems? Explain its characteristics. (Unit - I, Q.No.1,3)
   b) What is openness in distribution system? Explain its advantages. (Unit - I, Q.No.6)

Or

   c) Write about homogenous and heterogeneous systems. (Unit - I, Q.No.11,12)
   d) What is middle ware? Explain its sub areas. (Unit - I, Q.No.15,16)
2. a) Write a short note on Transport protocols. (Unit - II, Q.No.5)
   b) What is Remote Object Invocation? Explain about distributed objects. (Unit - II, Q.No.11,12)

Or

   c) What is message oriented communication? Write about message oriented persistent communication. (Unit - II, Q.No.16,19)
   d) Explain about Stream oriented communication and data streams. (Unit - II, Q.No.20,22)
3. a) Explain about Multi threaded clients and servers distributed systems. (Unit - III, Q.No.3)
   b) Write about networked user interfaces with examples. (Unit - III, Q.No.6)

Or

   c) Write about implementation of a name system. (Unit - III, Q.No.13)
   d) Explain about Home based and hierarchical approaches. (Unit - III, Q.No.16,17)
4. a) Explain about the basic components of CORBA. (Unit - IV, Q.No.3)
   b) Write about DCOM Object model. (Unit - IV, Q.No.13,14, 5)

Or

   c) Explain different types of GLOBE objects. (Unit - IV, Q.No.16)
   d) Compare CORBA, DCOM and GLOBE models. (Unit - IV, Q.No.17)
5. a) Write a short note on applications in multimedia. (Unit - V, Q.No.2)
   b) Write about Qos Manager's responsibilities. (Unit - V, Q.No.5)

Or

   c) Explain about Qos parameters for streams. (Unit - V, Q.No.7)
   d) Explain bandwidth reservation and statistical multiplexing. (Unit - V, Q.No.8)

FACULTY OF INFORMATICS

M.C.A II Year - IV Semester Examination

MODEL PAPER - III

DISTRIBUTED SYSTEMS

Time : 3 Hours] *[Max. Marks : 70*

ANSWERS

1. a) Write about scalability types and problems with scalability. (Unit - I, Q.No.7)
   b) Write a short note on Distributed Operating Systems. (Unit - I, Q.No.12)

   Or

   c) What are the advantages and disadvantages network operating systems. (Unit - I, Q.No.13,14)
   d) Write about multi tiered architectures in client server model. (Unit - I, Q.No.19)
2. a) Write a short note on Lower level and higher level protocols. (Unit - II, Q.No.3,6)
   b) Explain about Basic RPC and extended RPC Models. (Unit - II, Q.No.7,10)

   Or

   c) Explain static Remote Method Invocation Vs Dynamic Remote Method Invocation. (Unit - II, Q.No.14)
   d) Explain about
      i) continuous media ii) Stream synchronization. (Unit - II, Q.No.21,24)
3. a) Explain Threads vs Process. Write about thread in distributed systems. (Unit - III, Q.No.1,3)
   b) Write about client software for distribution transparency. (Unit - III, Q.No.5)

   Or

   c) Write about names, identifiers and addresses. (Unit - III, Q.No.10,11)
   d) Explain about Locating mobile entities in naming system. (Unit - III, Q.No.14)
4. a) Explain about the role of distributed objects in distributed systems. (Unit - IV, Q.No.1)
   b) Write about basic architecture of CORBA (Unit - IV, Q.No.4,5)

   Or

   c) Explain basic GLOBE object model. (Unit - IV, Q.No.16)
   d) Compare CORBA, DCOM and GLOBE models. (Unit - IV, Q.No.17)
5. a) What are the different characteristics of Multimedia data? (Unit - V, Q.No.3)
   b) Write about Qos negotiation parameters. (Unit - V, Q.No.5,6)

   Or

   c) Explain about admission control in multimedia. (Unit - V, Q.No.8)
   d) Explain resource management and resource schedulings. (Unit - V, Q.No.9)

UNIT I

Introduction: Definition of Distributed Systems, Goals: Connecting Users and Resources, Transparency, Openness, Scalability, Hardware Concepts: Multiprocessors, Homogeneous Multicomputer systems, Heterogeneous Multicomputer systems,

Software Concepts : Distributed Operating Systems, Network Operating Systems, Middleware, The client-server model: Clients and Servers, Application Layering, Client-Server Architectures.

Introduction to Distributed System

- Computer systems are undergoing a revolution. From 1945, when the modern Computer era began, until about 1985, computers were large and expensive.
- Starting around the mid-1980s, however, two advances in technology began to change that situation.
- The first was the development of powerful microprocessors. Initially, these were 8-bit machines, but soon 16-, 32-, and 64-bit CPUs became common.
- The second development was the invention of high-speed computer networks.
- Local-area networks or LANs allow hundreds of machines within a building to be connected in such a way that small amounts of information can be transferred between machines in a few microseconds or so.
- Larger amounts of data can be moved between machines at rates of 100 million to 10 billion bits/sec.
- Wide-area networks or WANs allow millions of machines all over the earth to be connected at speeds varying from 64 Kbps (kilobits per second) to gigabits per second.
- The result of these technologies is that it is now not only feasible, but easy, to put together computing systems composed of large numbers of computers connected by a high-speed network.
- They are usually caned computer networks or distributed systems, in contrast to the previous centralized systems (or single processor systems) consisting of a single computer, and perhaps some remote terminals.

1.1 Definition of a Distributed System

Q1. What is a distributed system?

Ans :

"A distributed system is a collection of independent computers that appear to the users of the system as a single system."

- This definition has several important aspects:
- The first one is that a distributed system consists of components (i.e., computers) that are autonomous.
- A second aspect is that users (be they people or programs) think they are dealing with a single system. This means that one way or the other the autonomous components need to collaborate.

"A distributed system is one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages."

Our definition of distributed systems has the following significant consequences:

Concurrency : In a network of computers, concurrent program execution is the norm. I can do my work on my computer while you do your

work on yours, sharing resources such as web pages or files when necessary.

No global clock : When programs need to cooperate they coordinate their actions by exchanging messages. Close coordination often depends on a shared idea of the time at which the programs' actions occur. But it turns out that there are limits to the accuracy with which the computers in a network can synchronize their clocks – there is no single global notion of the correct time.

Independent failures: All Distributed systems can fail in new ways. Faults in the network result in the isolation of the computers that are connected to it, but that doesn't mean that they stop running. In fact, the programs on them may not be able to detect whether the network has failed or has become unusually slow.

1.2 Examples of Distributed Systems

Q2. Expalin about distributed systems with examples.

Ans :

As we know, distributed systems encompass many of the most significant technological developments of recent years and hence an understanding of the underlying technology is absolutely central to knowledge of modern computing.

As mentioned, networks are everywhere and underpin many everyday services that we now take for granted: the Internet and the associated World Wide Web, web search, online gaming, email, social networks, e-Commerce, etc

There are more specific examples of distributed systems to further illustrate the diversity and indeed complexity of distributed systems provision today.

1. Web search

- **Web search** has emerged as a major growth industry in the last decade, with recent figures indicating that the global number of searches has risen to over 10 billion per calendar month.
- The task of a web search engine is to index the entire contents of the
- World Wide Web, encompassing a wide range of information styles including web pages, multimedia sources and (scanned) books.
- This is a very complex task, as current estimates state that the Web consists of over 63 billion pages and one trillion unique web addresses.
- Given that most search engines analyze the entire web content and then carry out sophisticated processing on this enormous database, this task itself represents a major challenge for distributed systems design.
- **Google,** the market leader in web search technology, has put significant effort into the design of a sophisticated distributed system infrastructure to support search.
- This represents one of the largest and most complex distributed systems installations in the history of computing.

Highlights of this searching infrastructure include:

- an underlying physical infrastructure consisting of very large numbers of networked computers located at data centers all around the world;
- a distributed file system designed to support very large files and heavily optimized for the style of usage required by search and other Google applications (especially reading from files at high and sustained rates);
- an associated structured distributed storage system that offers fast access to very large datasets;
- a lock service that offers distributed system functions such as distributed locking and agreement;
- a programming model that supports the management of very large parallel and distributed computations across the underlying physical infrastructure.

2. Massively multiplayer online games (MMOGs)

- Massively multiplayer online games offer an immersive experience whereby very large numbers of users interact through the Internet with a virtual world.
- Leading examples of such games include Sony's **EverQuest II** and **EVE Online** from the Finnish company CCP Games.
- Such worlds have increased significantly in sophistication and now include, complex playing arenas (for example EVE, Online consists of a universe with over 5,000 star systems) and multifarious social and economic systems.
- The number of players is also rising, with systems able to support over 50,000 simultaneous online players.
- The engineering of MMOGs represents a major challenge for distributed systems technologies, particularly because of the need for fast response times to reserve the user experience of the game.
- Other challenges include the real-time propagation of events to the many players and maintaining a consistent view of the shared world. This therefore provides an excellent example of the challenges facing modern distributed systems designers.

A number of solutions have been proposed for the design of massively multiplayer online games :

- Perhaps surprisingly, the largest online game, EVE Online, utilizes a **client-server** architecture where a single copy of the state of the world is maintained on a centralized server and accessed by client programs running on players' consoles or other devices.

  To support large numbers of clients, the server is a complexentity in its own right consisting of a cluster architecture featuring hundreds of computer nodes.

  The centralized architecture helps significantly in terms of the management of the virtual world and the single copy also eases consistency concerns.
- Other MMOGs adopt more distributed architectures where the universe is partitioned across a (potentially very large) number of servers that may also be geographically distributed.

  Users are then dynamically allocated a particular server based on current usage patterns and also the network delays to the server(based on geographical proximity for example).

  This style of architecture, which is adopted by Ever Quest, is naturally extensible by adding new servers.
- Most commercial systems adopt one of the two models presented above, but researchers are also now looking at more radical architectures that are not based on client-server principles but rather adopt completely decentralized approaches based on peer-to-peer technology where every participant contributes resources(storage and processing) to accommodate the game.

3. Financial Trading

- Another example of distributed systems is, support for **financial trading** markets.
- The financial industry has long been at the cutting edge of distributed systems technology with its need, in particular, for real-time access to a wide range of information sources.
- For example, current share prices and trends, economic and political developments. The industry employs automated monitoring and trading applications.
- The emphasis in such systems is on the communication and processing of items of interest, known as events in distributed systems, with the need also to deliver events reliably and in a timely manner to very large numbers of clients who have a stated interest in such information items.

- Examples of such events include a drop in a share price, the release of the latest unemployment figures, and so on.
- This requires a very different style of underlying architecture from the styles mentioned above (for example client-server), and such systems typically employ what are known as distributed event-based systems.

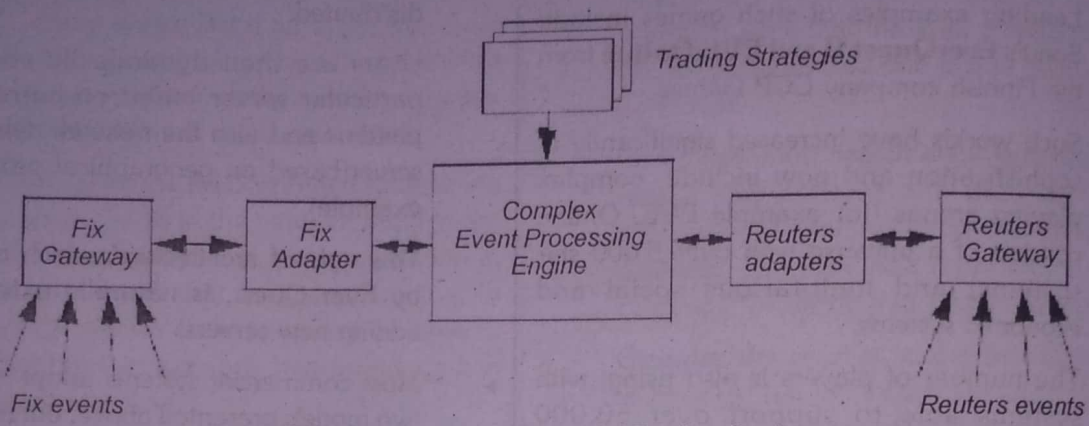


Fig. : An example of Financial Trading System

This approach is primarily used to develop customized algorithmic trading strategies covering both buying and selling of stocks and shares, in particular looking for patterns that indicate a trading opportunity and then automatically responding by placing and managing orders.

1.3 Important Characteristics of Distributed Systems

Q3. Explain, characteristics of distributed systems.

Ans :

- One important characteristic is that differences between the various computers and the ways in which they communicate are mostly hidden from users.
- Another important characteristic is that users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction takes place.
- In principle, distributed systems should also be relatively easy to expand or scale.
- A distributed system will normally be continuously available, although perhaps some parts may be temporarily out of order.
- Users and applications should not notice that parts are being replaced or fixed, or that new parts are added to serve more users or applications.

In order to support heterogeneous computers and networks while offering a single-system view, distributed systems are often organized by means of a layer of software-that is, logically placed between a higher-level layer consisting of users and applications, and a layer underneath consisting of operating systems and basic communication facilities, as shown in Fig. Accordingly, such a distributed system is sometimes called middleware.

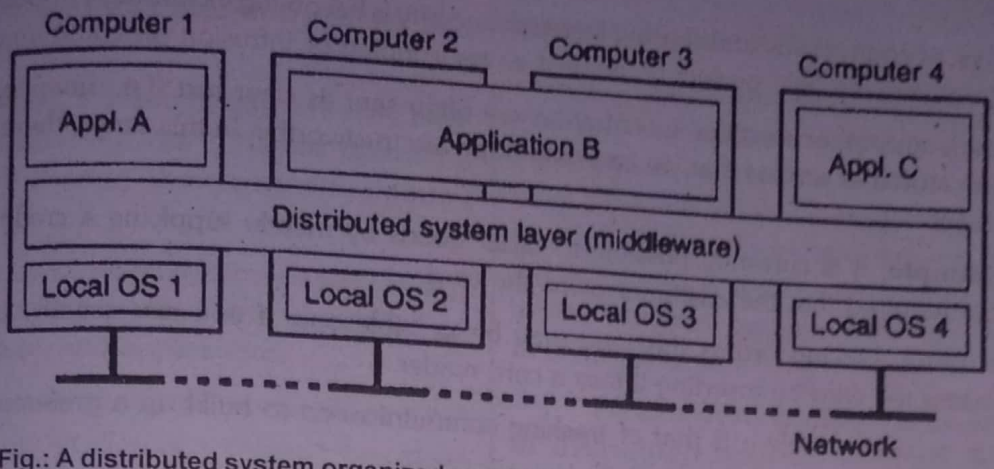


Fig.: A distributed system organized as middleware The middleware layer extends over multiple machines, and offers each application the same in terface

Above Fig., shows four networked computers and three applications, of which application B is distributed across computers 2 and 3.

Each application is offered the same interface. The distributed system provides the means for components of a single distributed application to communicate with each other, but also to let different applications communicate.

At the same time, it hides, as best and reasonable as possible, the differences in hardware and operating systems from each application.

1.4 GOALS

Q4. Write about Connecting Users and Resources ?

Ans :

A distributed system should make resources easily accessible; it should reasonably hide the fact that resources are distributed across a network; it should be open; and it should be scalable.

There are four important goals that should be met to make building a distributed system worth the effort.

Connecting Users and Resources

The main goal of a distributed system is to make it easy for the users (and applications) to access remote resources, and to share them in a controlled and efficient way.

Resources can be just about anything, but typical examples include things like printers, computers, storage facilities, data, files, Web pages, and networks...etc.

Connecting users and resources also makes it easier to collaborate and exchange information, as is clearly illustrated by the success of the Internet with its simple protocols for exchanging files, mail, documents, audio, and video.

The connectivity of the Internet is now leading to numerous virtual organizations in which geographically widely-dispersed groups of people work together by means of groupware, that is, software for collaborative editing, teleconferencing, and soon.

Likewise, the Internet connectivity has enabled electronic commerce allowing us to buy and sell all kinds of goods without actually having to go to a store or even leave home.

However, as connectivity and sharing increase, security is becoming increasingly important. In current practice, systems provide little protection against eavesdropping or intrusion on communication.

Passwords and other sensitive information are often sent as clear text (i.e., unencrypted) through the network, or stored at servers that we can only hope are trustworthy. In this sense, there is much room for improvement.

For example, it is currently possible to order goods by merely supplying a credit card number. Rarely is proof required that the customer owns the card.

In the future, placing orders this way may be possible only if you can actually prove that you physically possess the card by inserting it into a card reader.

Another security problem is that of tracking communication to build up a preference profile of a specific user.

Such tracking explicitly violates privacy, especially if it is done without notifying the user. A related problem is that increased connectivity can also lead to unwanted communication, such as electronic junk mail, often called spam.

In such cases, what we may need is to protect ourselves using special information filters that select incoming messages based on their content.

1.4.1 Distribution Transparency

Q5. Write a short note on Distribution Transparency, Types and its degree.

Ans :

An important goal of a distributed system is to hide the fact that its processes and resources are physically distributed across multiple computers.

A distributed system that is able to present itself to users and applications as if it were only a single computer system is said to be transparent. Let us first take a look at what kinds of transparency exist in distributed systems. After that we will address the more general question whether transparency is always required.

Types of Transparency

The concept of transparency can be applied to several aspects of a distributed system, the most important ones shown in Fig.

Transparency	Description
Access	hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may moved to another location while in use
Replication	Hide that resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Access transparency deals with hiding differences in data representation and the way that resources can be accessed by users. At a basic level, we wish to hide differences in machine architectures, but more

important is that we reach agreement on how data is to be represented by different machines and operating systems.

For example, a distributed system may have computer systems that run different operating systems, each having their own file-naming conventions.

Differences in naming conventions, as well as how files can be manipulated should all be hidden from users and applications.

An important group of transparency types has to do with the location of a resource. **Location transparency** refers to the fact that users cannot tell where a resource is physically located in the system.

Naming plays an important role in achieving location transparency. In particular, location transparency can be achieved by assigning only logical names to resources, i.e, names in which the location of a resource is not secretly encoded.

An example of such a name is the URL **http://www.prenhall.com/index.html**. Which gives no clue about the location of Prentice Hall's main Web server?

The URL also gives no clue as to whether index.html has always been at its current location or was recently moved there.

Distributed systems in which resources can be moved without affecting how those resources can be accessed are said to provide **migration transparency**.

Even stronger is the situation in which resources can be relocated while they are being accessed without the user or application noticing anything. In such cases, the system is said to support **relocation transparency.** An example of relocation transparency is when mobile users can continue to use their wireless laptops while moving from place to place without ever being (temporarily) disconnected.

Replication plays a very important role in distributed systems. For example, resources may be replicated to increase availability or to improve performance by placing a copy close to the place where it is accessed.

Replication transparency deals with hiding the fact that several copies of a resource exist. To hide replication from users, it is necessary that all replicas have the same name. Consequently, a system that supports replication transparency should generally support location transparency as well, because it would otherwise be impossible to refer to replicas at different locations.

We already mentioned that an important goal of distributed systems is to allow sharing of resources. For example, two independent users may each have stored their files on the same file server. This phenomenon is called **concurrency transparency**.

Making a distributed system **failure transparent** means that a user does not notice that a resource (he has possibly never heard of) fails to work properly, and that the system subsequently recovers from that failure.

Degree of Transparency

Although distribution transparency is generally considered preferable for any distributed system, there are situations in which attempting to completely hide all distribution aspects from users are not a good idea.

An example is requesting your electronic newspaper to appear in your mailbox before 7 A.M. local time, as usual, while you are currently at the other end of the world living in a different time zone. Your morning paper will not be the morning paper you are used to.

Likewise, a wide-area distributed system that connects a process in San Francisco to a process in Amsterdam cannot be expected to hide the fact that Mother Nature will not allow it to send a message from one process to the other in less than about 35 milliseconds. In practice it takes several hundreds of milliseconds using a computer network.

Signal transmission is not only limited by the speed of light. But also by limiting processing capacities of the intermediate switches.

There is also a trade-off between a high degree of transparency and the performance of a system. For example, many Internet applications repeatedly try to contact a server before finally giving up.

Consequently, attempting to mask a transient server failure before trying another one may slow down the system as a whole. In such a case, it may have been better to give up earlier, or at least let the user cancel the attempts to make contact.

Another example is where we need to guarantee that several replicas, located on different continents, need to be consistent all the time.

In other words, if one copy is changed, that change should be propagated to all copies before allowing any other operation. It is clear that a single update operation may now even take seconds to complete, something that cannot be hidden from users.

Finally, there are situations in which it is not at all obvious that hiding distribution is a good idea. As distributed systems are expanding to devices that people carry around, and where the very notion of location and context awareness is becoming increasingly important, it may be best to actually *expose* distribution rather than trying to hide it.

The conclusion is that aiming for distribution transparency may be a nice goal when designing and implementing distributed systems, but that it should be considered together with other issues such as performance and comprehensibility. The price for not being able to achieve full transparency may be surprisingly high.

1.4.2 Openness

Q6. Write a note on openness and its advantages.

Ans :

Another important goal of distributed systems is openness. An open distributed system is a system that offers services according to standard rules that describe the syntax and semantics of those services.

For e.g., in computer networks, standard rules govern the format, contents, and meaning of messages sent and received. Such rules are formalized in protocols.

In distributed systems services are generally specified through interfaces, which are often described in an Interface Definition Language (IDL). Interface definitions written in an IDL nearly always capture only the syntax of services.

In other words, they specify precisely the names of the functions that are available together with types of the parameters, return values, possible exceptions that can be raised, and so on. The hard part is specifying precisely what those services do, that is, the semantics of interfaces. In practice, such specifications are always given in an informal way by means of natural language.

If properly specified, an interface definition allows an arbitrary process that needs a certain interface to talk to another process that provides that interface. It also allows two independent parties to build completely different implementations of those interfaces, leading to two separate distributed systems that operate inexactly the same way.

Proper specifications are complete and neutral. Complete means that everything that is necessary to make an implementation has indeed been specified. However, many interface definitions are not at all complete. So that it is necessary for a developer to add implementation-specific details. Just as important is the fact that specifications do not prescribe what an implementation should look like: they should be neutral. Completeness and neutrality are important for interoperability and portability.

Inter operability characterizes the extent by which two implementations of systems or components from different manufacturers can co-exist and work together by merely relying on each other's services as specified by a common standard. Portability characterizes to what extent an application developed for a distributed system A can be executed without modification, on a different distributed system B that implements the same interfaces as A.

Another important goal for an open distributed system is that it should be easy to configure the system out of different components (possibly from different developers). Also, it should

be easy to add new components or replace existing ones without affecting those components that stay in place. In other words, an open distributed system should also be extensible. For example, in an extensible system ,it should be relatively easy to add parts that run on a different operating system. Or even to replace an entire file system. As many of us know from daily practice, attaining such flexibility is easier said than done.

Advantages of open distributed system

- **Interoperability:** Open systems can work together. This feature allows two implementations of system or components from different manufacturers of co-exist and work together. Both of them rely on each other's services or specified by a common standard.
- **Portability:** Portability is an ability to transfor man application from one software or hardware platform to another.

Goals of open distributed systems

- **Flexibility:** Different developers provide different components. So a flexible open distributed system is one which allows easy configuration of the system with different components from different developers.
- **Extensibility:** Secondly a flexible open distributed system allows adding new components to the system, replaces the existing ones. This is done without affecting those components. Those are in their original place.

Separating Policy from Mechanism

To achieve flexibility in open distributed systems, it is crucial that the system is organized as a collection of relatively small and easily replaceable or adaptable components. This implies that we should provide definitions not only for the highest-level interfaces, that is, those seen by users and applications, but also definitions for interfaces to internal parts of the system and describe how those parts interact. This approach is relatively new.

Many older and even contemporary systems are constructed using a monolithic approach in which components are only logically separated but implemented as one huge program. This approach makes it hard to replace or adapt a component without affecting the entire system. Monolithic systems thus tend to be closed instead of open.

The need for changing a distributed system is often caused by a component that does not provide the optimal policy for a specific user or application. As an example, consider caching in the World Wide Web. Browsers generally allow users to adapt their caching policy by specifying the size of the cache, and whether a cached document should always be checked for consistency, or perhaps only once per session.

However, the user cannot influence other caching parameters, such as how long a document may remain in the cache, or which document should be removed when the cache fills up. Also, it is impossible to make caching decisions based on the content of a document. For instance, a user may want to cache railroad timetables, knowing that these hardly change, but never information on current traffic conditions on the highways.

What we need is a separation between policy and mechanism. In the case of

Web caching, for example, a browser should ideally provide facilities for only storing documents, and at the same time allow users to decide which documents are stored and for how long. In practice, this can be implemented by offering a rich set of parameters that the user can set (dynamically). Even better is that a user can implement his own policy in the form of a component that can be plugged into the browser. Of course, that component must have an interface that the browser can understand so that it can call procedures of that interface.

1.4.3 Scalability

Q7. Write about Scalability types and problems with scalability.

Ans :

Worldwide connectivity through the Internet is rapidly becoming as common as being able to send a postcard to anyone anywhere around the world. With this in mind, scalability is one of the most important design goals for developers of distributed systems.

According to **Neuman** 1994 scalability is measured along 3 different dimensions.

- **Scalability with respect to size:** This means that in a scalable distributed system, users and resources can be easily added to the system.
- **Scalability with respect to geographic area:** In a scalable distributed system, the users and the resources may lie geographically at different locations. But still a communication between them is flexible and addition of new users and new resources is possible.
- **Scalability with respect to administration:** A distributed system with this feature allows easy and proper management of the system even if the system is widely split amongst various organizations.

Problems with Scalability

When a system needs to scale, very different types of problems need to be solved. Let us first consider scaling with respect to size. If more users or resources need to be supported, we are often confronted with the limitations of centralized services, data, and algorithms (Fig. 1-3).

Concept	Example
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

Table : Examples of Scalability limitations

For example, many services are centralized in the sense that they are implemented by means of only a single server running on a specific machine in the distributed system. The problem with this scheme is obvious: the server can become a bottleneck as the number of users and applications grows. Even if we have virtually unlimited processing and storage capacity, communication with that server will eventually prohibit further growth.

Unfortunately, using only a single server is sometimes unavoidable. Imagine that we have a service for managing highly confidential information such as medical records, bank accounts and so on. In such cases, it may be best to implement that service by means of a single server in a highly secured separate room, and protected from other parts of the distributed system through special network components. Copying the server to several locations to enhance performance may be out of the question as it would make the service less secure.

Just as bad as centralized services are centralized data. How should we keep track of the telephone numbers and addresses of 50 million people? Suppose that each data record could be fit into 50 characters. A single 2.5-gigabyte disk partition would provide enough storage. But here again, having a single database would undoubtedly saturate all the communication lines into and out of it. Likewise, imagine how the Internet would work if its **Domain Name System (DNS)**was still implemented as a single table. DNS maintains information on millions of computers worldwide and forms an essential service for locating Web servers.

If each request to resolve a URL had to be forwarded to that one and only

DNS server, it is dear that no one would be using the Web.

Finally, centralized algorithms are also a bad idea. In a large distributed system,

An enormous number of messages have to be routed over many lines. From a theoretical point of view, the optimal way to do this is collect complete information about the load on all machines and lines, and then run an algorithm to compute all the optimal routes. This information can then be spread around the system to improve the routing.

The trouble is that collecting and transporting all the input and output information would again be a bad idea because these messages would overload part of the network. In fact, any algorithm that operates by collecting information from all the sites sends it to a single machine for processing, and then distributes the results should generally be avoided. Only decentralized algorithms should be used.

These algorithms generally have the following characteristics, which distinguish them from centralized algorithms :

1. No machine has complete information about the system state.
2. Machines make decisions based only on local information,
3. Failure of one machine does not ruin the algorithm.
4. There is no implicit assumption that a global clock exists.

1.4.3.1 Scaling Techniques

Q8. Explain about various Scaling Techniques in distributed systems.

Ans :

In the distributed systems the scalability problems are actually performance problems. These problems arise because there are limited number of servers and network.

There are three techniques for scaling.

- Hiding communication latencies
- Distribution
- Replication

The techniques must basically satisfy scalability issues. i.e. scalability means that the system should remain efficient with a significant increase in the number of users and resources connected,

a) The cost of adding the resources must be reasonable.

b) Performance loss with increased number of users and resources must be controlled.

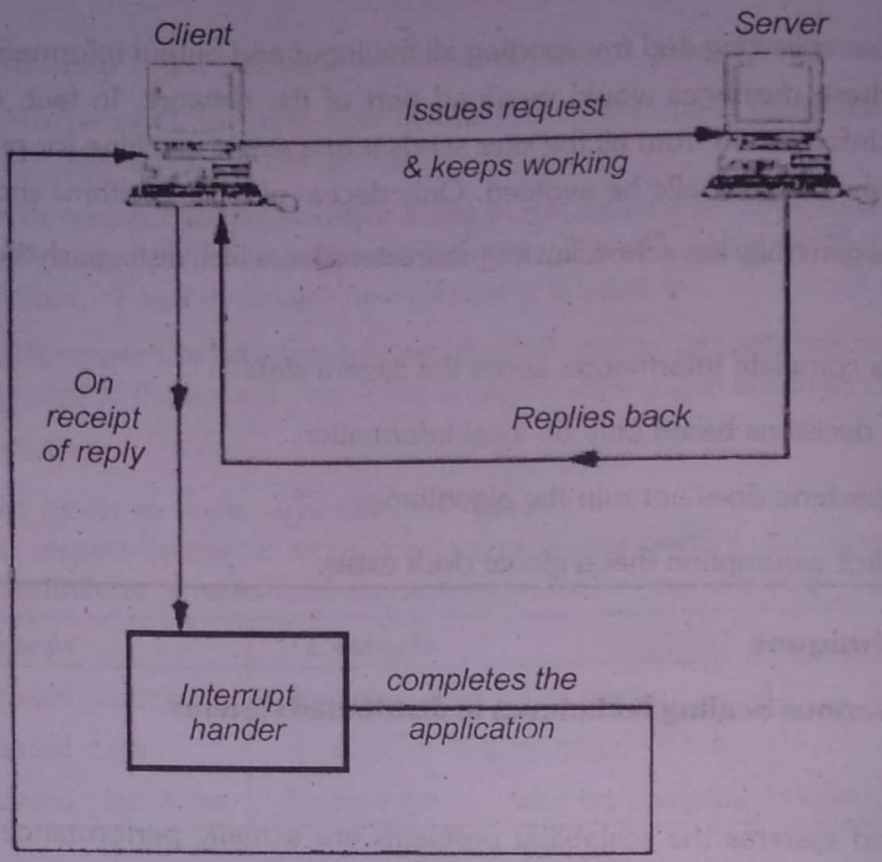
c) Software resources must not run out.

Hiding Communication Latencies

This technique in used when geographical scalability is an issue of concern. In distributed systems which are geographically far located, the communication is mostly synchronous, i.e. the client waits till the server responds.

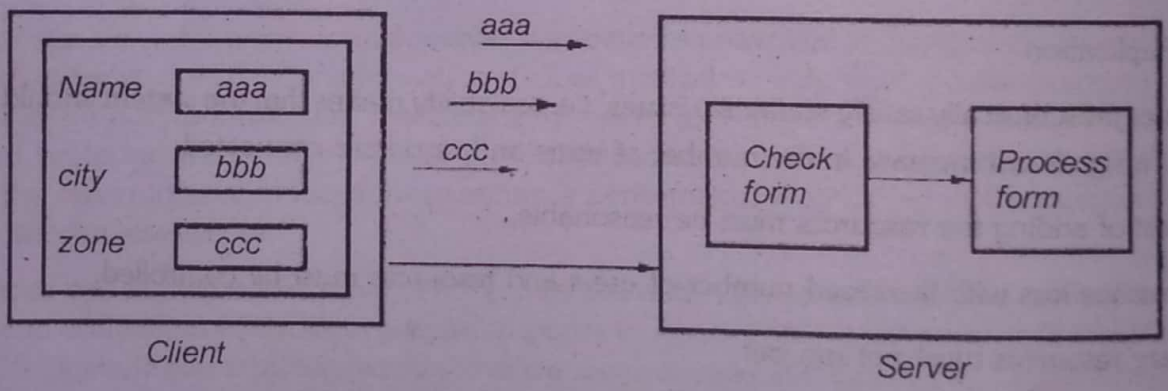
This technique says that we should avoid the waiting for the server's response. This time could be utilized for some efficient work i.e. the system should necessarily use asynchronous communication. In this type of communication, the requesting application does not wait for the reply from the server.

When a reply comes in, the application is interrupted. The previously issued request is completed by a special interrupt handler which is called on receipt of the reply.

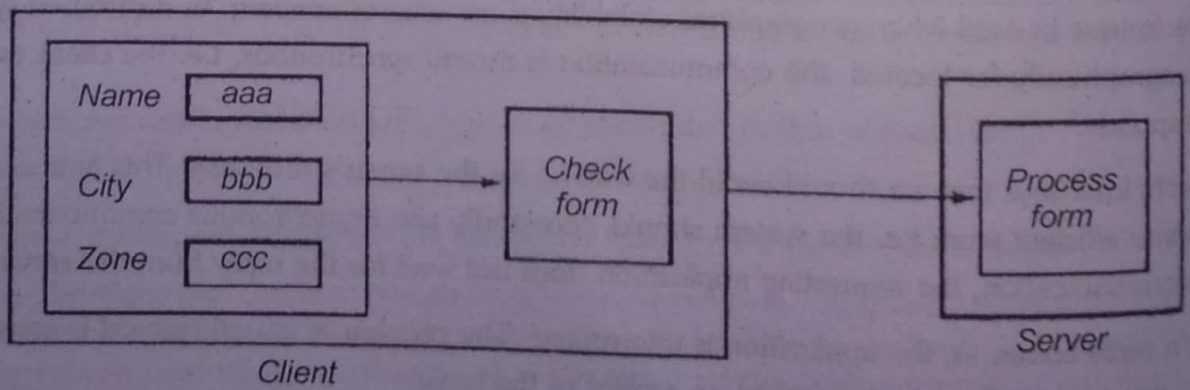


This type of technique is now widely used in internet applications where a client is asked to fill up a form. Previously a server used to check the form. Now java applets are written in such a way that the client only does the checking and hence communication time is reduced.

a) Previous processing



b) Now using java applets



Distribution

As its name, this technique involves selecting a component and then splitting it into smaller parts. These smaller parts are spread around the system, following are the examples.

- **Web :** To a user, the web has n number of document to be processed. This n is a very large number. Practically, there are several handling a particular amount of documents. The document UPL consists the name of the server handling that document. The web appears to be a single sever because of the distribution technique of scalability.
- **DNS :** internet Domain Name System (DNS) is another example of distribution. The organization of DNS is done by hierarchically dividing it into tree of domains. These trees are divided into non overlapping Zones. Each zone has a single name server and the names in each zone are handled by this server.

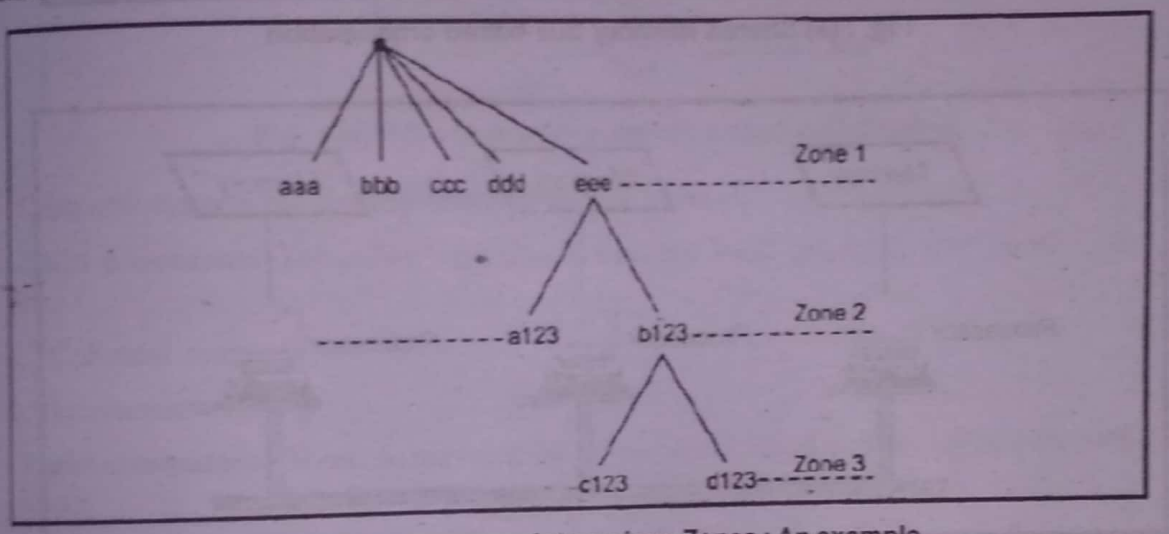


Fig. : DNS name space into various Zones : An example

eee.b123.c123 name is resolved by first passing it to the server of zone 1, then passing it to the server of zone 2 and finally passing it to the server of zone 3 and thus address of the associated is returned.

Replication

Since improper scaling results in performance degradation, it is a better practice to replicate the components in the distributed system. Replication leads to availability of the components to the users at a faster rate thus improving the performance of the system.

1.5 Hardware Concepts

Q9. Explain about Bus based Architecture, Switch Based Architecture.

Ans :

A distributed system consists of various processors (CPUs).There are various ways of organizing these processor i.e. the ways the various processors are connected varies.

Over the years, many classification schemes have been proposed .These schemes are by based on the architecture of the interconnection network.

There are 2 commonly used architectures:

Bus Based Architecture

In this type of architecture there is a single network and a bus cable that connects all the machines. A common example of bus based architecture is cable TV.

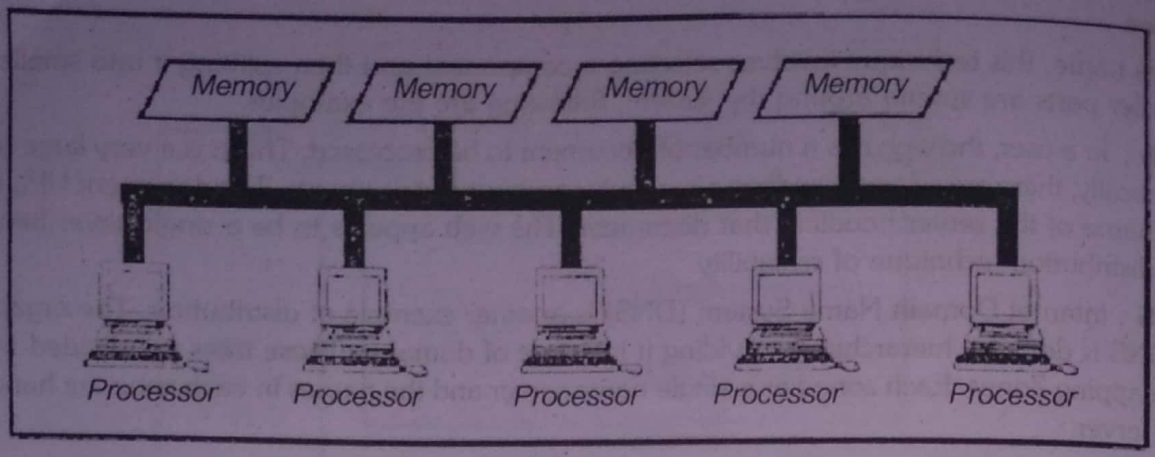


Fig. : (a) Shared memory bus-based organisation

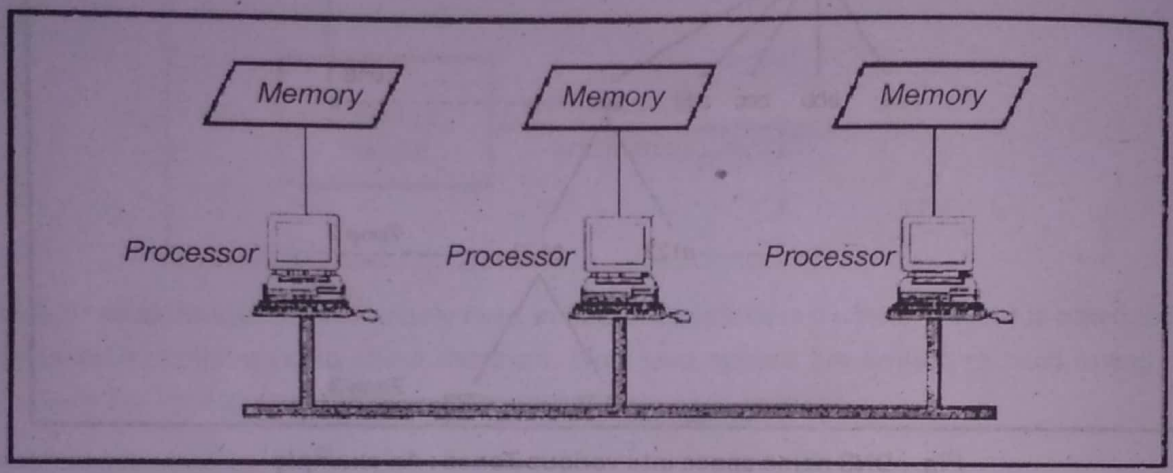


Fig. : (b) Private memory bus-based organisation

Switch Based Architecture

In this architecture, multiple individual wires run from machine to machine with different writing patterns. A common example of this architecture is public telephone lines.

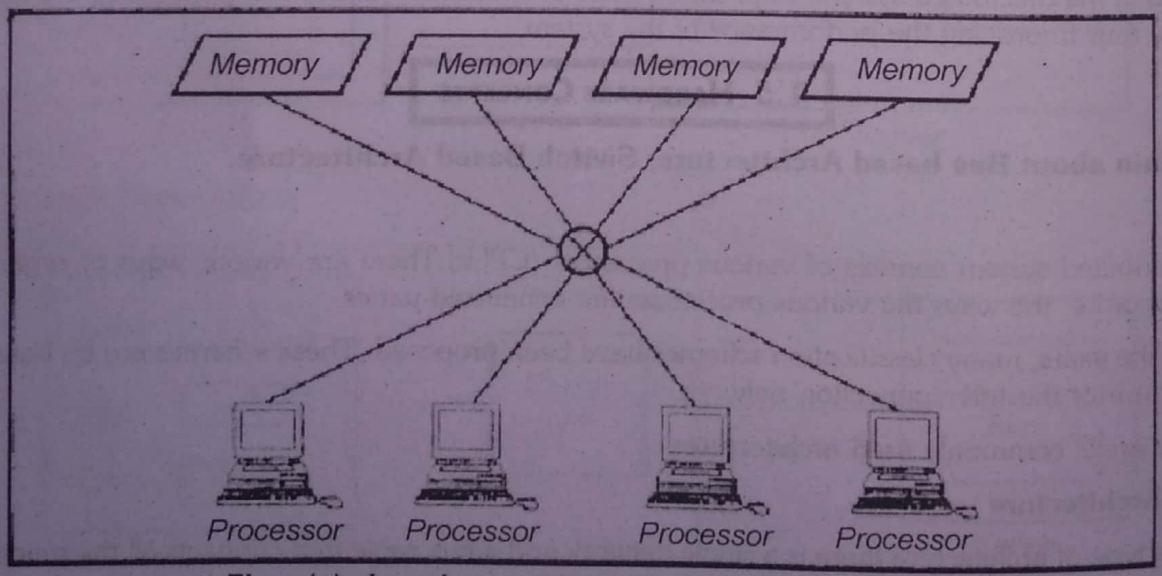


Fig. : (a) shared memory switch based architecture

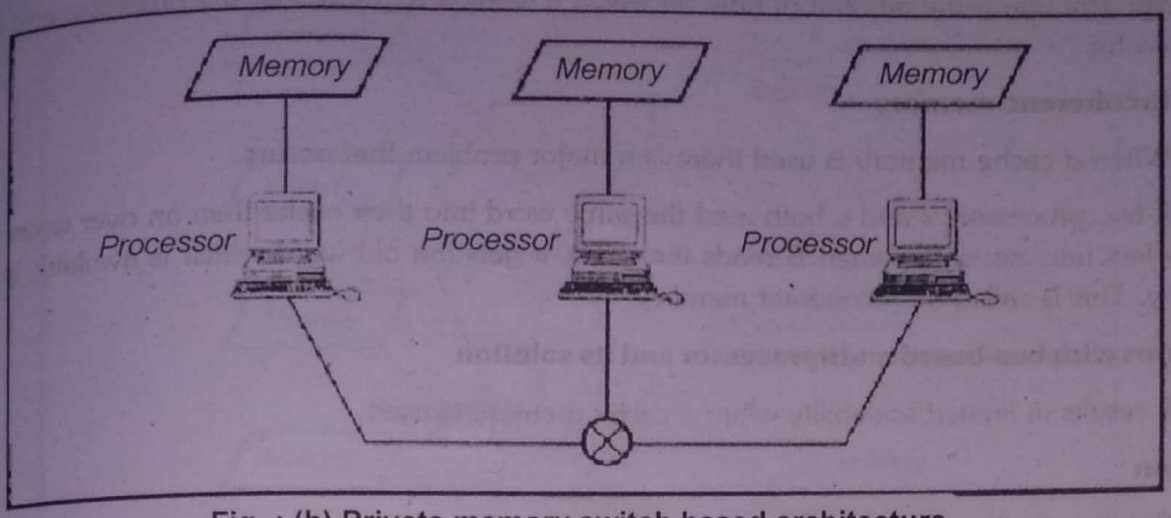


Fig. : (b) Private memory switch based architecture

Computer systems are broadly classified in to 2 types :

1. **Multi processors** -Those that have shared memory. Multi processor systems are further classified into :
   a) Coherent memory
   b) Incoherent memory
2. **Multi computers -** Those do not have shared memory Multi computer systems are further classified into :
   a) Homogeneous multi computers
   b) Heterogeneous multi computers

1.5.1 Multi Processors

Q10. Write about multi processor system.

Ans :

Multi processors are those that have shared memory. In a Multi processor each CPU has direct access to the shared memory.

In bus based architecture, there are number of CPUs and all are connected to the common bus. All have access to shared memory.

a) **Coherent memory**

Since there is a single memory, if a CPU x writes the word to memory and CPU y wants to read back the word after some seconds, it gets the newly written word.

This property of the memory is called the coherent property.

Problems with coherent memory:

If there are many CPU's in the system, the bus will always be overloaded.

Solution :

The simplest solution to this problem is to have a cache memory in between processor and memory. If the processor request for a word and if it is available in the cache memory, it returns the word faster. This increases the efficiency of the system. Cache holds the most recently accessed word.

Hit rate: The rate is the amount of time on which a word is requested by the processor and if is found in the cache.

b) Incoherent memory

When a cache memory is used there is a major problem that occurs.

If two processors x and y both read the same word into their cache then an over writes the word. After a few microseconds, when B reads the word, it gets the old word which is available in its cache memory. This is called an incoherent memory

Problems with bus-based multiprocessor and its solution

It results in limited scalability when a cache memory is used.

Solution :

Solution to this problem is the crossbar switch.

Divide the memory into modules and connect them to the CPU by crossbar switch. Fig 1.14 shows the architecture where each processor and each memory has a connection coming out.

There are several intersection points. A cross point switch can be used at every intersection point. This switch can be opened and closed in the hardware.

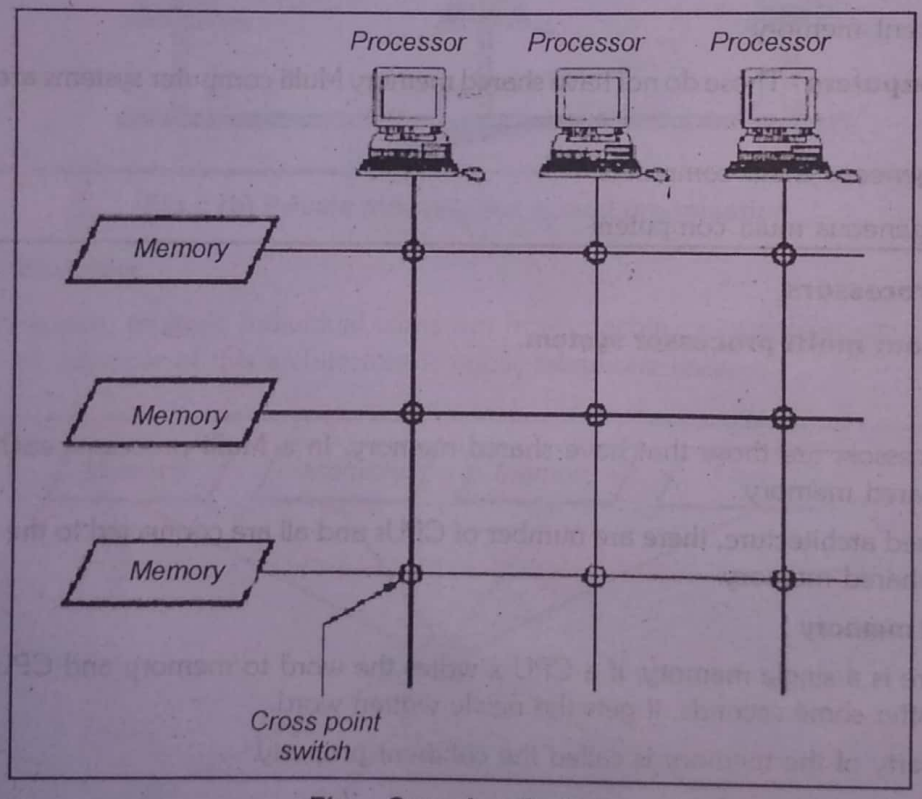


Fig. : Crossbar Switch

The omega network shown in the fig 1.15 consists of 2x2 switches i.e. each switch has 2 inputs and 2 outputs. Now also with proper settings every processor can access every memory. To avoid low latency problems between CPU and memory, switching has to be tremendously fast. This may also lead to overhead cost.

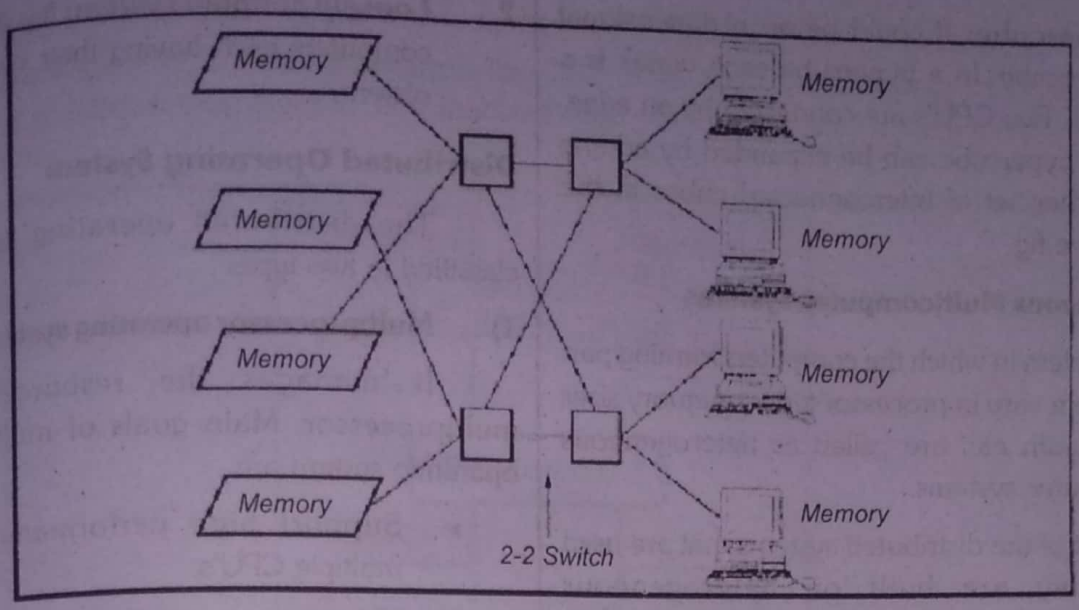


Fig. : An omega switching network

1.5.2 Multi Computers

Q11. Explain about homogenous and hetero-geneous systems.

Ans :

Multi computer systems are further classified into :

a) Homogeneous multi computers

b) Heterogeneous multi computers

a) Homogenous Multicomputer Systems:

Homogenous multicomputer system is also called as **System Area Network (SAN).** In these types of systems the nodes are mounted in a big rack and connected through a high performance single network.

- **Bus-based multicomputer :** In this type of systems shared multi-access network like an Ethernet is used to connect the processors. Messages are broadcasts in bus-based multicomputer systems.
- **Switch based multicomputer :** Instead of broadcasting, in a switch based multicomputer messages between the processors are routed through interconnection network.

Two commonly used topologies are :

Grid : Grid structure is shown in the fig, they are easy to lay down on PCB's. The grid structure is mainly used in applications that have a two dimensional nature. Applications like graph theory and robotics (visions) are the areas where a grid structure is used.

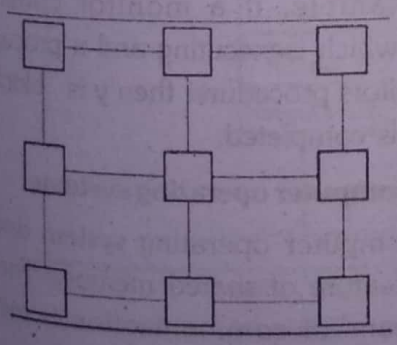


Fig.: A typical gridd structure

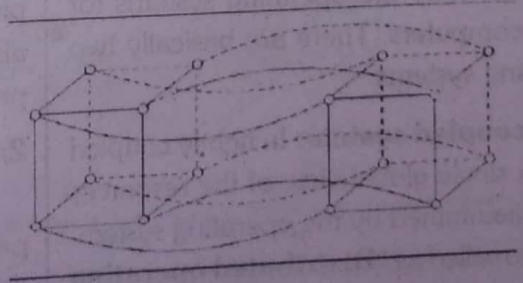


Fig.: Four dimensional hypercube

a) **Hypercube:** It could be an n dimensional hypercube. In a hypercube each vertex is a CPU. Two CPU's are connected by an edge. The hypercube can be expanded by adding another set of interconnected cubes in the above fig.

Heterogeneous Multicomputer Systems

A system in which the computers forming part of the system vary in processor types, memory sizes I/O bandwidth etc. are called as heterogeneous multicomputer systems.

Most of the distributed systems that are used now a day are built on heterogeneous multicomputer systems. The interconnection network used may also be heterogeneous.

1.6 Software Concepts

Q12. Write a short note on distributed operating systems.

Ans :

It is not only the hardware that is important for a distributed system, but the software also plays a major role in the system.

Distributed systems exhibit the properties of operating systems.

- It hides the differences of the various machines and networks in the distributed system just like O.S. hides the features from the user.
- Distributed systems, like OS acts as resource managers for the hardware. It allows various users and applications to share the resources.

So, we will study the operating systems for the distributed computers. There are basically two types of operating systems.

1. **Tightly coupled system:** In tightly coupled system, a single global view of the resources is being maintained by the operating system. This is also called as "**Distributed operating system**".
2. **Loosely coupled system:** It is a collection of computers each having their own operating system.

Distributed Operating System

The distributed operating systems are classified in two types :

1) Multiprocessor operating system

It manages the resources of the multiprocessor. Main goals of multiprocessor operating system are

- Support high performance through multiple CPU's.
- Make the number of CPU's transparent to the application.

Two important primitives that are used for protection are synchronization primitives. They are

i) Semaphores

Semaphore is an integer which has basically two values a 0 and a 1. A 0, blocks the calling process whereas a 1 value of the semaphore checks the blocked processes and then unblocks one of the process and then continues.

Once the semaphore operation is started, no other process can access the semaphore until the operation is completed.

ii) Monitor

A monitor is like a programming language construct. It is a module consisting of variables and procedures. Monitors procedure can access the variables. A monitor allows a single process at a time to execute.

For example, if a monitor consists of procedure x which isexecuting and a procedure y also call monitors procedure, then y is blocked till procedure x is completed.

2) Multicomputer operating systems

Multicomputer operating system does not provide the feature of shared memory. Therefore the only means of communication is message passing.

The organization of multicomputer O.S. is shown in Fig.below.

In multicomputer O.S. the local resources like CPU, memory and local disk are being managed by the nodes or machines own Kernel. The machine also has a separate module for inter processor communication.

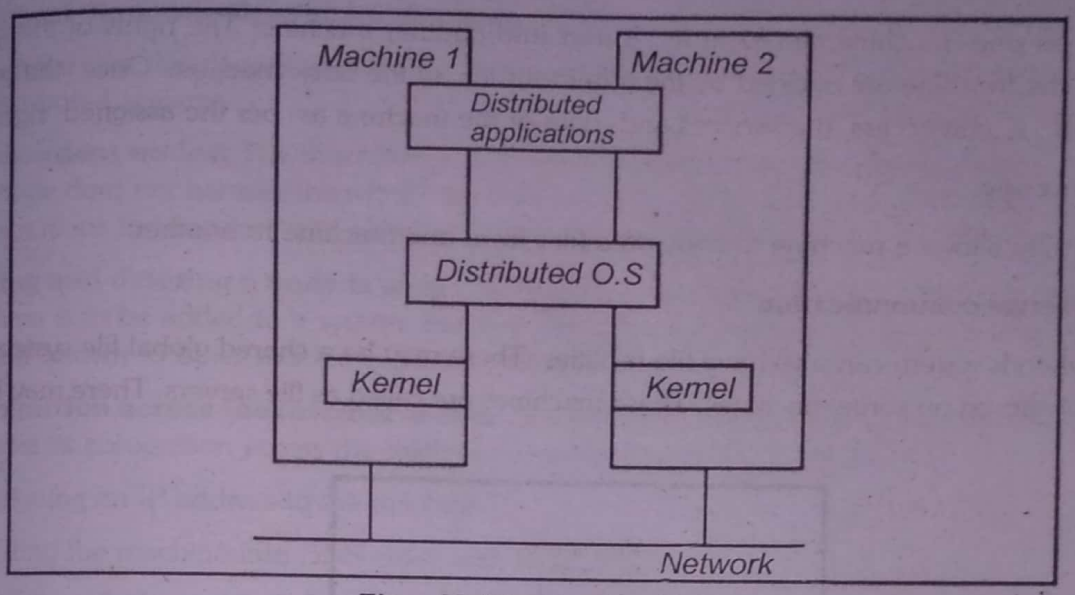


Fig. : Multicomputer O.S

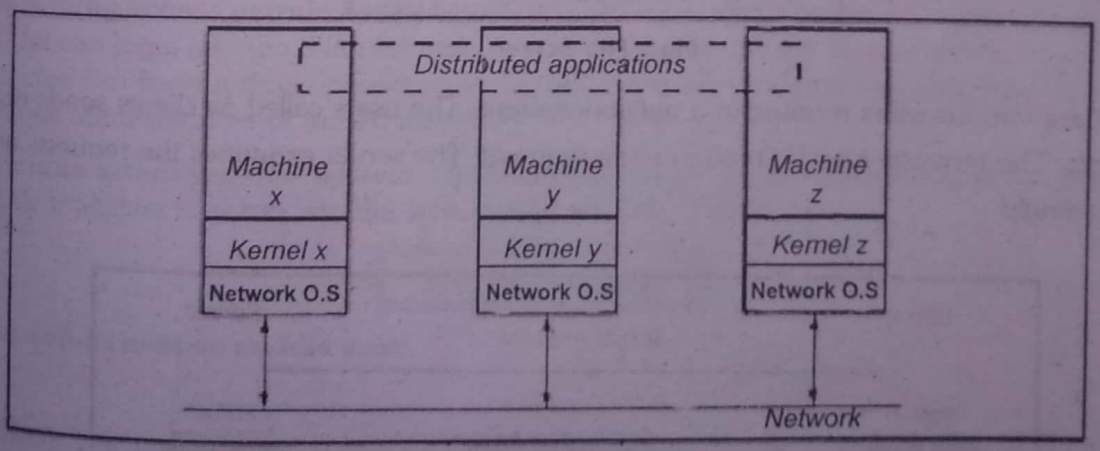
1.6.1 Network Operating System

Q13. Write a short note on network operating systems ?

Ans :

The network operating systems consists of various uniprocessor Systems. Each uniprocessor system has its own operating system.

Since each processor has its own operating system, every processor may have different O.S. and different configurations for the machines. But all these machines are connected to the network.



Like the distributed systems, the network operating system does not assume an homo-geneous setup, i.e. it does not assume the whole setup as a single machine.

But in a network system also there are facilities which allows the services of the remote system .

Some commonly provided services of the network system are

1) Remote login

A user on one machine can login as a user into another machine. The rights of the guest user logging on to the machine are decided by the administrator of the host machine. Once the guest user has logged in , it can access the services and data of the machine as per the assigned rights.

2) Remote copy

This facility allows a machine to copy data files from one machine to another.

3) Client server communication

The network system can also have file facilities. There may be a shared global file system. The file system may be stored on some machines. These machines are called as file servers. There may be several file servers.

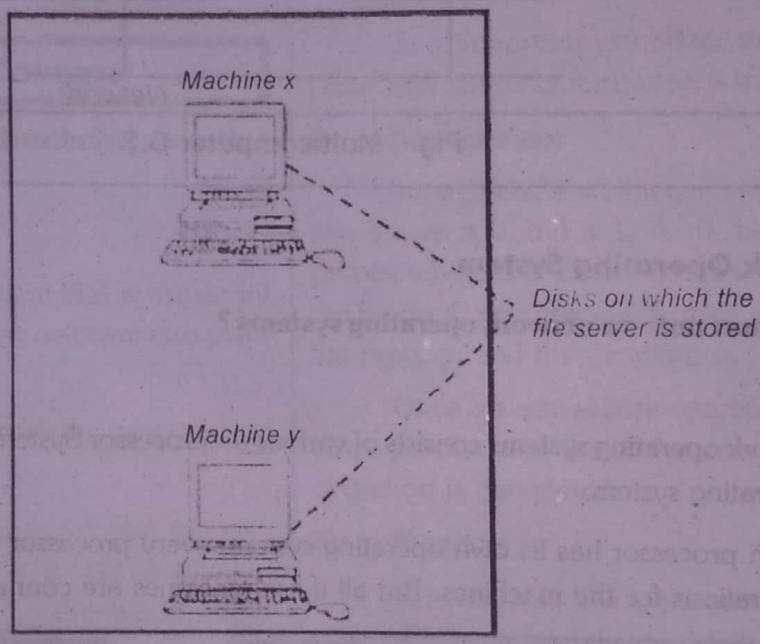


Fig. : File Server

There are various users running in a network system. The users called as clients sends requests to the file servers. The requests may be read or write requests. The server examines the requests and sends a reply accordingly.

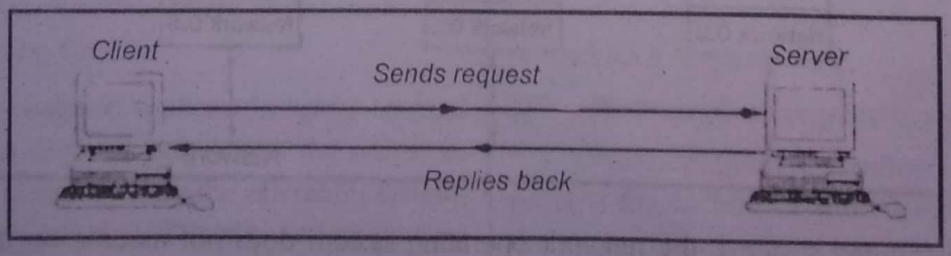


Fig. : Client Server Communication

Advantages and Disadvantages of Network Operating System

What are the advantages and disadvantages of network Operating systems ?

Ans :

Advantages of Network Operating Systems

Except the case that the network systems have only the disadvantages, there are various advantages of distributed systems.

Independent nodes: The machines in the network system are independent nodes. Any failure in any node does not tamper the whole network, only that particular machine is bypassed. Repairing and solutions to hardware and software failure are comparatively easy.

Adding and deleting a node is easy : Since all the machines in the network are independent, if a machine is to be added to a system, the only thing to be done is to connect it to the network and make it known to other machines on the network.

Recognition across the internet is easy : If a new machine is added to the network, it is also easy to make its recognition across the internet primarily this can be done by

i) Assigning an IP address to the machine.

ii) Adding the machine into DNS along with its IP address.

Disadvantages of Network O.S.

Lack of transparency : Unlike distributed systems, the network operating system does not provide the transparency feature i.e. the system does not appear as a single machine to its user. The processes and resources which are spread out may be accessible by certain remote accessing features. Normally, the client server type if communication is preferred in different networks.

This type of communication is normally harder. The data could also be accessed by remote login or remote copy features as explained above.

Management difficulty : In a network system all machines work independently. If there is a problem in machine A, then it has to be solved from machine A only, it can also be solved by remotely logging on to machine A. But this may give rise to the third type of difficulty normally arising in network systems.

Maintaining access permissions : Suppose a user has a right to access 10 machines in the network i.e. if he can login into ten different machines, then he has to have 10 passwords for entry. He, by no choice can have a single password for logging on to all machines, but this password has to be set by all administrators of the 10 machines.

Malicious attacks on the system : In a network, there is always a possibility of remote malicious entry. Intruders may log into the system and lead to a mishap.

1.7 Middleware

Write a short note on middle ware.

Ans :

Middleware is an additional layer of software which is used in the network systems. This layer has a useful feature of hiding the heterogeneity of the systems in the underlying network.

The layer tries to overcome the drawbacks of distributed systems and network systems.

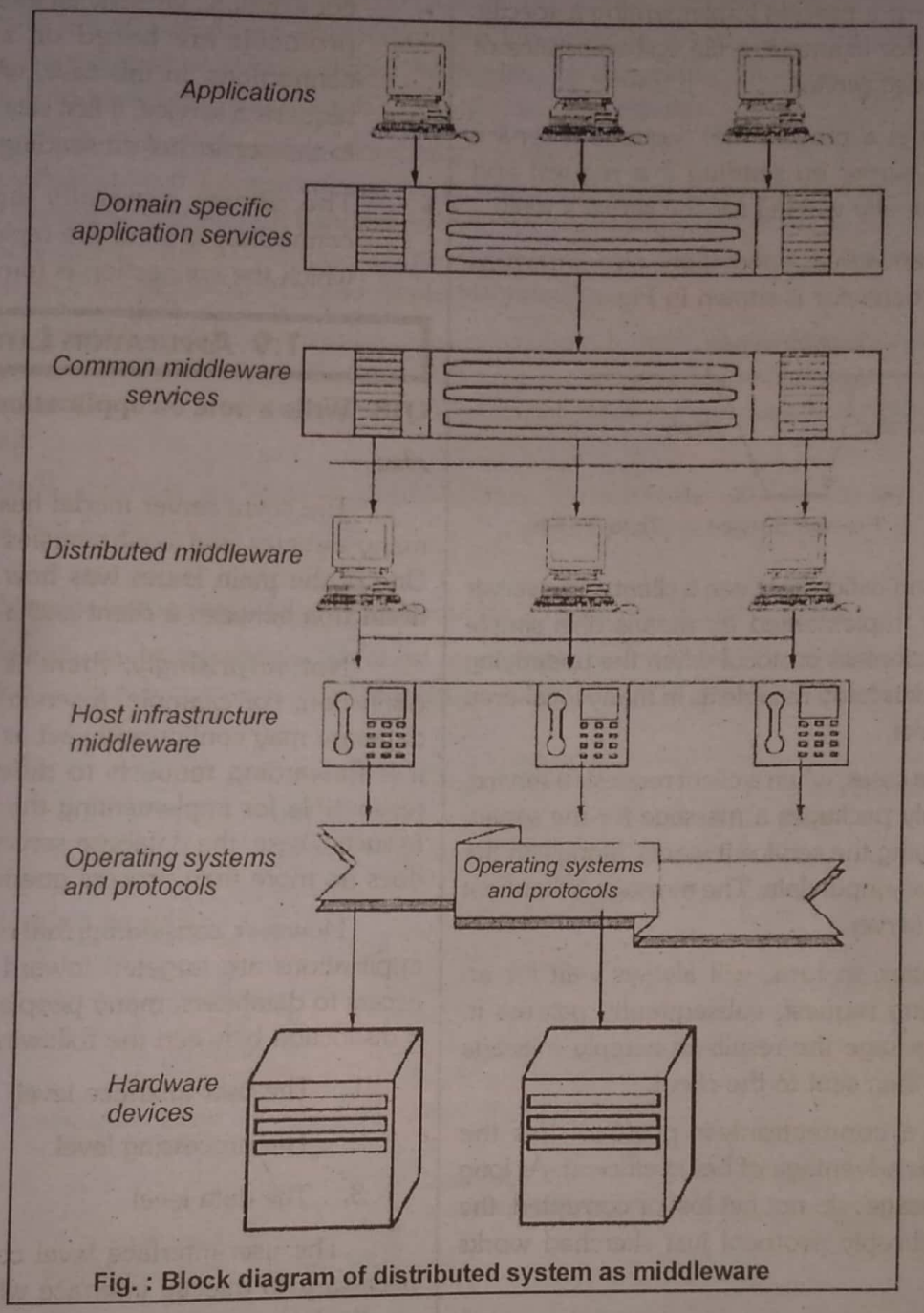


Fig. : Block diagram of distributed system as middleware

1.8 CLIENT-SERVER MODEL : CLIENTS AND SERVERS

Q17. Write a short note on Client-server model.

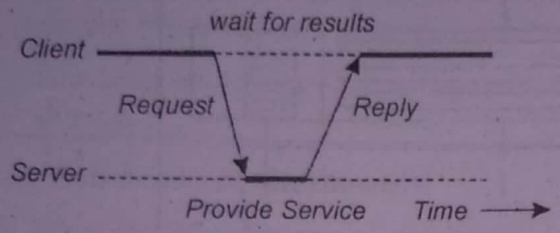
Ans :

The architecture of client server model of a distributed system basically consists of 2 main entities.

- **Clients.** They are sending messages to the servers.
- **Servers.** They are receiving messages from clients and providing them with necessary services.
- In the basic client-server model, processes in a distributed system are divided into two (possibly overlapping) groups.

- A server is a process implementing a specific service, for example, a file system service or a database service.
- 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.

This client-server interaction, also known as request-reply behavior is shown in Fig.



- Communication between a client and a server can be implemented by means of a simple connectionless protocol when the underlying network is fairly reliable as in many local-area networks.
- In these cases, when a client requests a service, it simply packages a message for the server, identifying the service it wants, alongwith the necessary input data. The message is then sent to the server.
- The latter, in turn, will always wait for an incoming request, subsequently process it, and package the results in a reply message that is then sent to the client.
- Using a connectionless protocol has the obvious advantage of being efficient. As long as messages do not get lost or corrupted, the request/reply protocol just sketched works fine.
- Unfortunately, making the protocol resistant to occasional transmission failures is not trivial. The only thing we can do is possibly let the client resend the request when no reply message comes in.
- As an alternative, many client-server systems use a reliable connection oriented protocol. Although this solution is not entirely appropriate in a local-area network due to relatively low performance, it works perfectly fine in wide-area systems in which communication is inherently unreliable.
- For example, virtually all Internet application protocols are based on reliable TCP/IP connections. In this case, whenever a client requests a service, it first sets up a connection to the server before sending the request.
- The server generally uses that same connection to send the reply message, after which the connection is torn down.

1.9 Application Layering

Q18. Write a note on application layering.

Ans :

The client-server model has been subject to many debates and controversies over the years. One of the main issues was how to draw a clear distinction between a client and a server.

Not surprisingly, there is often no clear distinction. For example, a server for a distributed database may continuously act as a client because it is forwarding requests to different file servers responsible for implementing the database tables. In such a case, the database server itself essentially does no more than process queries.

However, considering that many client-server applications are targeted toward supporting user access to databases, many people have advocated a distinction between the following three levels:

1. The user-interface level
2. The processing level
3. The data level

The user-interface level contains all that is necessary to directly interface with the user, such as display management. The processing level typically contains the applications. The data level manages the actual data that is being acted on.

Clients typically implement the user-interface level. This level consists of the programs that allow end users to interact with applications. There is a considerable difference in how sophisticated user-interface programs are.

The simplest user-interface program is nothing more than a character-based screen. Such an interface has been typically used in mainframe environments.

In those cases where the mainframe controls all interaction, including the keyboard and monitor, one can hardly speak of a client-server environment. However, In many cases, 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.

Nowadays, even in mainframe environments, we see more advanced user interfaces. Typically, the client machine offers at least a graphical display in which pop-up or pull-down menus are used, and of which many of the screen controls are handled through a mouse instead of the keyboard. Typical examples of such interfaces include the X-Windows interfaces as used in many UNIX environments, and earlier interfaces developed for MS-DOS PCs and Apple Macintoshes.

Many client-server applications can be constructed from roughly three different pieces: a part that handles interaction with a user, a part that operates on a database or file system, and a middle part that generally contains the core functionality of an application. This middle part is logically placed at the processing level.

- For example, consider an Internet search engine. The user interface of a search engine is very simple: a user types in a string of keywords and is subsequently presented with a list of titles of Web pages.
- The back end is formed by a huge database of Web pages that have been pre fetched and indexed.
- It subsequently ranks the results into a list, and transforms that list into a series of HTML pages. Within the client-server model, this information retrieval part is typically placed at the processing level.

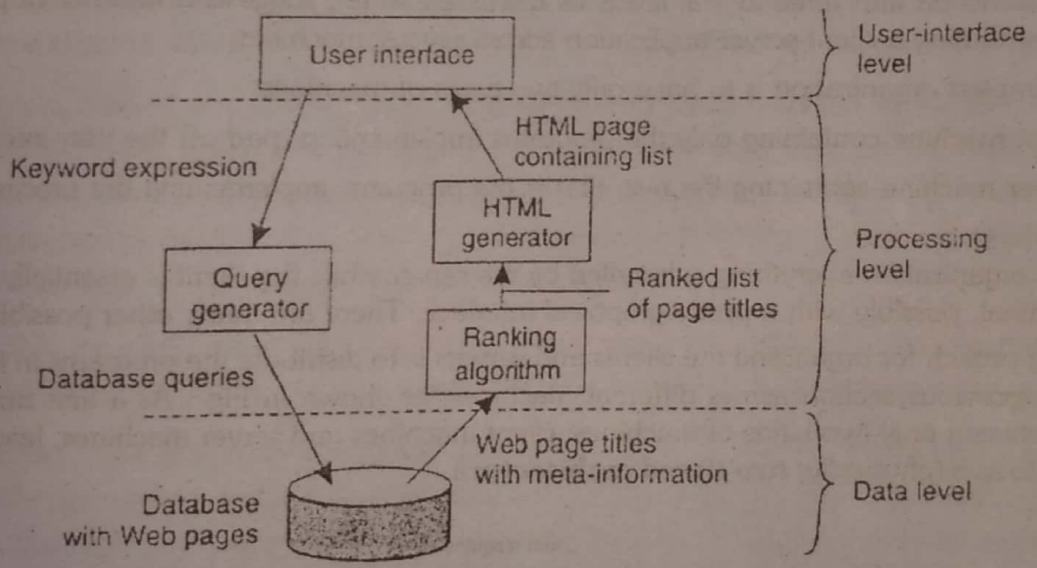


Fig. : The Simplified organisation of an Internet search engine into three differen layers

As a second example, consider a decision support system for a stock brokerage. Analogous to a search engine, such a system can be divided into a front end implementing the user interface, a back end for accessing a database with the financial data, and the analysis programs between these two. Analysis of financial data may require sophisticated methods and techniques from statistics and artificial intelligence. In some cases, the core of a financial decision support system may even need to be executed on high-performance computers in order to achieve the throughput and responsiveness that is expected from its users. Throughput and responsiveness that is expected from its users.

As a last example, consider a typical desktop package, consisting of a word processor, a spreadsheet application, communication facilities, and so on.

Such "office" suites are generally integrated through a common user interface that supports compound documents, and operates on files from the user's home directory.(In an office environment, this home directory is often placed on a remote fileserver.) In this example, the processing level consists of a relatively large collection of programs, each having rather simple processing capabilities.

The data level in the client-server model contains the programs that maintain the actual data on which the applications operate. An important property of this level is that data are often persistent, that is, even if no application is running, data will be stored somewhere for next use. In its simplest form, the data level consists of a file system, but it is more common to use a full-fledged database.

In the client-server model, the data level is typically implemented at the server side.

Besides merely storing data, the data level is generally also responsible for keeping data consistent across different applications. When databases are being used, maintaining consistency means that metadata such as table descriptions, entry constraints and application-specific metadata are also stored at this level. For example, in the case of a bank, we may want to generate a notification when a customer's credit card debt reaches a certain value. This type of information can be maintained through a database trigger that activates a handler for that trigger at the appropriate moment.

1.10 Multi Tiered Architectures

Q19. Explain about multi tiered architectures.

Ans :

The distinction into three logical levels as discussed so far, suggests a number of possibilities for physically distributing a client-server application across several machines.

The simplest organization is to have only 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.

In this organization everything is handled by the server while the client is essentially no more than a dumb terminal, possibly with a pretty graphical interface. There are many other possibilities.

One approach for organizing the clients and servers is to distribute the programs in the application layers of the previous section across different machines, as shown in Fig. As a first step, we make a distinction between only two kinds of machines: client machines and server machines, leading to what is also referred to as a (physically) **two tiered architecture**.

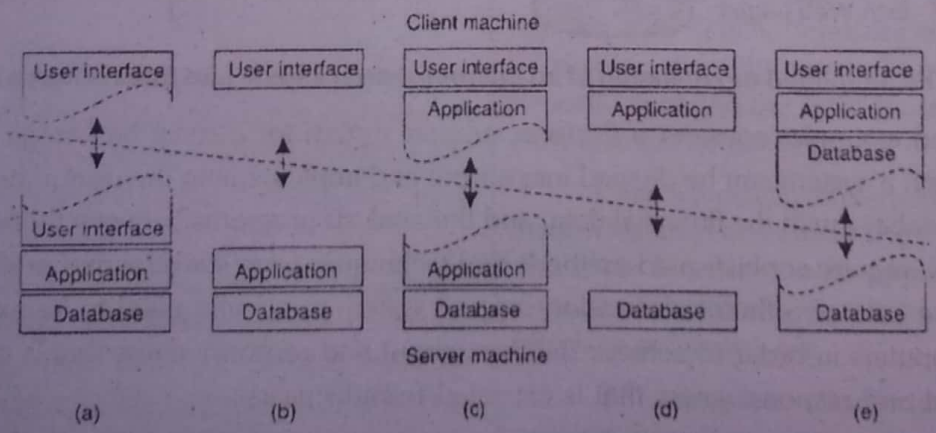


Fig. : Alternative Client-server organsiations

One possible organization is to have only the terminal-dependent part of the user interface on the client machine, as shown in Fig. 2-5(a), and give the applications remote control over the presentation of their data.

An alternative is to place the entire user-interface software on the client side, as shown in Fig. 2-5(b). In such cases, we essentially 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. In this model, the front end (the client software)does no processing other than necessary for presenting the application's interface.

Continuing along this line of reasoning, we may also move part of the application to the front end, as shown in Fig. 2-5(c). An example where this makes sense is where the application makes use of a form that needs to be filled in entirely before it can be processed. The front end can then check the correctness and consistency of the form, and where necessary interact with the user. Another example of the organization of Fig. 2:.5(c),is that of a word processor in which the basic editing functions execute on the client side where they operate on locally cached, or in-memory data. but where the advanced support tools such as checking the spelling and grammar execute on the server side.

In many client-server environments, the organizations shown in Fig. 2-5(d)and Fig. 2-5(e) are particularly popular. These organizations are used where the client machine is a PC or workstation, connected through a network to a distributed file system or database. Essentially, most of the application is running on the client machine, but all operations on files or database entries go to the server.

For example, many banking applications run on an end-user's machine where the user prepares transactions and such. Once finished, the application contacts the data base on the bank's server and uploads the transactions for further processing. Fig. 2-5(e) represents the situation where the client's local disk contains part of the data. For example, when browsing the Web, a client can gradually build a huge cache on local disk of most recent inspected Web pages.

We note that for a few years there has been a strong trend to move away from the configura-tions shown in Fig. 2-5(d) and Fig. 2-5(e) in those case that client software is placed at end-user machines.

In these cases, most of the processing and data storage is handled at the server side. The reason for this is simple: 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 what are called fat clients is not optimal. Instead the thin clients as represented by the organizations shown in Fig. 2-5(a)-(c) are much easier, perhaps at the cost of less sophisticated user interfaces and client-perceived performance.

Note that this trend does not imply that we no longer need distributed systems.

On the contrary, what we are seeing is that server-side solutions are becoming increasingly more distributed as a single server is being replaced by multiple servers running on different machines. In particular, when distinguishing only client and server machines as we have done so far, we miss the point that a server may sometimes need to act as a client, as shown in Fig. 2-6, leading to a (physically)**three-tiered architecture.**

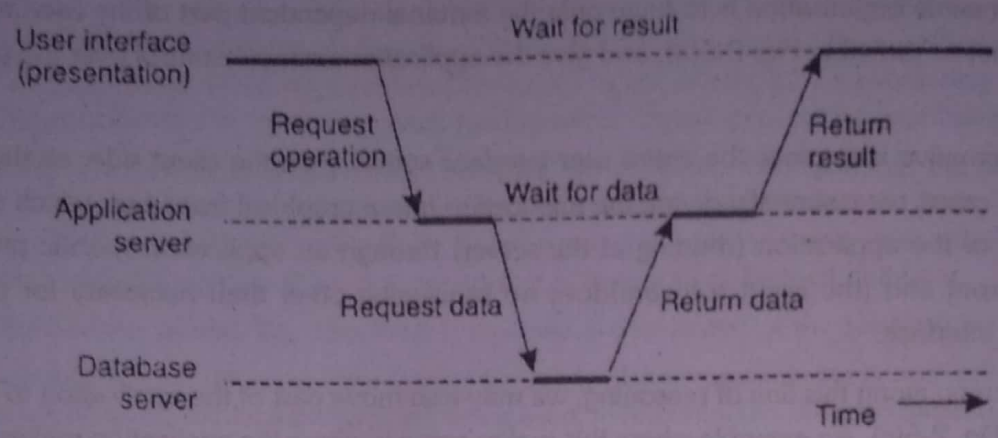


Fig. : An example of a server acting as client

In this architecture, 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. A typical example of where a three-tiered architecture is used is in transaction processing. As we discussed in Chap. 1, a separate process, called the transaction processing monitor, coordinates all transactions across possibly different data servers.

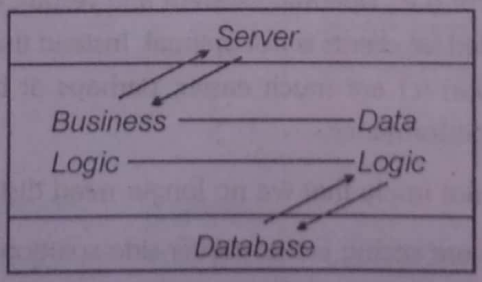
Another, but very different example where we often see a three-tiered architecture is in the organization of Web sites. In this case, a Web server acts as an entry point to a site, passing requests to an application server where the actual processing takes place. This application server, in turn, interacts with a database server. For example, an application server may be responsible for running the code to inspect the available inventory of some goods as offered by an electronic bookstore. To do so, it may need to interact with a database containing the raw inventory data.

Client Server Architectures

- 1 tier architecture
- 2 tier architecture
- 3 tier architecture

1 tier architecture

In this type of client server environment the user interface, business logic & data logic are present in same system. This kind of client server service is cheapest but it is difficult to handle because of data inconsistency that allows repetition of work.

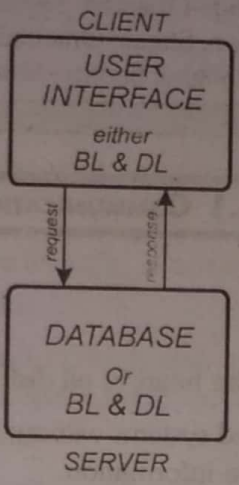


2 tier architecture

In this type of client server environment user interface is stored at client machine and database is stored on server.

Database logic & business logic are stored at either client or server but it must be unchanged. If Business Logic & Data Logic is stored at client side, it is called fat client thin server architecture. If Business Logic & Data Logic are stored on server, it is called thin client fat server architecture.

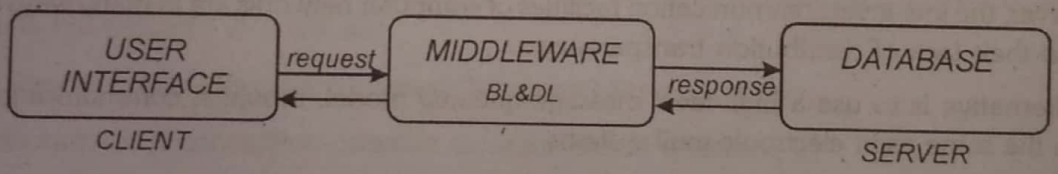
This kind of architecture are affordable and comparatively better.



3 tier architecture

In this kind of client server environment an additional middleware is used that means client request goes to server through that middle layer and the response of server is firstly accepted by middleware then to client. This architecture overcomes all the drawbacks of 2-tier architecture and gives best performance. It is costly and easy to handle. The middleware stores all the business logic and data access logic. If there are multiple Business Logic & Data Logic, it is called n-tier architecture.

The purpose of middleware is to database staging, queuing, application execution, scheduling etc. middleware can be file server, message server, application server, transaction processing monitor etc. It improves flexibility and gives best performance.



UNIT II

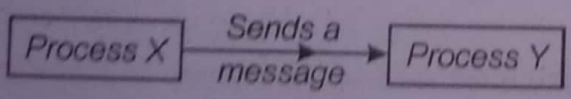
Communication : Layered Protocols, Lower-Level Protocols, Transport Protocols, Higher-Level Protocols, Remote Procedure Call: Basic RPC Operation, Parameter Passing. Extended RPC Models, Remote Object Invocation: Distributed Objects, Binding a Client to an Object; Static verses Dynamic Remote Method Invocations, Parameter Passing, Message Oriented Communication: Persistence and synchronicity in Communication, Message-Oriented Transient Communication, Message-Oriented' Persistent Communication, Stream Oriented Communication: Support for Continuous Media, Streams and Quality of Service, Stream Synchronization.

2.1 COMMUNICATION

Q1. What is communication ?

Ans :

- Inter process communication is at the heart of all distributed systems.
- It makes no sense to study distributed systems without carefully examining the ways that processes on different machines can exchange information.
- 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, as available for non distributed platforms.
- Modem distributed systems often consist of thousands or even millions of processes scattered across a network with unreliable communication such as the Internet.
- Unless the primitive communication facilities of computer networks are replaced by something else, development of large-scale distributed applications is extremely difficult.
- In many distributed applications, communication does not follow the rather strict pattern of client-server interaction. In those cases, it turns out that thinking in terms of messages is more appropriate.
- However, the low-level communication facilities of computer networks are in many ways not suitable due to their lack of distribution transparency.
- An alternative is to use a high-level message-queuing model, in which communication precedes much the same as in electronic mail systems.
- Message-oriented middleware (MOM) is a subject important enough to warrant a section of its own.
- In a distributed system, the processes or tasks needs to communicate with each other because there is no shared memory, so there is a need for message passing mechanisms. For example, process X sends a message to process Y. Process Y receives it.




```
Process X
    send (message, Y)
    Receive (Reply, Y)
Process Y
    Receive (message, X)
    Send (Reply, X)
```

2.2 Layered Protocols

Q2. Write a short note on OSI Model (Layered protocols).

Ans :

- 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.
- Although this basic idea sounds simple enough, in order to prevent chaos, A and B have to agree on the meaning of the bits being sent.

If A sends a brilliant new novel written in French and encoded in IBM's EBCDIC character code, and B expects the inventory of a supermarket written in English and encoded in ASCII, communication will be less than optimal.

To make it easier to deal with the numerous levels and issues involved in communication, the International Standards Organization (ISO) developed a reference model called **Open Systems Interconnection Reference Model** usually abbreviated as OSI model.

- The 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.
- These rules are formalized in what are called **protocols**.
- To allow a group of computers to communicate over a network, they must all agree on the protocols to be used.
- A distinction is made between two general types of protocols. With connection oriented protocols, before exchanging data the sender and receiver first explicitly establish a connection, and possibly negotiate the protocol they will use.
- When they are done, they must release (terminate) the connection.

The telephone is a connection-oriented communication system.

- With connectionless protocols, no setup in advance is needed. The sender just transmits the first message when it is ready. Dropping a letter in a mailbox is an example of connectionless communication.
- With computers, both connection-oriented and connectionless communications are common.
- In the OSI model, communication is divided up into **seven levels or layers**, as shown in Fig. 4-1.
- Each layer deals with one specific aspect of the communication. In this way, the problem can be divided up into manageable pieces, each of which can be solved independent of the others.
- 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.

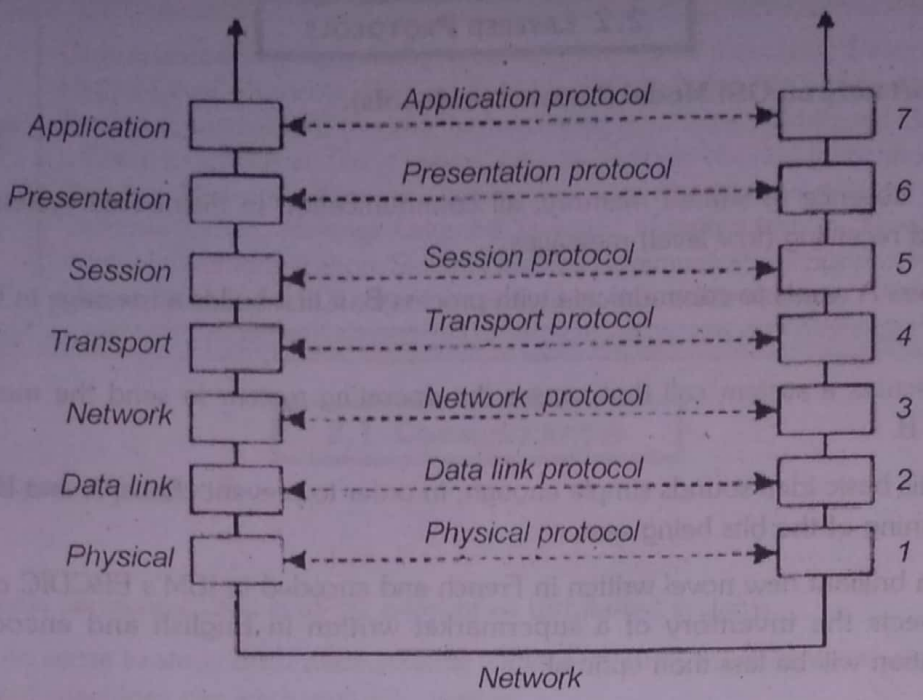


Fig. : Layers, interfaces, and protocols in the OSI model

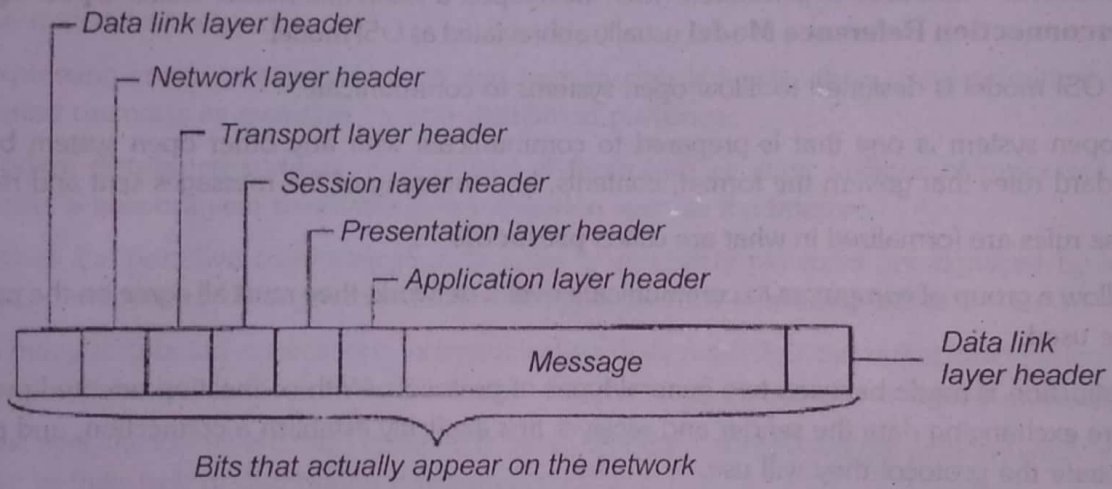


Fig. : A typical message as it appears on the network

Functions of OSI Layers

- When **process *A*** on machine 1 wants to communicate with **process *B*** on machine 2, it builds a message and passes the message to the application layer on its machine.
- The **application layer** software then adds a header to the front of the message and passes the resulting message across the layer 6/7 interface to the presentation layer.
- The **presentation layer** in turn adds its own header and passes the result down to the session layer, and so on.
- Some layers add not only a header to the front, but also a trailer to the end.
- When it hits the bottom, the **physical layer** actually transmits the message by putting it onto the physical transmission medium.

- When the message arrives at machine 2, it is passed upward, with each layer stripping off and examining its own header.
- Finally, the message arrives at the receiver; process B, which may reply to it using the reverse path. The information in the layer n header is used for the layer n protocol.

2.3 Lower Level Protocols

Q3. Explain about lower level protocols of OSI model.

Ans :

a) The physical layer

- It is the Lower layer of OSI architecture.
- It handles the electrical and mechanical interface.
- It also handles the procedural requirements of the interconnection medium.
- This layer is also responsible for bit synchronization.
- It includes mechanical, electrical, functional and procedural specifications.

Typical protocols at the physical layer include the RS-232, the RS-449 family, CCITT

X.25 and X.21 facility interfaces, other CCITT (V) and (X) series recommendations and the physical aspects of the IEEE 802.X media access protocols for Local Area Networks.

b) The Data Link layer

- This layer is responsible for reliable interchange of data across a point-to-point link established by physical layer.
- It attempts to add reliability, flow and error control.
- It provides communication management.

Data link layer protocols include character oriented Binary Synchronous Communication (BSC), ANSI X3.28, Advanced Data Communication Control Procedure (ADCCP)...etc.

c) The Network layer

- This layer is responsible for providing communication between two hosts across communication network.
- It provides services like routing, switching, sequencing of data.
- It also provides flow control and error recovery.
- It provides the interface that higher layers needs.

The best known network layer protocol for packet-switched networks is the CCITT X.25 Packet Layer protocol.

2.4 Transport Protocols

Q5. Write a short note on transport protocols.

Ans :

Transport layer

- This is the highest layer directly associated with the movement of data through the network.
- It provides a universal transparent mechanism for use by the higher layers that represent the users of the communications service.
- The transport layer is expected to optimize the use of available **resources** while meeting user requirements.
- Responsible for the **end-to-end integrity** of the edit exchange.
- It must bridge the gap between services provided by the underlying network and those required by the higher layers.
- This layer is responsible for Quality of Service **(QOS).**
- Simple transport layer can be used when the network provides a high quality, reliable service.
- A complex transport protocol is used when the underlying service does not, or is assumes to be unable to, provide the required level of service.

International **Standard 8073** is a transport protocol. This standard defines five (5) classes of protocols, ranging from a simple class (0) to a complex class (4).

Another transport protocol example is the Transmission Control Protocol (TCP) developed by the DoD and now finding wide application in commercial environments. We will elaborate TCP in detail in the section.

Transmission Control Protocol (TCP)

Most applications on the Internet make use of TCP. This is because the TCP ensures safe delivery of data across an unreliable IP layer that is present below it. In this section we explore the fundamental concepts behind TCP and how it is used to transport data between two endpoints.

A communication protocol is description of the rules computers must follow to communicate with each other. The internet communication protocol defines the rules for computer communication over the Internet.

Since TCP and IP were originally envisioned functionally as a single protocol, the protocol suite, which actually refers to a large collection of protocols and applications, is usually referred to simply as TCP/IP.

Client Server TCP

Since TCP is a communication protocol, it does the basic job of providing a framework for communication between a client and a server.

On the internet, client server applications are build using an TCP.

The TCP client server process takes place in the following way.

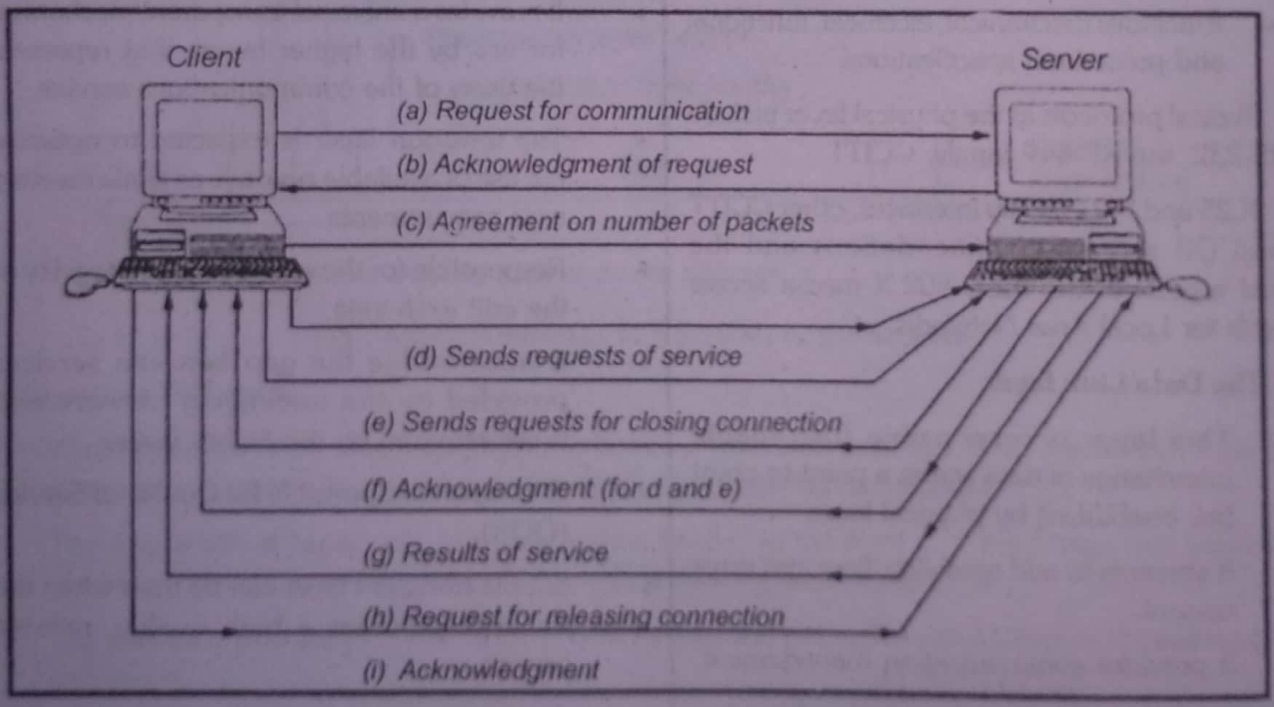
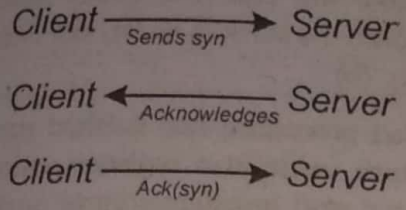


Fig. : TCP client server communication

- A three way handshake protocol is used by the client to establish the connection between client and server.



This protocol makes the client and server to agree on the sequence numbering of packets that will be sent across the connection once it is established.

- After the connection is established, the client sends its request.
- It immediately sends the packet to inform the server to close the connection.
- On receiving the request from the client, the server acknowledges that it received the request and the signal to close the connection.
- The server performs the requested task and sends the result to the client.
- The server then sends a request to release the connection.
- Finally, the client acknowledges this request indicating that this is the end of the communication with the server.

2.5 Higher Level Protocols

Q6. Write about Higher level protocols.

Ans :

The protocols in session layer, presentation layer and application layer are higher level protocols.

1. Session layer

- A session **binds two application processes** into a co-operative relationship for a certain time.
- This layer provides administrative service that handles the **establishment** (binding) and **release** (unbinding) of a connection between two presentation entities.
- It provides half/full-duplex, **synchronization** of long transaction, **exception handling**.
- Sessions are established when an application process requests access to another application process.

2. Presentation Layer

- The services of this layer allow an application to properly interpret the information being transferred.
- This layer involves the translation, transformation, formatting and syntax of the information.

The syntactical representation of data has been defined in DIS 8824 and 8825.

CCITT has described the presentation protocol for message handling systems in X.409 and for Telex in X.61.

3. Application layer

- This layer provides management functions to **support distributed applications** utilizing the OSI environment.
- It is the window through which the applications gain access to the services provided by the communications architecture.
- These include identification of the co-operating processes, authentication of the communicant, authority verification.
- **This layer also provides agreement on encryption** mechanisms, determination of resource availability and agreement on **syntax**, e.g. character set, data structure.

2.6 Remote Procedure Call

Q7. What is RPC?_Explain basic principle beyond RPC.

Ans :

Many distributed systems have been based on explicit message exchange between processes. However, the procedures send and receive do not conceal communication at all, which is important to achieve access transparency in distributed systems.

When a process on machine A calls' a procedure on machine B, the calling process on A is suspended, and execution of the called procedure takes place on B. Information can be transported from the caller to the callee in the parameters and can come back in the procedure result. This method is known as **Remote Procedure Call**, or often just **RPC**.

While the basic idea sounds simple and elegant, subtle problems exist, because the calling and called procedures run on different machines, they execute in different address spaces, which causes complications. Parameters and results also have to be passed, which can be complicated, especially if the machines are not identical. Finally, either or both machines can crash and each of the possible failures causes different problems. Still, most of these can be dealt with, and RPC is a widely-used technique that underlies many distributed systems.

2.6.1 Basic RPC Operation

Q8. Explain about basic RPC operation.

Ans :

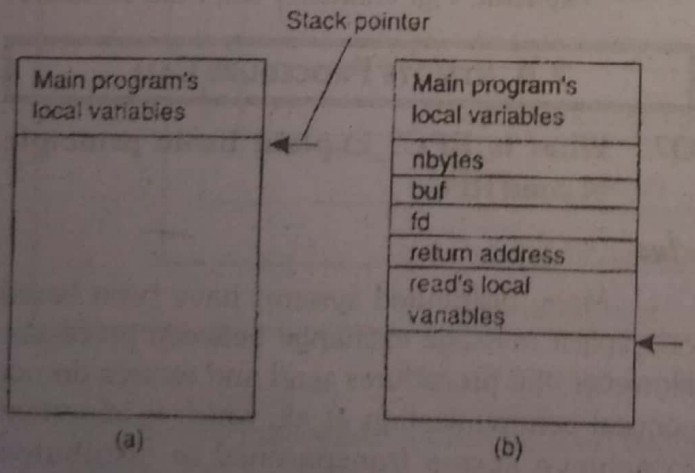
Conventional Procedure Call

To understand how RPC works, it is important first to fully understand how a conventional (i.e., single machine) procedure call works.

Consider a call in C like **count = read(fd, buf, nbytes);**

where fd is an integer indicating a file, buf is an array of characters into which data are read, and nbytes is another integer telling how many bytes to read.

If the call is made from the main program, the stack will be as shown in Fig (a) before the call.



To make the call, the caller pushes the parameters onto the stack in order, last one first, as shown in Fig. (b).

After the read procedure has finished running, it puts the return value in a register, removes the return address, and transfers control back to the caller. The caller then removes the parameters from the stack, returning the stack to the original state it had before the call.

Several things are worth noting. For one, in C, parameters can be call-by value or call-by-reference. A value parameter, such as fd or nbytes, is simply copied to the stack as shown in Fig. 4-5(b). To the called procedure, a value parameter is just an initialized local variable. The called procedure may modify it, but such changes do not affect the original value at the calling side.

A reference parameter in C is a pointer to a variable (i.e., the address of the variable), rather than the value of the variable. In the call to read, the second parameter is a reference parameter because arrays are always passed by reference in C.

What is actually pushed onto the stack is the address of the character array.

If the called procedure uses this parameter to store something into the character array, it does modify the array in the calling procedure. The difference between call-by-value and call-by-reference is quite important for RPC, as we shall see.

One other parameter passing mechanism also exists, although it is not used in C.

It is called **call-by-copy/restore**. It consists of having the variable copied to the stack by the caller, as in call-by-value, and then copied back after the call, overwriting the caller's original value. Under most conditions, this achieves exactly the same effect as

call-by-reference, but in some situations. such as the same parameter being present multiple times in the parameter list. The semantics are different. The call-by-copy/restore mechanism is not used in many languages.

The idea behind RPC is to make a remote procedure call look as much as possible like a local one. In other words, we want RPC to be transparent-the calling procedure should not be aware that the called procedure is executing on a different machine or vice versa. Suppose that a program needs to read some data from a file. The programmer puts a call to read in the code to get the data. In a traditional (single-processor) system, the read routine is extracted from the library by the linker and inserted into the object program. It is a short procedure, which is generally implemented by calling an equivalent read system call. In other words, the read procedure is a kind of interface between the user code and the local operating system.

Client and Server Stubs

The idea behind RPC is to make a remote procedure call look as much as possible like a local one. In other words, we want RPC to be transparent-the calling procedure should not be aware that the called procedure is executing on a different machine or vice versa.

Suppose that a program needs to read some data from a file. The programmer puts a call to read in the code to get the data. In a traditional (single-processor) system, the read routine is extracted from the library by the linker and inserted into the object program.

When the message arrives at the server, the server's operating system passes it up to a server stub. A server stub is the server-side equivalent of a client stub: it is a piece of code that transforms requests coming in over the network into local procedure calls. Typically the server stub will have called receive and be blocked waiting for incoming messages. The server stub unpacks the parameters from the message and then calls the server procedure in the usual way.

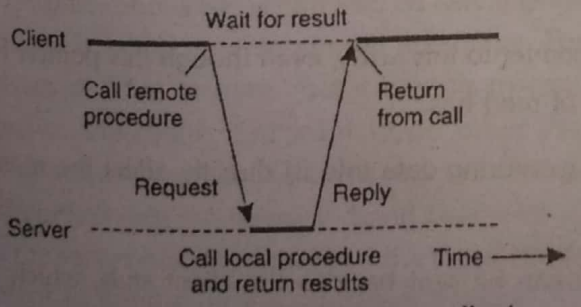


Fig.: Principle of RPC between a client and server program

To summarize, a remote procedure call occurs in the following steps:

1. The client procedure calls the client stub in the normal way.
2. The client stub builds a message and calls the local operating system.
3. The client's as sends the message to the remote as.
4. The remote as gives the message to the server stub.
5. The server stub unpacks the parameters and calls the server.
6. The server does the work and returns the result to the stub.
7. The server stub packs it in a message and calls its local as.
8. The server's as sends the message to the client's as.
9. The client's as gives the message to the client stub.
10. The stub unpacks the result and returns to the client.

2.6.2 Parameter Passing

Q9. Write a short note on parameter passing in RPC.

Ans :

The function of the client stub is to take its parameters, pack them into a message, and send them to the server stub.

Passing Value Parameters

Packing parameters into a message is called parameter marshaling. As a very simple example, consider a remote procedure, add(i, j), that takes two integer parameters i and j and returns their arithmetic sum as a result.

The call to add, is shown in the left-hand portion (in the client process) in Fig. 4-7. The client stub takes its two parameters and puts them in a

message as indicated, It also puts the name or number of the procedure to be called in the message because the server might support several different calls, and it has to be told which one is required.

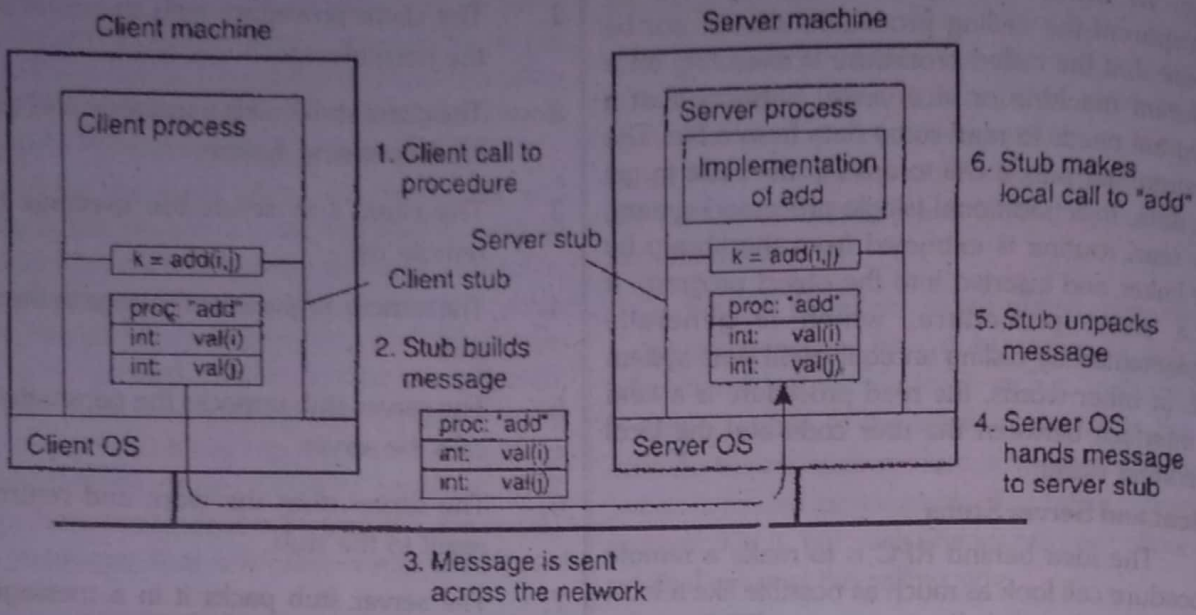


Fig. : The steps involved in a doing a remote computation through RPC

When the message arrives at the server, the stub examines the message to see which procedure is needed and then makes the appropriate call. If the server also supports other remote procedures, the server stub might have a switch statement in it to select the procedure to be called, depending on the first field of the message.

The actual call from the stub to the server looks like the original client call, except that the parameters are variables initialized from the incoming message.

When the server has finished, the server stub gains control again. It takes the result sent back by the server and packs it into a message. This message is sent back to the client stub. This unpacks it to extract the result and returns the value to the waiting client procedure.

Passing Reference Parameters

In the read example, the client stub knows that the second parameter points to an array of characters. Suppose, for the moment, that it also knows how big the array is. One strategy then becomes apparent: copy the array into the message and send it to the server.

The server stub can then call the server with a pointer to this array, even though this pointer has a different numerical value than the second parameter of read has.

Changes the server makes using the pointer (e.g., storing data into it) directly affect the message buffer inside the server stub.

When the server finishes, the original message can be sent back to the client stub, which then copies it back to the client. In effect, call-by-reference has been replaced by copy/restore.

2.6.3 Extended RPC Models

Q10. Write a note on extended RPC models.

Ans :

Asynchronous RPC

As in conventional procedure calls, when a client calls a remote procedure, the client will block until a reply is returned. This strict request-reply behavior is unnecessary when there is no result to return, and only leads to blocking the client while it could have proceeded and have done useful work just after requesting the remote procedure to be called. Examples of where there is often no need to wait for a reply include: transferring money from one account to another, adding entries into a database, starting remote services, batch processing, and so on.

To support such situations, RPC systems may provide facilities for what are called asynchronous RPCs, by which a client immediately continues after issuing the RPC request. With asynchronous RPCs, the server immediately sends a reply back to the client the moment the RPC request is received, after which it calls the requested procedure. The reply acts as an acknowledgment to the client that the server is going to process the RPC. The client will continue without further blocking as soon as it has received the server's acknowledgment. Fig.(b) shows how client and server interact in the case of asynchronous RPCs. For comparison, Fig.(a) shows the normal request-reply behavior.

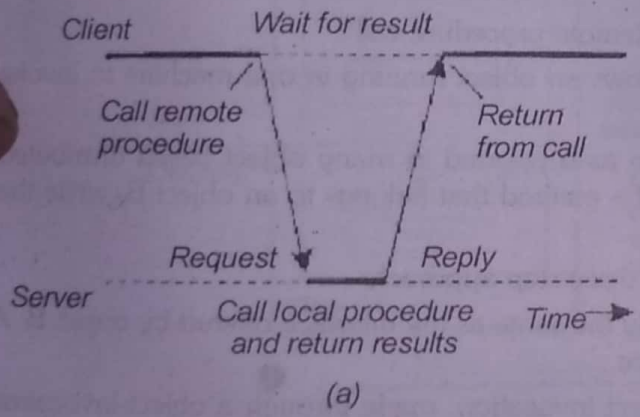


Fig.(a) : The interaction between client and werver in a traditional RPC

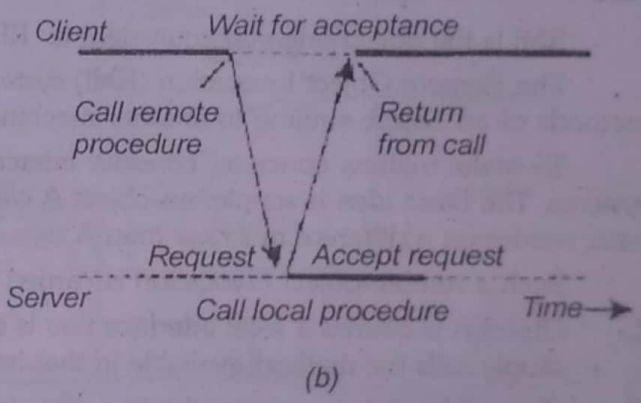


Fig.(b) : The interaction using asynchronous Rp'c

Synchronous RPC

Asynchronous RPCs can also be useful when a reply will be returned but the client is not prepared to wait for it and do nothing in the meantime. For example, a client may want to prefetch the network addresses of a set of hosts that it expects to contact soon. While a naming service is collecting those addresses, the client may want to do other things. In such cases, it makes sense to organize the communication between the client and server through two asynchronous RPCs, as shown in Fig. 4-11. The client first calls the server to hand over a list of host names that should be looked up, and continues when the server has acknowledged the receipt of that list. The second call is done by the server, who calls the client to hand over the addresses it found. Combining two asynchronous RPCs is sometimes also referred to as a deferred **synchronous RPC**.

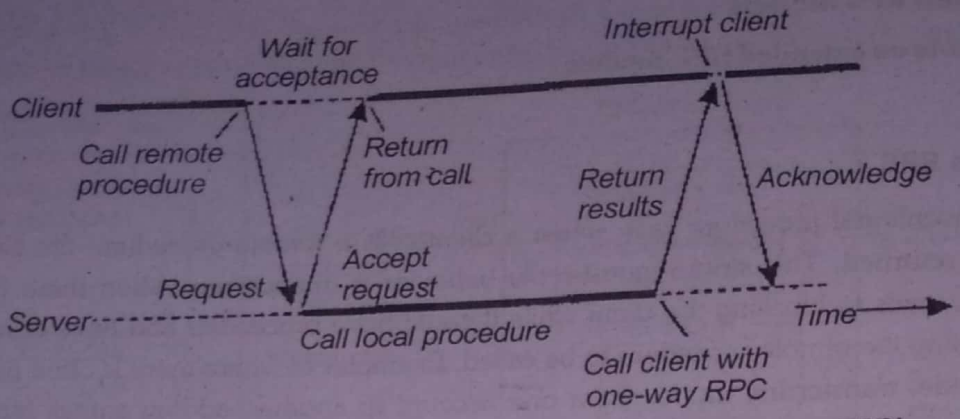


Fig. : A client and server interacting through two asynchronous RPC's

It should be noted that variants of asynchronous RPCs exist in which the client continues executing immediately after sending the request to the server. In other words, the client does not wait for an acknowledgment of the server's acceptance of the request. We refer to such RPCs as one-way RPCs. The problem with this approach is that when reliability is not guaranteed, the client cannot know for sure whether or not its request will be processed.

2.7 Remote Object Invocation/ Remote Method Invocation

Q11. Write about Remote Object Invocation / Remote Method Invocation.

Ans :

RMI is the object oriented equivalent to RPC (Remote procedure call).

The Remote Object Invocation (RMI) system allows an object running in one machine to invoke methods of an object running in another machine.

To make matters concrete, consider interception as supported in many object based distributed systems. The basic idea is simple: an object A can call a method that belongs to an object B, while the latter resides on a different machine than A.

Such a remote-object invocation is carried as a three-step approach:

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 through a 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 OS.

2.7.1 Distributed Objects

Q12. Write about distributed objects.

Ans :

- Distributed objects form an important paradigm because it is relatively easy to hide distribution aspects behind an object's interface.
- Furthermore, because an object can be virtually anything, it is also a powerful paradigm for building systems.
- These principles of distributed systems are applied to a number of well-known object-based systems such as CORBA, Java-based systems, and Globe.

- Object orientation began to be used for developing distributed systems in the 1980s. Again, the notion of an independent object hosted by a remote server while attaining a high degree of distribution transparency formed a solid basis for developing a new generation of distributed systems.
- 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. It is important to understand that there is no "legal" way a process can access or manipulate the state of an object other than by invoking methods made available to it via an object's interface.
- An object may implement multiple interfaces. Likewise, given an interface definition, there may be several objects that offer an implementation for it.
- This separation between interfaces and the objects implementing these interfaces is crucial for distributed systems.
- A strict separation allows us to place an interface at one machine, while the object itself resides on another machine. This organization, which is shown in Fig. 10-1, is commonly referred to as a distributed object.
- 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.
- A proxy is analogous to a client stub in RPC systems. The only thing it does is marshal method invocations into messages and unmarshal reply messages to return the result of the method invocation to the client.

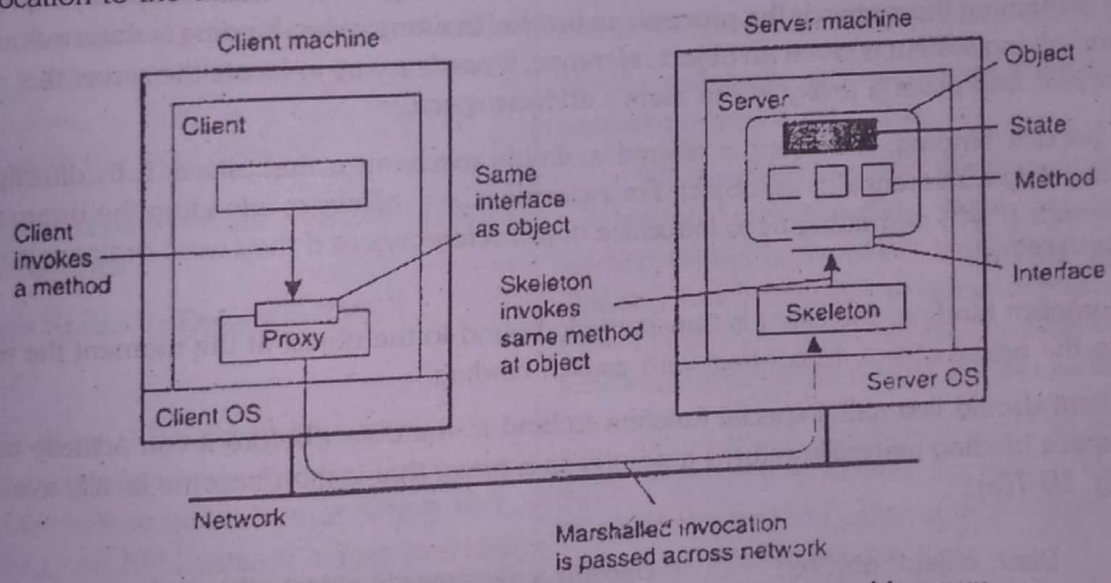


Fig.: Common organisationof a remote object with client-side proxy

- The actual object resides at a server machine, where it offers the same interface as it does on the client machine.
- Incoming invocation requests are first passed to a server stub, which unmarshals them to make method invocations at the object's interface at the server.
- The server stub is also responsible for marshaling replies and forwarding reply messages to the client side proxy.
- 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.
- In practice, it often contains incomplete code in the form of a language-specific class that needs to be further specialized by the developer.

- A characteristic, but somewhat counter intuitive feature of most distributed objects is that their state is *not* distributed: it resides at a single machine.
- Only the interfaces implemented by the object are made available on other machines. Such objects are also referred to as remote objects.
- In a general distributed object, the state itself may be physically distributed across multiple machines, but this distribution is also hidden from clients behind the object's interfaces.

2.7.2 Binding a Client to an Object

Q13. Explain about binding a client to an object.

Ans :

An interesting difference between traditional RPC systems and systems supporting distributed objects is that the latter generally provides system wide object references. Such object references can be freely passed between processes on different machines, for example as parameters to method invocations. By hiding the actual implementation of an object reference, that is, making it opaque, and perhaps even using it as the only way to reference _objects, distribution transparency is enhanced compared to traditional RPCs.

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. When the underlying system is given an object reference, it needs a way to locate the server that manages the actual object, and place a proxy in the client's address space.

With implicit binding, the client is offered a simple mechanism that allows it to directly invoke methods using only a reference to an object. For example, C++ allows overloading the unary member selection operator ("=") permitting us to introduce object references as if they were ordinary pointers as shown in Fig. 10-7(a).

With implicit binding, the client is transparently bound to the object at the moment the reference is resolved to the actual object. In contrast with explicit binding.

The client should first call a special function to bind to the object before it can actually invoke its methods. Explicit binding generally returns a pointer to a proxy that is then become locally available, as shown in Fig. 10-7(b).

```
Distr_object* obj_ref;          // Declare a systemwide object reference
obj_ref = ...;                  // Initialize the reference to a distrib. obj.
obj_ref→do_something( );        // Implicitly bind and invoke a method
```

(a)

```
Distr_object obj_ref;           // Declare a systemwide object reference
Local_object* obj_ptr;          // Declare a pointer to local objects
obj_ref = ...;                  // Initialize the reference to a distrib. obj.
obj_ptr = bind(obj_ref);        // Explicitly bind and get ptr to local proxy
obj_ptr→do_something( );        // Invoke a method on the local proxy
```

(b)

Figure 10-7. (a) An example with implicit binding using only global references. (b) An example with explicit binding using global and local references.

Implementation of Object References

It is clear that an object reference must contain enough information to allow a client to bind to an object. 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. Note that part of this information will be provided by an object adapter.

However, there are a number of drawbacks to this scheme. First, if the server's machine crashes and the server is assigned a different end point after recovery, all object references have become invalid. This problem can be solved as is done in DCE: have a local daemon per machine listen to a well known end point and keep track of the server-to-end point assignments in an end point table. When binding a client to an object, we first ask the daemon for the server's current end point. This approach requires that we encode a server ID into the object reference that can be used as an index into the end point table. The server, in turn, is always required to register itself with the local daemon.

2.7.3 Static versus Dynamic Remote Method Invocations

Q14. Explain Static Vs Dynamic RMI.

Ans :

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.

An essential difference between an RMI and an RPC is that RMIs generally support

System wide object references as explained above. Also, it is not necessary to have only general-purpose client-side and server-side stubs available. Instead, we can more easily accommodate object-specific stubs as we also explained. The usual way to provide RMI support is to specify the object's interfaces in an interface definition language, similar to the approach followed with RPCs.

Alternatively, we can make use of an object-based language such as Java, that will handle stub generation automatically. This approach of using predefine interface definitions is generally referred to as **static invocation**. Static invocations require that the interfaces of an object are known when the client application is being developed.

It also implies that if interfaces change, then the client application must be recompiled before it can make use of the new interfaces.

As an alternative, method invocations can also be done in a more dynamic fashion.

In particular, it is sometimes convenient to be able 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);

Where object identifies the distributed object, method is a parameter specifying exactly which method should be invoked, input-parameters is a data structure that holds the values of that method's input parameters, and output-parameters refers to a data structure where output values can be stored.

For example, consider appending an integer int to a file object fobject, for which the object provides the method append.In this case, a static invocation would take the form

fobject.append(int);

Whereas the dynamic invocation looks like

invoke(fobject,id(append),int);

Where the operation id(append) returns an identifier for the method append.

To illustrate the usefulness of dynamic invocations, consider an object browser that is used to examine sets of objects. Assume that the browser supports remote object invocations. Such a browser

is capable of binding to a distributed object and subsequently presenting the object's interface to its user. The user could then be asked to choose a method and provide values for its parameters, after which the browser can do the actual invocation. Typically, such an object browser should be developed to support any possible interface. Such an approach requires that interfaces can be inspected at runtime, and that method invocations can be dynamically constructed.

Another application of dynamic invocations is a batch processing service to which invocation requests can be handed along with a time when the invocation should be done. The service can be implemented by a queue of invocation requests, ordered by the time that invocations are to be done. The main loop of the service would simply wait until the next invocation is scheduled, remove the request from the queue, and call invoke as given above.

2.7.4 Parameter Passing

Q15. Explain about parameter passing in RMI.

Ans :

Because most RMI systems support system wide object references, passing parameters in method invocations is generally less restricted than in the case of RPCs. However, there are some subtleties that can make RMIs trickier than one might initially expect as we briefly discuss in the following pages.

Let us first consider the situation that there are only distributed objects. In other words, all objects in the system can be accessed from remote machines. In that case, we can consistently use object references as parameters in method invocations. References are passed by value, and thus copied from one machine to the other. When a process is given an object reference as the result of a method invocation, it can simply bind to the object referred to when needed later.

Unfortunately, using only distributed objects can be highly inefficient, especially when objects are small, such as integers, or worse yet, Booleans. Each invocation by a client that is not co located in the same server as the object generates a request between different address spaces or, even worse; between different machines. Therefore, references to remote objects and those to local objects are often treated differently.

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. However, when the reference refers to a local object, that is an object in the same address space as the client, the referred object is copied as a whole and passed along with the invocation .In other words; the object is passed by value.

These two situations are illustrated in Fig. 10-8, 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 0 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 02 residing at machine B, which is also used as a parameter. When calling the server, a copy of 0 1 is passed to the server on machine C, along with only a copy of the reference to 02.

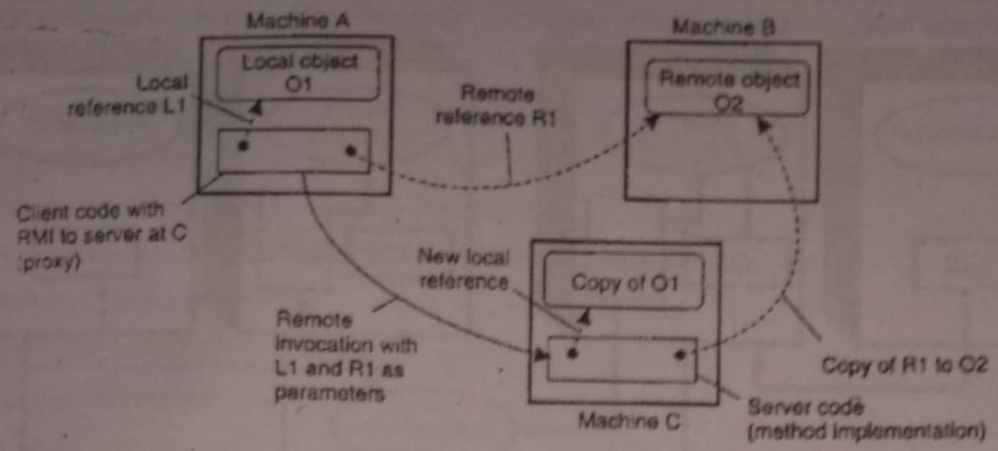


Fig. : The situation when passing an object by reference or by value

2.8 Message-Oriented Communication

Q16. What is message oriented communication ?

Ans :

Remote procedure calls and remote object invocations contribute to hiding communication in distributed systems, that is, they enhance access transparency.

Unfortunately, neither mechanism is always appropriate. In particular, when it cannot be assumed that the receiving side is executing at the time a request is issued, alternative communication services are needed. Likewise, the inherent synchronous nature of RPCs, by which a client is blocked until its request has been processed, sometimes needs to be replaced by something else. That something else is messaging.

2.8.1 Persistence and synchronicity in Communication

Q17. Explain about Persistence and Synchronicity in Communication.

Ans :

- **Persistent communication**

In this method, the communication system receives the message that is to be transmitted. It stores this message as long as it takes to deliver to the receiver. i.e. the communication server stores the message till it is received by another communication server.

Advantages

1) The sending application need not continue execution after submitting the message.

2) During the submission of the message, the receiving application need not be executing.

Fig 2.2 shows the persistent communication system where the communication server sends message to another communication server.

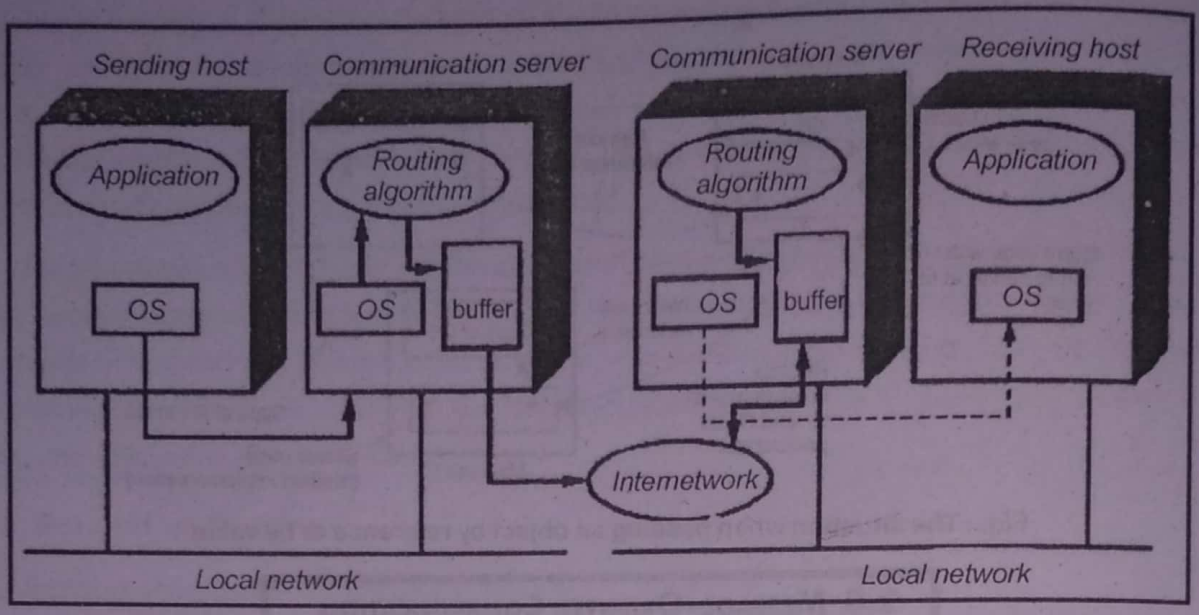


Fig. : Example of persistent communication

This type of communication is similar to **Pony express postal service**. In this service, the letters are deposited at the local post office. Here the letters are sorted as per their destination and sent to the next post office on the route. Again, in this post office they are sorted and sent to the next post office in the route. This process continues t ill the letter reaches the final destination on the route. In the final post office, the letters are again sorted area wise and the postman delivers it to various consumers. This process is shown in Fig 2.3.

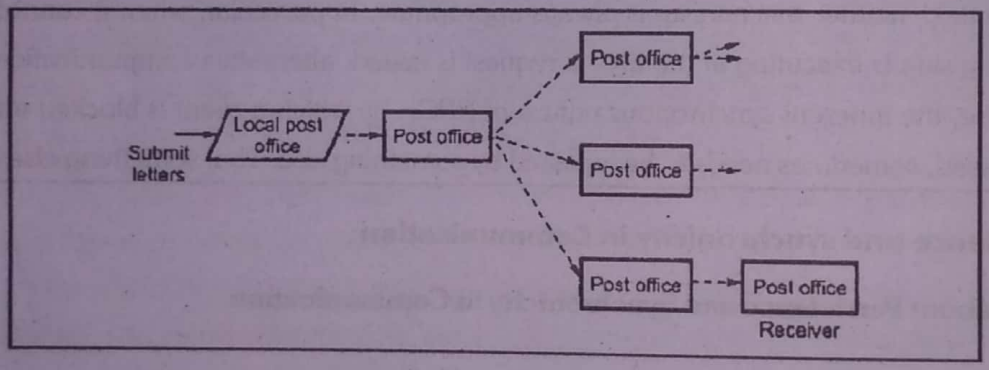


Fig. : Pony express days of persistent communication

- **Transient communication**

  In transient communication, the communi-cation system stores the message till the sending and receiving applications are executing.

  In the Fig.2.2 above of persistent communi-cation if the communication server is not able to deliver message to the next communi-cation server, the message is discarded.

- **Asynchronous communication**

  In this type, the sender continues immediately after submitting the message for transmission. The local buffer at the sending host stores the message.

- **Synchronous communication**

  In this type of communication, the sender is blocked, till its message is stored at the receiving host in its local buffer.

Synchronous communication in its strongest form says that sender should be blocked till the receiver has received the message.

The above types of communications results in various combinations.

1) Persistent asynchronous communication
2) Persistent synchronous communication
3) Transient asynchronous communication
4) Receipt based transient synchronous communication
5) Delivery based transient synchronous communication
6) Response based transient synchronous communication

2.8.2 Message-Oriented Transient Communication

Q18. Write a short note on Berkeley sockets.

Ans :

Many distributed systems and applications are built directly on top of the simple message-oriented model offered by the transport layer. To better understand and appreciate the message-oriented systems as part of middleware solutions.

Berkeley Sockets

The sockets interface has introduced in the 1970s in Berkeley UNIX.

Conceptually, a socket is a 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.

A socket forms an abstraction over the actual communication end point that is used by the local operating system for a specific transport protocol. The socket primitives for TCP are shown in Fig.

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

Fig. : The socket primitives for TCPIP

Servers generally execute the first four primitives, normally in the order given. When calling the socket primitive, the caller creates a new communication end point for a specific transport protocol. Internally, creating a communication end point means that the local operating system reserves resources to accommodate sending and receiving messages for the specified protocol.

The bind primitive associates a local address with the newly-created socket.

For example, a server should bind the IP address of its machine together with a (possibly well-known) port number to a socket. Binding tells the operating system that the server wants to receive messages only on the specified address and port.

The listen primitive is called only in the case of connection-oriented communication. It is a nonblocking call that allows the local operating system to reserve enough buffers for a specified maximum number of connections that the caller is willing to accept.

A call to accept blocks the caller until a connection request arrives. When a request arrives, the local operating system creates a new socket with the same properties as the original one, and returns it to the caller. This approach will allow the server to, for example, fork off a process that will subsequently handle the actual communication through the new connection. The server, in the meantime, can go back and wait for another connection request on the original socket.

Let us now take a look at the client side. Here, too, a socket must first be created using the socket primitive, but explicitly binding the socket to a local address is not necessary, since the operating system can dynamically allocate a port when the connection is set up. The connect primitive requires that the caller specifies the transport-level address to which a connection request is to be sent. The client is blocked until a connection has been set up successfully, after which both sides can start exchanging information through the send and receive primitives.

Finally, closing a connection is symmetric when using sockets, and is established by having both the client and server call the close primitive. The general pattern followed by a client and server for connection-oriented communication using sockets is shown in Fig. 4-15.

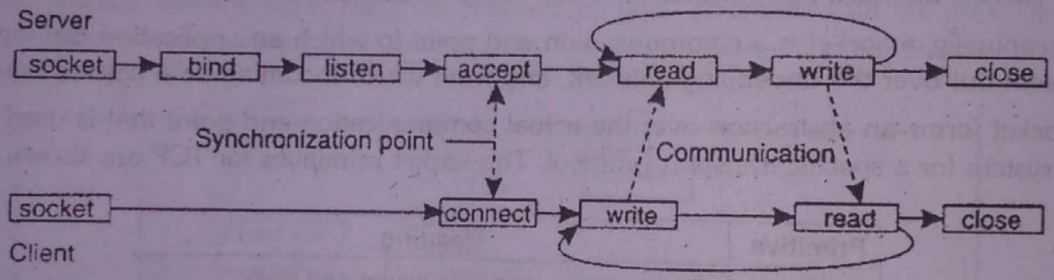


Fig. : Connection-oriented communication pattern using sockets

2.8.3 Message-Oriented persistent Communication

Q19. Write about message queuing system.

Ans :

Message-queuing systems provide extensive support for persistent asynchronous communication. The essence of these systems is that they offer intermediate-term storage capacity for messages, without requiring either the sender or receiver to be active during message transmission.

An important difference with Berkeley sockets and MPI is that message-queuing systems are typically targeted to support message transfers that are allowed to take minutes instead of seconds or milliseconds.

Message-Queuing Model

The basic idea behind a message-queuing system is that applications communicate by inserting messages in specific queues.

These messages are forwarded over a series of communication servers and are eventually delivered to the destination, even if it was down when the message was sent. In practice, most communication

servers are directly connected to each other. In other words, a message is generally transferred directly to a destination server. In principle, each application has its own private queue to which other applications can send messages.

A queue can be read only by its associated application, but it is also possible for multiple applications to share a single queue.

An important aspect of message-queuing systems is that a sender is generally given only the guarantees that its message will eventually be inserted in the recipient's queue. No guarantees are given about when, or even if the message will actually be read, which is completely determined by the behavior of the recipient.

Once a message has been deposited in a queue, it will remain there until it is removed, irrespective of whether its sender or receiver is executing. This gives us 4 combinations with respect to the execution mode of the sender and receiver, as shown in Fig.

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

Fig. : Basic interface to a queue in a message-queueing system

In Fig.4-17(a), both the sender and receiver execute during the entire transmission of a message. In.Fig. 4-17(b), only the sender is executing, while the receiver is passive, that is, in a state in which message delivery is not possible.

Nevertheless, the sender can still send messages. The combination of a passive sender and an executing receiver is shown in Fig. 4-17(c). In this case, the receiver can read messages that were sent to it, but it is not necessary 'that their respective senders are executing as well. Finally, in Fig. 4-17(d), we see the situation that the system is storing (and possibly transmitting) messages even while sender and receiver are passive.

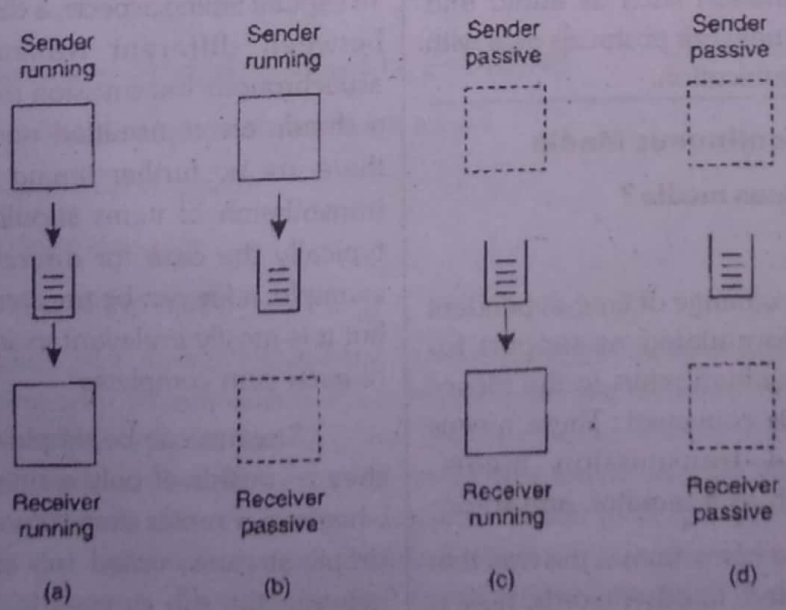


Fig. : Four combinaTions for loosely-coupled communications using queues.

2.9 Stream-Oriented Communication

Q20. Write about steam oriented communication with continuous media and data streams.

Ans :

Communication is concentrated on exchanging more or less independent and complete units of information. Examples include a request for invoking a procedure, the reply to such a request, and messages exchanged between applications as in message-queuing systems.

There are also forms of communication in which timing plays a crucial role. Consider, for example, an audio stream built up as a sequence of 16-bit samples, each representing the amplitude of the sound wave as is done through Pulse Code Modulation (PCM).

Also assume that the audio stream represents CD quality, meaning that the original sound wave has been sampled at a frequency of 44, 100Hz. To reproduce the original sound, it is essential that the samples in the audio stream are played out in the order they appear in the stream, but also at intervals of exactly 1/44,100 sec. playing out at a different rate will produce an incorrect version of the original sound.

A distributed system should offer to exchange time-dependent information such as audio and video streams. Various network protocols deal with stream-oriented communication.

2.9.1 Support for Continuous Media

Q21. What is continuous media ?

Ans :

Support for the exchange of time-dependent information is often formulated as support for continuous media. A medium refers to the means by which information is conveyed. These means include storage and transmission media, presentation media such as a monitor, and so on.

An important type of medium is the way that information is *represented*. In other words, how is information encoded in a computer system? Different representations are used for different types of information.

For example, text is generally encoded as ASCII or Unicode. Images can be represented in different formats such as GIF or IPEG. Audio streams can be encoded in a computer system by, for example, taking 16-bit samples using PCM.

In continuous (representation) media, the temporal relationships between different data items are fundamental to correctly interpreting what the data actually means.

2.9.2 Data Stream

Q22. What is data stream ?

Ans :

To capture the exchange of time-dependent information, distributed systems generally provide support for data streams. A data stream is nothing but a sequence of data units. Data streams can be applied to discrete as well as continuous media.

For example, UNIX pipes or TCPIIP connections are typical examples of (byte-oriented) discrete data streams. Playing an audio file typically requires setting up a continuous data stream between the file and the audio device.

Timing is crucial to continuous data streams. To capture timing aspects, a distinction is often made between different transmission modes. In asynchronous transmission mode the data items in a stream are transmitted one after the other, but there are no further timing constraints on when transmission of items should take place. This is typically the case for discrete data streams. For example, a file can be transferred as a data stream, but it is mostly irrelevant exactly when the transfer of each item completes.

Streams can be simple or complex. A simple stream consists of only a single sequence of data, whereas a complex stream consists of several related simple streams, called sub streams. The relation between the sub streams in a complex stream is often also time dependent. For example, stereo

audio can be transmitted by means of a complex stream consisting of two sub streams, each used for a single audio channel. It is important, however, that those two sub streams are continuously synchronized.

This general architecture reveals a number of important issues that need to be dealt with. In the first place, the multimedia data, notably video and to a lesser extent audio, will need to be compressed substantially in order to reduce the required storage and especially the network capacity. More important from the perspective of communication are controlling the quality of the transmission and synchronization issues.

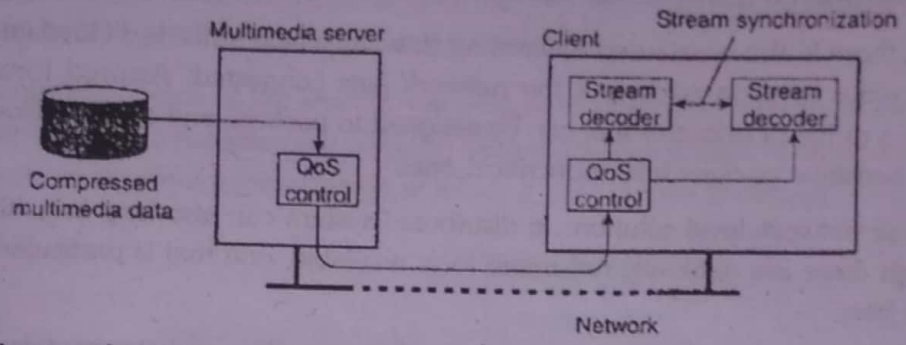


Fig. : A general architecture for streaming stored multimedia data over a network

2.10 Streams and Quality of Service

Q23. Explain about streams and QOS.

Ans :

Timing (and other nonfunctional) requirements are generally expressed as Quality of Service (QoS) requirements. These requirements describe what is needed from the underlying distributed system and network to ensure that, for example, the temporal relationships in a stream can be preserved.

QoS for continuous data streams mainly concerns timeliness, volume, and reliability. From an application's perspective, in many cases it boils down to specifying a few important properties (Halsall, 2001):

1. The required bit rate at which data should be transported.
2. The maximum delay until a session has been set up.
3. The maximum end-to-end delay.
4. The maximum delay variance..
5. The maximum round-trip delay.

It should be noted that many refinements can be made to these specifications, as explained. However, when dealing with stream-oriented communication that is based on the Internet protocol stack, we simply have to live with the fact that the basis of communication is formed by an extremely simple, best-effort datagram service: IP. When the going gets tough, as may easily be the case in the Internet, the specification of IP allows a protocol implementation to drop packets whenever it sees fit. Many, if not all distributed systems that support stream-oriented communication, are currently built on top of the Internet protocol stack. So much for QoS specifications. (Actually, IP does provide some QoS support, but it is rarely implemented.)

Enforcing QoS

Given that the underlying system offers only a best-effort delivery service, a distributed system can try to conceal as much as possible of the *lack* of quality of service. Fortunately, there are several mechanisms that it can deploy.

A sending host can essentially mark outgoing packets as belonging to one of several classes, including an expedited forwarding class that essentially specifies that a packet should be forwarded by the current router with absolute priority (Davie et al., 2002).

In addition, there is also an assured forwarding class, by which traffic is divided into four subclasses, along with three ways to drop packets if the network gets congested. Assured forwarding therefore effectively defines a range of priorities that can be assigned to packets, and as such allows applications to differentiate time-sensitive packets from noncritical ones.

Besides these network-level solutions, a distributed system can also help in getting data across to receivers. Although there are generally not many tools available, one that is particularly useful is to use buffers to reduce jitter.

The principle is simple, as shown in Fig. 4-27. Assuming that packets are delayed with a certain variance when transmitted over the network, the receiver simply stores them in a buffer for a maximum amount of time.

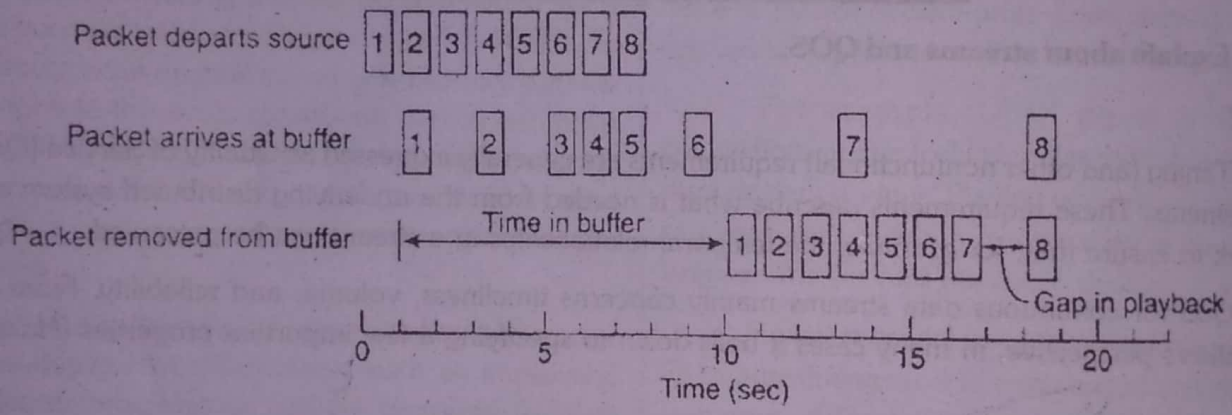


Fig. : Using a Buffer to reduce jitter

2.11 Stream Synchronization

Q24. Write about stream synchronization.

Ans :

An important issue in multimedia systems is that different streams, possibly in the form of a complex stream, are mutually synchronized.

Synchronization of streams deals with maintaining temporal relations between streams.

Two types of synchronization occur.

The simplest form of synchronization is that between a discrete data stream and a continuous data stream. Consider, for example, a slide show on the Web that has been enhanced with audio. Each slide is transferred from the server to the client in the form of a discrete data stream. At the same time, the client should play out a specific (part of an) audio stream that matches the current slide that is also

fetched from the server. In this case, the audio stream is to be 'synchronized with the presentation of slides.

A more demanding type of synchronization is that between continuous data streams. A daily example is playing a movie in which the video stream needs to be synchronized with the audio, commonly referred to as lip synchronization.

Another example of synchronization is playing a stereo audio stream consisting of two sub streams, one for each channel. Proper play out requires that the two sub streams are tightly synchronized: a difference of more than 20 users can distort the stereo effect.

Synchronization takes place at the level of the data units of which a stream is made up. In other words, we can synchronize two streams only between data units. The choice of what exactly a data unit is depends very much on the level of abstraction at which a data stream is viewed.

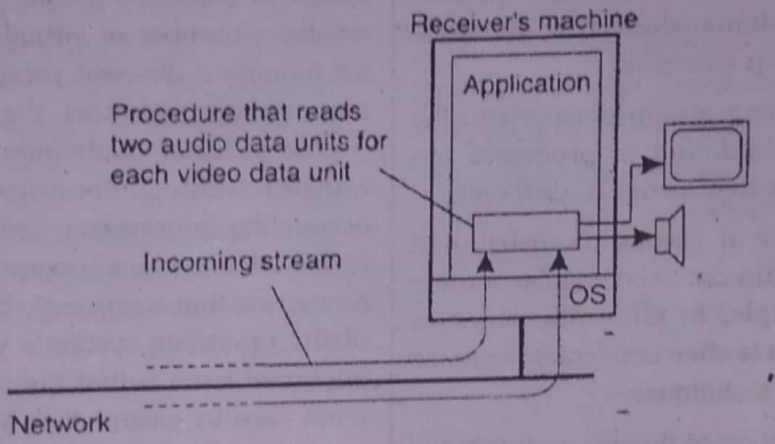


Fig. : The principle of explicit synchronization on the level data units

UNIT III

Process: Threads: Introduction to Threads, Threads in Distributed Systems, Clients: user Interfaces, Client-Side Software for Distribution Transparency, Servers: General Design Issues, Object Servers, Software Agents: Software Agents in Distributed Systems, Agent Technology, Naming: Naming Entities: Names, Identifiers, and Address, Name Resolution, The Implementation of a Name System, Locating Mobile Entities: Naming verses Locating Entities, Simple Solutions, Home-Based Approaches, Hierarchical Approaches

PROCESS

The concept of a process originates from the field of operating systems where it is generally defined as a program in execution.

From an operating-system perspective, the management and scheduling of processes are perhaps the most important issues to deal with.

However, when it comes to distributed systems, other issues turn out to be equally or more important. For example, to efficiently organize client-server systems, it is often convenient to make use of multithreading techniques.

A main contribution of threads in distributed systems is that they allow clients and servers to be constructed such that communication and local processing can overlap, resulting in a high level of performance.

3.1 THREADS

Q1. Discuss processes Vs Threads.

Ans :

Although processes form a building block in distributed systems, practice indicates that the granularity of processes as provided by the operating systems on which distributed systems are built is not sufficient.

Instead, multiple threads of control per process make it much easier to build distributed applications and to attain better performance.

3.1.1 Introduction to Threads

To understand the role of threads in distributed systems, it is important to understand what a process is, and how processes and threads relate. To execute a program, an operating system creates a number of virtual processors, each one for running a different program. To keep track of these virtual processors, the operating system has a process table, containing entries to store CPU register values, memory maps, open files, accounting information. privileges, etc. A process is often defined as a program in execution, that is, a program that is currently being executed on one of the operating system's virtual processors. An important issue is that the operating system takes great care to ensure that independent processes cannot maliciously or inadvertently affect the correctness of each other's behavior.

In other words, the fact that multiple processes may be concurrently sharing the same CPU and other hardware resources is made transparent. Usually, the operating system requires hardware support to enforce this separation.

This concurrency transparency comes at a relatively high price. For example, each time a process is created, the operating system must create a complete independent address space. Allocation can mean initializing memory segments by, for example, zeroing a data segment, copying the associated program into a text segment, and setting up a stack for temporary data. Likewise. switching the CPU between two processes may be relatively expensive as well. Apart from saving the CPU context (which consists of register values, program counter, stack pointer, etc.), the operating system will also have to modify registers of the memory management unit (MMU) and invalidate address

translation caches such as in the translation look aside buffer (TLB). In addition, 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.

Like a process, a thread executes its own piece of code, independently from other threads. However, in contrast to processes, no attempt is made to achieve a high degree of concurrency transparency if this would result in performance degradation.

Therefore, a thread system generally maintains only the minimum information to allow a CPU to be shared by several threads. In particular, a thread context often consists of nothing more than the CPU context, along with some other information for thread management. For example, a thread system may keep track of the fact that a thread is currently blocked on a mutex variable, so as not to select it for execution. Information that is not strictly necessary to manage multiple threads is generally ignored. For this reason, protecting data against inappropriate access by threads within a single process is left entirely to application developers.

There are two important implications of this approach. First of all, the performance of a multithreaded application need hardly ever be worse than that of its single-threaded counterpart. In fact, in many cases, multithreading leads to a performance gain. Second, because threads are not automatically protected against each other the way processes are, development of multithreaded applications requires additional intellectual effort. Proper design and keeping things simple, as usual, help a lot. Unfortunately, current practice does not demonstrate that this principle is equally well understood.

3.1.2 Thread Usage in non distributed Systems

Q2. Explain about usage of threads in non distributed system.

Ans :

Before discussing the role of threads in distributed systems, let us first consider their usage in traditional, non distributed systems. For example consider a spread sheet program. An important property of a spreadsheet program is that, it maintains the relational dependencies between different cells, often from different spreadsheets. Therefore, whenever a cell is modified, all dependent cells automatically updated. When a user changes the value in a single cell, such a modification can trigger a large series of computations. If there is only a single thread of control, computation cannot proceed while the program is waiting for input. Likewise, it is not easy to provide input while dependencies are being calculated.

The easy solution is to have at least two threads of control: one for handling interaction with the user and one for updating the spreadsheet. In the mean time, a third thread could be used for backing up the spreadsheet to disk while the other two are doing their work.

Another advantage of multithreading is that it becomes possible to exploit parallelism when executing the program on a multiprocessor system. Multithreading for parallelism is becoming increasingly important with the availability of relatively cheap multiprocessor workstations. Such computer systems are typically used for running servers in client-server applications.

Multithreading is also useful in the context of large applications. Such applications are often developed as a collection of cooperating programs, each to be executed by a separate process. This approach is typical for a UNIX environment. Cooperation between programs is implemented by means of inter process communication (IPC) mechanisms.

For UNIX systems, these mechanisms typically include (named) pipes, message queues, and shared memory segments. The major drawback of all IPC mechanisms is that communication often requires extensive context switching, shown at three different points in Figure.

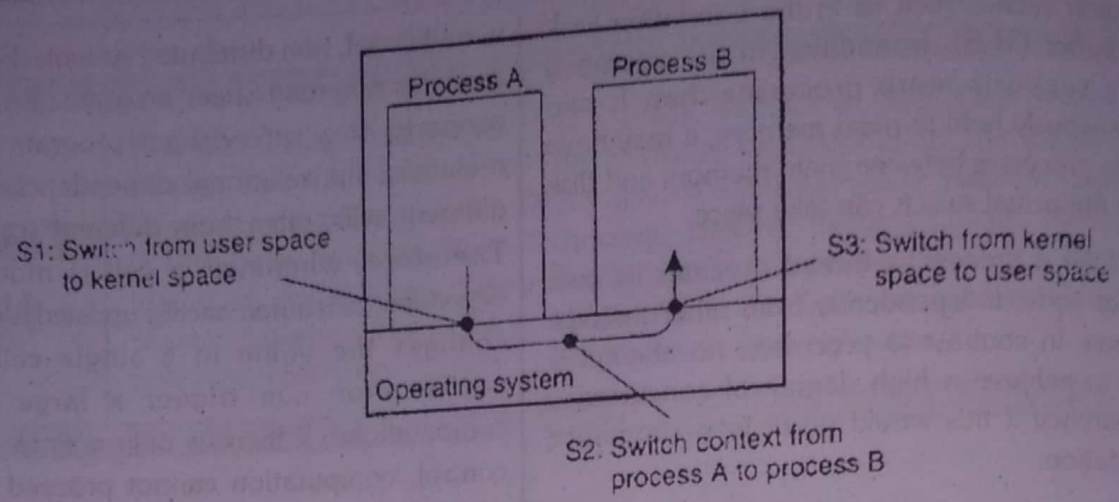


Fig. : Context switching as the result of IPC

Because IPC requires kernel intervention, a process will generally first have to switch from user mode to kernel mode, shown as S 1 in Fig. 3-1. This requires changing the memory map in the MMU, as well as flushing the TLB. Within the kernel, a process context switch takes place (52 in the figure), after which the other party can be activated by switching from kernel mode to user mode again (53 in Fig. 3-1). The latter switch again requires changing the MMU map and flushing the TLB.

Instead of using processes, an application can also be constructed such that different parts are executed by separate threads. Communication between those parts is entirely dealt with by using shared data. Thread switching can sometimes be done entirely in user space, although in other implementations, the kernel is aware of threads and schedules them. The effect can be a dramatic improvement in performance.

Finally, there is also a pure software engineering reason to use threads: many applications are simply easier to structure as a collection of cooperating threads.

Think of applications that need to perform several (more or less independent) tasks. For example, in the case of a word processor, separate threads can be used for handling user input, spelling and grammar checking, document layout, index generation, etc.

3.2 Threads in Distributed Systems

Q3. Explain about multi threaded clients and servers.

Ans :

An important property of threads is that they can provide a convenient means of allowing blocking system calls without blocking the entire process in which the thread is running. This property makes threads particularly attractive to use in distributed systems as it makes it much easier to express communication in the form of maintaining multiple logical connections at the same time.

Multithreaded Clients

To establish a high degree of distribution transparency, distributed systems that operate in wide-area networks may need to conceal long inter process message propagation times. A typical example where this happens is in Web browsers.

In many cases, a Web document consists of an HTML file containing plain text along with a collection of images, icons, etc. To fetch each element of a Web document, the browser has to set up a TCP/IP connection, read the incoming data, and passes it to a display component.

A Web browser often starts with fetching the HTML page and subsequently displays it. To hide communication latencies as much as possible, some browsers start displaying data while it is still coming in. As it turns out, developing the browser as a multithreaded client simplifies matters considerably. As soon as the main HTML file has been fetched, separate threads can be activated to take care of fetching the other parts. Each thread sets up a separate connection to the server and pulls in the data. Setting up a connection and reading data from the server can be programmed using the standard system calls, assuming that a blocking call does not suspend the entire process.

There is another important benefit to using multithreaded Web browsers in which several connections can be opened simultaneously. When using a multithreaded client, connections may be set up to different replicas, allowing data to be transferred in parallel, effectively establishing that the entire Web document is fully displayed in a much shorter time than with a non replicated server.

Multithreaded Servers

Although there are important benefits to multithreaded clients, as we have seen, the main use of multithreading in distributed systems is found at the server side.

Practice shows that multithreading not only simplifies server code considerably, but also makes it much easier to develop servers that exploit parallelism to attain high performance, even on uniprocessor systems.

However, now that multiprocessor computers are widely available as general-purpose workstations, multithreading for parallelism is even more useful.

To understand the benefits of threads for writing server code, consider the organization of a file server that occasionally has to block waiting for the disk.

The file server normally waits for an incoming request for a file operation, subsequently carries out the request, and then sends back the reply.

One possible, and particularly popular organization is shown in Fig. 3-3. Here one thread, the dispatcher, reads incoming requests for a file operation. The requests are sent by clients to a well-known end point for this server. After examining the request, the server chooses an idle (i.e., blocked) worker thread and hands it the request.

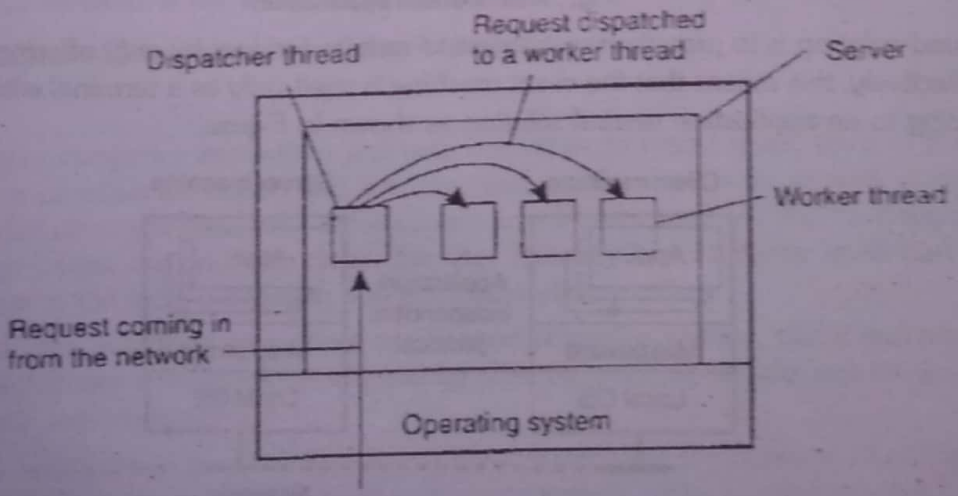


Fig. : A multithreaded server organised in a dispatcher/worker model

The worker proceeds by performing a blocking read on the local file system, which may cause the thread to be suspended until the data are fetched from disk. If the thread is suspended, another thread is selected to be executed.

For example, the dispatcher may be selected to acquire more work. Alternatively, another worker t. read can be selected that is now ready to run.

3.3 CLIENTS

3.3.1 Networked User Interfaces

Q4. Write a short note on networked user interfaces with an example.

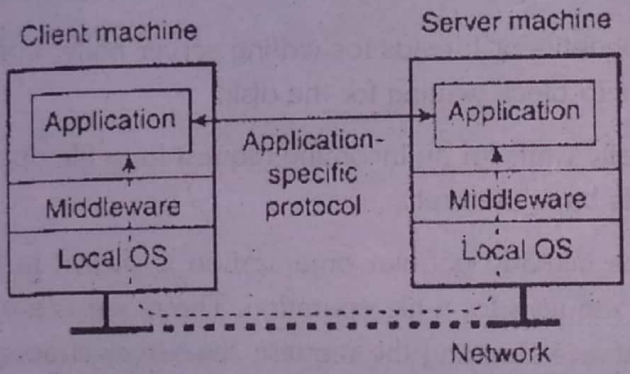
Ans :

The client-server model is the basis for communication in distributed systems.

Client server interaction can be achieved in a specific manner.

A major task of client machines is to provide the means for users to interact with remote servers. There are 2 ways in which this interaction can be supported.

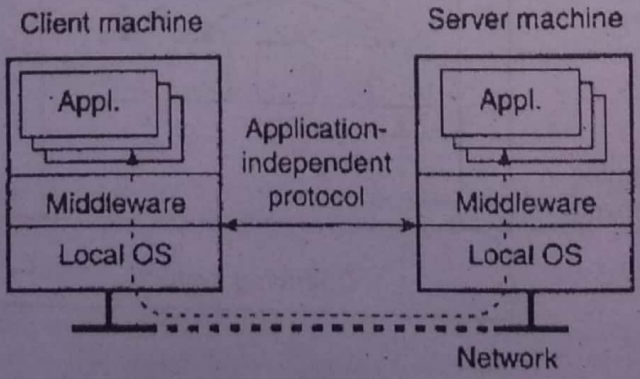
First, for each remote service the client machine will have a separate counterpart that can contact the service over the network. A typical example is an agenda running on a user's PDA that needs to synchronize with a remote, possibly shared agenda. In this case, an application-level protocol will handle the synchronization, as shown in Figure.



(a)

Fig. : A networked application

A second solution is to provide direct access to remote services by only offering a convenient user interface. Effectively, this means that 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)

In the case of networked user interfaces, everything is processed and stored at the server. This thin-client approach is receiving more attention as Internet connectivity increases, and hand-held devices are becoming more sophisticated. As we argued in the previous chapter, thin-client solutions are also popular as they ease the task of system management. Let us take a look at how networked user interfaces can be supported.

Example: The X Window System

Perhaps one of the oldest and still widely-used networked user interfaces is the X Window system. The X Window System, generally referred to simply as X, is used to control bit-mapped terminals, which include a monitor, keyboard ,and a pointing device such as a mouse.

In a sense, X can be viewed as that part of an operating system that controls the terminal. The heart of the system is formed by what we shall call the X kernel.

It contains all the terminal-specific device drivers, and as such, is generally highly hardware dependent.

The X kernel offers a relatively low-level interface for controlling the screen, but also for capturing events from the keyboard and mouse. This interface is made available to applications as a library called Xlib. This general organization is shown in Figure.

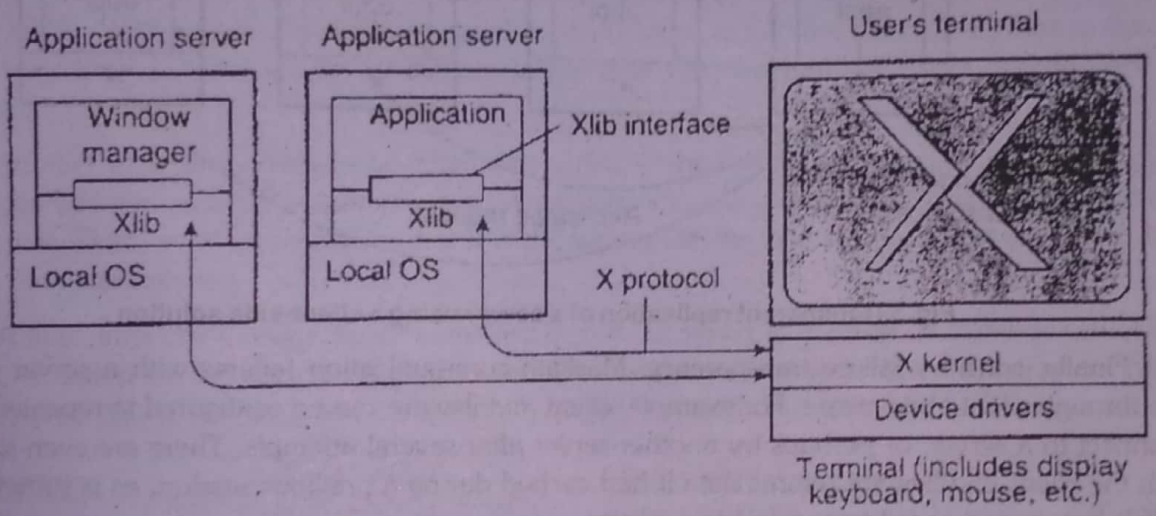


Fig. : The basic organisation of the X Window System

3.3.2 Client-Side Software for Distribution Transparency

Q5. Write about Client-Side Software for Distribution Transparency.

Ans :

Client software comprises more than just user interfaces. In many cases, parts of the processing and data level in a client-server application are executed on the client side as well. A special class is formed by embedded client software, such as for automatic teller machines (ATMs), cash registers, barcode readers, TV set-top boxes, etc. In these cases, the user interface is a relatively small part of the client software, in contrast to the local processing and communication facilities.

Besides the user interface and other application-related software, client software comprises components for achieving distribution transparency. Ideally, a client should not be aware that it is communicating with remote processes.

In contrast, distribution is often less transparent to servers for reasons of performance and correctness. Replicated servers sometimes need to communicate in order to establish that operations are performed in a specific order at each replica.

Access transparency is generally 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.

There are different ways to handle location, migration, and relocation transparency.

Using a convenient naming system is crucial, In many cases, cooperation with client-side software is also important. For example, when a client is already bound to a server, the client can be directly informed when the server changes location. In this case, the client's middleware can hide the server's current geographical location from the user, and also transparently rebind to the server if necessary.

In a similar way, many distributed systems implement replication transparency by means of client-side solutions. For example, imagine a distributed system with replicated servers; such replication can be achieved by forwarding a request to each replica, as shown in Fig. 3-10. Client-side software can transparently collect all responses and pass a single response to the client application.

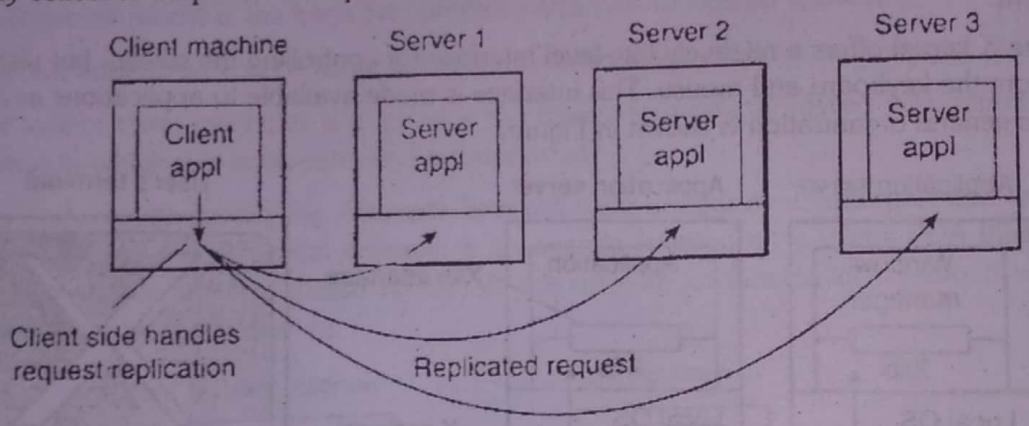


Fig. : Transparent replication of a server using a client-side solution

Finally, consider failure transparency. Masking communication failures with a server is typically done through client middleware. For example, client middleware can be configured to repeatedly attempt to connect to a server, or perhaps try another server after several attempts. There are even situations in which the client middleware returns data it had cached during a previous session, as is sometimes done by Web browsers that fail to connect to a server.

Concurrency transparency can be handled through special intermediate servers, notably transaction monitors, and requires less support from client software.

Likewise, persistence transparency is often completely handled at the server.

3.4 SERVERS

3.4.1 General Design Issues

Q6. Discuss about general design issues of a Server.

Ans :

A Server is the major component in the communication. A serer can handle multiple clients. There are a number of general design issues for servers.

A server is a process implementing a specific service on behalf of a collection of clients. In essence, each server is organized in the same way: it waits for an incoming request from a client and subsequently ensures that the request is taken care of, after which it waits for the next incoming request.

There are several ways to organize servers. In the case of an **iterative server**, the server itself handles the request and, if necessary, returns a response to the requesting client. A **concurrent server** does not handle the request itself, but passes it to a separate thread or another process, after which it immediately waits for the next incoming request. A multithreaded server is an example of a concurrent server.

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.

Another issue is where clients contact a server. In all cases, 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? One approach is to globally assign end points for well-known services.

For example, servers that handle Internet FTP requests always listen to TCP port 21. Likewise, 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), and are documented in Reynolds and Postel (1994). With assigned end points, the client only needs to find the network address of the machine where the server is running. As we explain in the next chapter, name services can be used for that purpose.

Another issue that needs to be taken into account when designing a server is whether and how a server can be interrupted. For example, consider a user who has just decided to upload a huge file to an FTP server. Then, suddenly realizing that it is the wrong file, he wants to interrupt the server to cancel further data transmission.

A final, important design issue is whether or not the server is stateless. 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. A Web server, for example, is stateless. It merely responds to incoming HTTP requests.

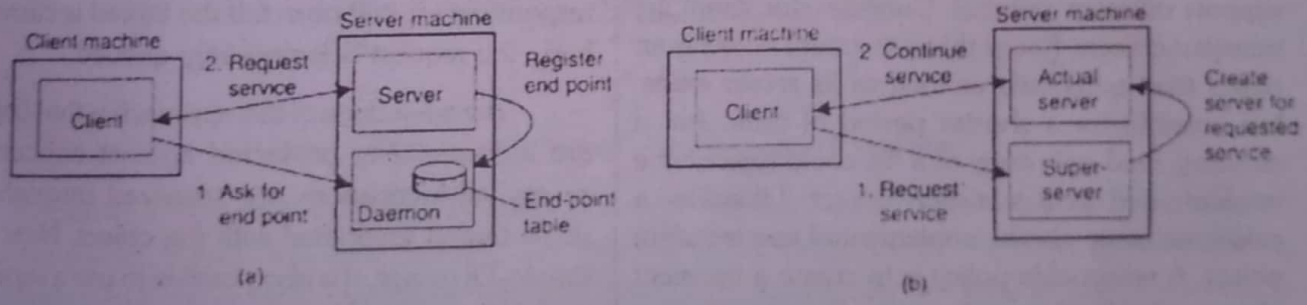


Fig.(a) : Client-to-server binding using a daemon (b) Client-to-server binding using a superserver

3.4.2 Object Servers

Q7. Explain the use of Object Servers.

Ans :

A key role in object-based distributed systems is played by object servers, that is, the server designed to host distributed objects.

An object server is a server tailored to support distributed objects. The important difference between a general object server and other (more traditional) servers is that an object server by itself does not provide a specific service. Specific services are implemented by the objects that reside in the server.

Essentially, the server provides only the means to invoke local objects, based on requests from remote clients. As a consequence, it is relatively easy to change services by simply adding and removing 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. Also, there are differences in the way an object server invokes its objects. For example, in a multithreaded server, each object may be assigned a separate thread, or a separate thread may be used for each invocation request.

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. Unfortunately, such an approach is generally inflexible and often unnecessarily constrains developers of distributed objects.

A much better approach is for a server to support different policies. Consider, for example, transient objects. Recall that a transient object is an object that exists only as long as its server exists, but possibly for a shorter period of time. An in memory, read-only copy of a file could typically be implemented as a transient object. Likewise, a calculator could also be implemented as a transient object. A reasonable policy is to create a transient object at the first invocation request and to destroy it as soon as no clients are bound to it anymore.

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. Therefore, an alternative policy is sometimes to create all transient objects at the time the server is initialized, at the cost of consuming resources even when no client is making use of the object.

In a similar fashion, a server could follow the policy that each of its objects is placed in a memory segment of its own. In other words, objects share neither code nor data. Such a policy may be necessary when an object implementation does not separate code and data, or when objects need to be separated for security reasons. In the latter case, the server will need to provide special measures, or require support from the underlying operating system, to ensure that segment boundaries are not violated.

The alternative approach is to let objects at least share their code. For example, a database containing objects that belong to the same class can be efficiently implemented by loading the class implementation only once into the server. When a request for an object invocation comes in, the server need only fetch that object's state from the database and execute the requested method.

Likewise, there are many different policies with respect to threading. The simplest approach is to implement the server with only a single thread of control. Alternatively, the server may have several threads, one for each of its objects. Whenever an invocation request comes in for an object, the server passes the request to the thread responsible for that object. If the thread is currently busy, the request is temporarily queued.

The advantage of this approach is that objects are automatically protected against concurrent access: all invocations are serialized through the single thread associated with the object. Neat and simple, Of course, it is also possible to use a separate thread for each invocation request, requiring that objects should have already been protected against concurrent access.

Independent of using a thread per object or thread per method is the choice of whether threads are created on demand or the server maintains a pool of threads. Generally there is no single best policy. Which one to use depends on whether threads are available, how much performance matters, and similar factors.

3.5 Software Agents

Q8. Write a short note on Software agents.

Ans :

Computer science, a **software agent** is a computer program that acts for a user or other program in a relationship of agency, which derives from the Latin agere (to do): an agreement to act on one's behalf.

Such "action on behalf of" implies the authority to decide which, if any, action is appropriate. Agents are colloquially known as bots, from robot. They may be embodied, as when execution is paired with a robot body, or as software such as a chat bot executing on a phone (e.g. Siri) or other computing device.

Software Agents may be autonomous or work together with other agents or people. Software agents interacting with people (e.g. chatbots, human-robot interaction environments) may possess human-like qualities such as natural language understanding and speech, personality or embody humanoid form.

Role of Software Agents in Distributes Systems

The central principle of today's distributed programming is remote procedure calling (RPC). The RPC approach, which was conceived in the 1970s, views computer-to-computer communi-cation as enabling one computer to call a procedure in another. In RPC, all messages go through the network; each either requests or acknowledges a procedure's actions. This approach, however, has its own limitations. Most notably, all interactions between the client and server must go through the network as shown in Figure.

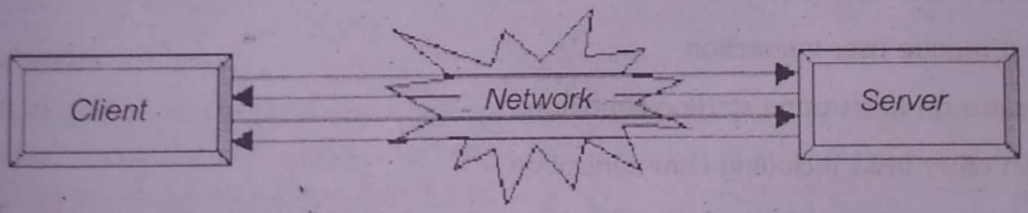


Fig. : RCP - based Client/Server Computing Paradigm

Another approach that is forming a new paradigm for distributed computing is one that employs software agents. Initially this approach was known as Remote Programming. The Remote Programming approach views computer-to computer communication as one computer not only to call procedures in another, but also to supply the procedures to be performed. Each message that goes through the network comprises a procedure that the receiving computer is to perform and data that are its arguments. The procedure and its state are termed a software agent as they represent the sending computer even while they are in the receiving computer as shown in figure

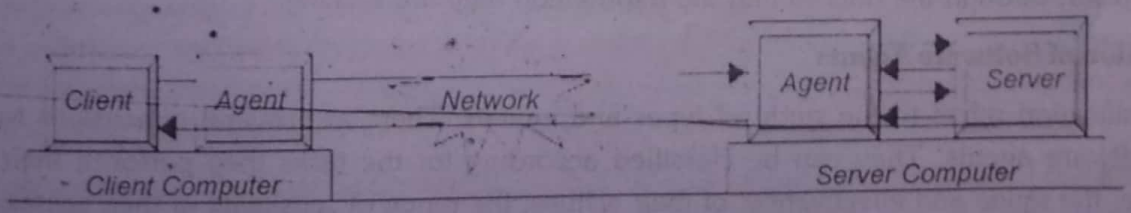


Fig. : mobile agents-based computing paradigm

3.5.1 Agent Technology

Q9. Write about the importance of Agent Technology.

Ans :

"An agent is an entity that :

- acts on behalf of others in an autonomous fashion
- performs its actions in some level of pro activity and reactivity
- Exhibits some levels of the key attributes of learning, co-operation, and mobility."

But what exactly is a software agent? And how does it differ from a software object?

You may think of a software agent as one (or more) software object(s) that conforms to the above characteristics of agents and can be described as inhibiting computers and networks, assisting users with computer-based tasks.

A software agent is a piece of software that functions as an agent for a user or another program, working autonomously and continuously in a particular environment. It is inhibited by other processes and agents, but is also able to learn from its experience in functioning in an environment over a long period of time.

Software agents offer various benefits to end users by automating repetitive tasks. The basic concepts related to software agents are:

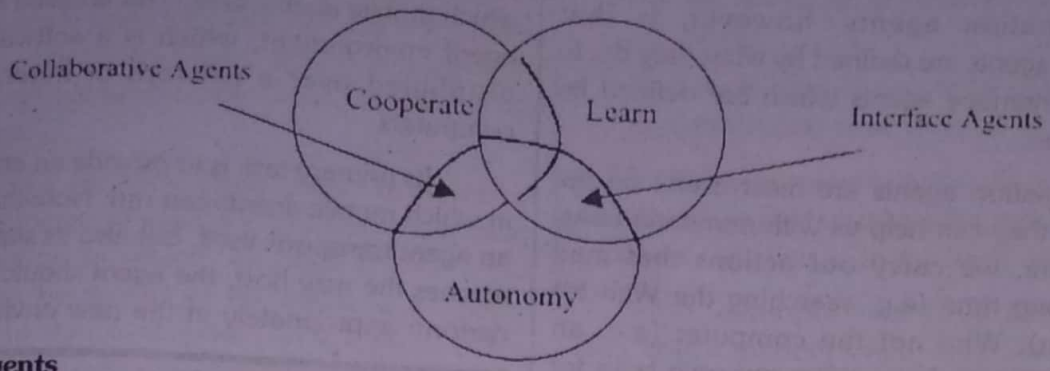
- They are invoked for a task
- They reside in "wait" status on hosts
- They do not require user interaction
- They run status on hosts upon starting conditions
- They invoke other tasks including communication

There are a number of different software agents, including:

- **Buyer Agents or Shopping Bots:** These agents revolve around retrieving network information related to good and services.
- **User or Personal Agents:** These agents perform a variety of tasks such as filling out forms, acting as opponents in games, assembling customized reports and checking email, among other tasks.
- **Monitoring and Surveillance Agents:** These agents observe and report on equipment.
- **Data-Mining Agents:** These agents find trends and patterns in many different sources and allow users to sort through the data to find the information they are seeking.

Classification of Software Agents

Classification refers to the study of types and entities. There are several dimensions to classify existing software agents. They can be classified according to: the tasks they perform; their control architecture; the range and effectiveness of their actions; the range of sensitivity of their senses; or how much internal state they posses.



Interface Agents

Interface agents perform tasks for their owners by emphasizing autonomy and learning. They support and provide assistance to a user learning to use a particular application such as a spreadsheet. The agent here observes the actions being carried out by the user and tries to learn new short cuts, and then it will try to suggest better ways of doing the same task.

The key metaphor underlying interface agents is that of a personal assistant who is collaborating with the user in the same work environment. Interface agents learn to better assist its users in four ways.

- By observing and imitating the user
- Through receiving positive and negative feedback from the user
- By receiving explicit instructions from the user
- By asking other agents for advice

While interface agents ask other agents for advice (learning from peers), their cooperation with other agents however, is limited.

Collaborative Agents

As the proliferation of computer communi-cation networks was a big step toward the development of "virtual societies". Collaboration between individuals (in the virtual society) requires that communication links be established and used effectively. Distributed Artificial Intelligence, which is a subfield of Artificial Intelligence, is concerned with a virtual society of problem solvers (agents) interacting to solve a common problem.

The goal of collaborative agents is to interconnect separately developed collaborative agents, thus enabling the ensemble to function beyond the capabilities of any of its members. Implementing efficient ways of cooperation among agents is actually one of the central issues for Multi-Agent Systems development. One of the motivations for having collaborative agents is to provide solutions to inherently distributed problems, such as distributed sensor network, or air traffic control.

Information Agents

The explosive growth of information on the Word-Wide Web has given a rise to information agents (also known as Internet agents) in the hope that these agents will be able to help us manage, manipulate, or collate information from many distributed resources. One may notice however, that information agents seem a bit similar to interface agents.

However, it is important to note that not all types of agents discussed here started at the same time. So, with the explosive growth of information, and the need for tools to manage such information, one would expect a degree of overlap between the goals of some agents. One distinction between interface

and information agents, however, is that information agents are defined by what they do, in contrast to interface agents which are defined by what they are.

Information agents are most useful on the Web where they can help us with mundane tasks. For example, we carry out actions that may consume long time (e.g. searching the Web for information). Why not the computer (e.g. an information agent) does carries out such tasks for us and later on presents us with the results?

Reactive Agents

Reactive Agents act and respond in a stimulus-response [18] manner to the present state of the environment in which they are embedded.

The following three key ideas which underpin reactive agents:

- Emergent functionality: the dynamics of the interaction leads to the emergent complexity.
- Task decomposition: a reactive agent is viewed as a collection of modules which operate autonomously and responsible for specific tasks (e.g. sensing, computation, etc.).
- They tend to operate on representations that are close to raw sensor data

Hybrid Agents

Hybrid Agents refer to those agents whose constitution is a combination of two or more agent philosophies within a singular agent. These philosophies may be mobile, interface, information, collaborative, ... etc.

The goal of having hybrid agents is the notion that the benefits accrued from having the combination of philosophies within a single agent is greater than the gains obtained from the same agent based on a singular philosophy. An example of this is collaborative interface agents.

Mobile Agents

A software agent is a mobile software agent if it is able to migrate from host to host to work in a heterogeneous network environment. This means we must also consider the software environment in which mobile agents exist. This is called the mobile agent environment, which is a software system distributed over a network of heterogeneous computers.

Its primary task is to provide an environment in which mobile agents can run. Note that not only an agent transports itself, but also its state. When it reaches the new host, the agent should be able to perform appropriately in the new environment.

3.6 Naming

Q10. Write a short note on naming system.

Ans :

Names play a very important role in all computer systems. They are used to share resources, to uniquely identify entities, to refer to locations, and more. An important issue with naming is that a name can be resolved to the entity it refers to. Name resolution thus allows a process to access the named entity. To resolve names, it is necessary to implement a naming system. The difference between naming in distributed systems and non distributed systems lies in the way naming systems are implemented.

In a distributed system, the implementation of a naming system is itself often distributed across multiple machines. How this distribution is done plays a key role in the efficiency and scalability of the naming system. In this chapter, we concentrate on three different, important ways that names are used in distributed systems.

First, after discussing some general issues with respect to naming, we take a closer look at the organization and implementation of human-friendly names. Typical examples of such names include those for file systems and the World Wide Web. Building worldwide, scalable naming systems is a primary concern for these types of names.

Second, names are used to locate entities in a way that is independent of their current location. As it turns out, naming systems for human-friendly names are not particularly suited for supporting this type of tracking down entities. Most names do not

even hint at the entity's location. Alternative organizations are needed, such as those being used for mobile telephony where names are location independent identifiers, and those for distributed hash tables.

Finally, humans often prefer to describe entities by means of various characteristics, leading to a situation in which we need to resolve a description by means of attributes to an entity adhering to that description. This type of name resolution is notoriously difficult and we will pay separate attention to it.

3.6.1 Names, Identifiers and Addresses

Q11. Write about Names, Identifiers and Addresses.

Ans :

- A name in a distributed system is a string of bits or characters that is used to refer to an entity.
- An entity in a distributed system can be practically anything. Typical examples include resources such as hosts, printers, disks, and files.
- To operate on an entity, it is necessary to access it, for which we need an access point. An access point is yet another, but special, kind of entity in a distributed system. The name of an access point is called an address.
- An address is thus just a special kind of name: it refers to an access point of an entity. Because an access point is tightly associated with an entity.
- As a comparison, a telephone can be viewed as an access point of a person, whereas the telephone number corresponds to an address.
- In addition to addresses, there are other types of names that deserve special treatment, such as names that are used to uniquely identify an entity.
- A true identifier is a name that has the following properties
  1. An identifier refers to at most one entity.
  2. Each entity is referred to by at most one identifier.
  3. An identifier always refers to the same entity.
- By using identifiers, it is much easier to unambiguously refer to an entity.
- In many computer systems, addresses and identifiers are represented in machine-readable form only, that is, in the form of bit strings.

For example, an Ethernet address is essentially a random string of 48 bits.

3.6.2 Name Resolution

Q12. Explain the use of name resolution.

Ans :

Having names, identifiers, and addresses, how do we resolve names and identifiers to addresses?

It is important to realize that there is often a close relationship between name resolution in distributed systems and message routing. In principle, a naming system maintains a name-to-address binding which in its simplest form is just a table of (name, address) pairs.

However, in distributed systems that span large networks and for which many resources need to be named, a centralized table is not going to work.

Instead, what often happens is that a name is decomposed into several parts such as **Jtp.cs.vu.nl** and that name resolution takes place through a recursive look up of those parts.

For example, a client needing to know the address of the FTP server named by jtp.cs.vu.nl would first resolve nl to find the server responsible for names that end with nl, after which the rest of the name is passed to server NS(nl). This server may then resolve the name vu to the server NS(tvu.ni) responsible for names that end with vu.nl who can further handle the remaining name jtp.cs.

Eventually, this leads to routing the name resolution request as:

NS(.) ~ NS(nl) ~ NS(vu.nl) ~ address of **jtp.cs.vu.nl**

where NS(.) denotes the server that can return the address of NS(nl), also known as the root server. NS(vu.nl) will return the actual address of the FTP server.

3.6.2.1 The Implementation of a Name System

Q13. Explaina bout the Implementation of a Name System.

Ans :

In a distributed system, names are used to refer to a wide variety of resources such as computers, services, remote objects and files, as well as to users. Naming is an issue that is easily overlooked but is nonetheless fundamental in distributed system design. Names facilitate communication and resource sharing. A name is needed to request a computer system to act upon a specific resource chosen out of many; for example, a name in the form of a URL is needed to access a specific web page.

Processes cannot share particular resources managed by a computer system unless they can name them consistently. Users cannot communicate with one another via a distributed system unless they can name one another, for example, with email addresses.

Names are not the only useful means of identification: descriptive attributes are another. Sometimes clients do not know the name of the particular entity that they seek, but they do have some information that describes it. Or they may require a service and know some of its characteristics but not what entity implements it.

Name services and the Domain Name System

A name service stores information about a collection of textual names, in the form of bindings between the names and the attributes of the entities they denote, such as users, computers, services and objects. The collection is often subdivided into one or more naming contexts: individual subsets of the bindings that are managed as a unit. The major operation that a name service supports is to resolve a name – that is, to look up attributes from a given name.

The Domain Name System is a name service design whose main naming database is used across the Internet.DNS replaced the original Internet naming scheme, in which all host names and addresses were held in a single central master file and downloaded by FTP to all computers that required them.

This original scheme was soon seen to suffer from three major shortcomings:

- It did not scale to large numbers of computers.
- Local organizations wished to administer their own naming systems.
- A general name service was needed – not one that serves only for looking up computer addresses.

The objects named by the DNS are primarily computers – for which mainly IP addresses are stored as attributes – and what we have referred to in this chapter as naming domains are called simply domains in the DNS. In principle, however, any type of object can be named, and its architecture gives scope for a variety of implementations.

Organizations and departments within them can manage their own naming data. Millions of names are bound by the Internet DNS, and lookups are made against it from around the world. Any name can be resolved by any client. This is achieved by hierarchical partitioning of the name database, by replication of the naming data, and by caching.

Domain names

The DNS is designed for use in multiple implementations, each of which may have its own name space. In practice, however, only one is in widespread use, and that is the one used for naming across the Internet. The Internet DNS name space is partitioned both organizationally and according to geography. The names are written with the highest-level domain on the right. The original top-level organizational domains (also called generic domains) in use across the Internet were :

com – Commercial organizations
edu – Universities and other educational institutions
gov – US governmental agencies
mil – US military organizations
net – Major network support centres
org – Organizations not mentioned above
int – International organizations

New top-level domains such as biz and mobi have been added since the early 2000s. A full list of current generic domain names is available from the Internet Assigned Numbers Authority.

In addition, every country has its own domains:

us – United States

uk – United Kingdom

fr – France

... – ...

Countries, particularly those other than the US often use their own sub domains to distinguish their organizations. The UK, for example, has domains co.uk and ac.uk, which correspond to com and edu respectively (ac stands for 'academic community'). Note that, despite its geographic-sounding uk suffix, a domain such as doit.co.uk could have data referring to computers in the Spanish office of Doit Ltd., a national British company. In other words, even geographic-sounding domain names are conventional and are completely independent of their physical locations.

3.6.3 Simple Solutions

Q14. Explain broad casting and Multi-casting.

Ans :

There are two simple solutions for locating an entity. Both solutions are applicable only to local-area networks.

1. Broadcasting and Multicasting

Consider a distributed system built on a computer network: that offers efficient broadcasting facilities. Typically, such facilities are offered by local-area networks in which all machines are connected to a single cable. Also, local-area wireless networks fall into this category.

Locating an entity in such an environment is simple: 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. Only the machines that can offer an access point for the entity send a reply message containing the address of that access point.

This principle is 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 the given IP address. When the message arrives at a machine, the receiver checks whether it should listen to the requested IP address. If so, it sends a reply packet containing , for example, its Ethernet address.

Broadcasting becomes inefficient when the network grows. One possible solution is to switch to multicasting, by which only a restricted group of hosts receives the request. For example, Ethernet networks support data-link level multicasting directly in hardware.

Multicasting can also be used to locate entities in point-to-point networks. For example, the Internet supports network-level multicasting by allowing hosts to join a specific multicast group. Such 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.

Another way to use a multicast address is to associate it with a replicated entity, and to use multicasting to locate the nearest replica. When sending a request to the multicast address, each replica responds with its current (normal) IP address.

3.6.4 Locating Mobile Entities: Naming verses Locating Entities

Q15. Briefly discuss about locating mobile entities with forwarding pointer method.

Ans :

Forwarding Pointer Method

Another popular approach to locating mobile entities is to make use of forwarding pointers. The principle is simple: when an entity moves from A to B, it leaves behind in A a reference to its new location at B. The main advantage of this approach

is its simplicity: as soon as an entity has been located, for example by using a traditional naming service, a client can look up the current address by following the chain of forwarding pointers.

There are also a number of important drawbacks. **First**, if no special measures are taken, a chain for a highly mobile entity can become so long that locating that entity is prohibitively expensive. **Second**, all intermediate locations in a chain will have to maintain their part of the chain of forwarding pointers as long as needed.

A **third** (and related) drawback is the vulnerability to broken links. As soon as any forwarding pointer is lost (for whatever reason) the entity can no longer be reached. An important issue is, therefore, to keep chains relatively short, and to ensure that forwarding pointers are robust.

To better understand how forwarding pointers work, consider their use with respect to remote objects: objects that can be accessed by means of a remote procedure call. Following the approach in SSP chains, each forwarding pointer is implemented as a (client stub, server stub) pair as shown in Fig. 5-1. (We note that in Shapiro's original terminology, a server stub was called a scion, leading to (stub.scion} pairs , which explains its name.) A server stub contains either a local reference to the actual object or a local reference to a remote client stub for that object.

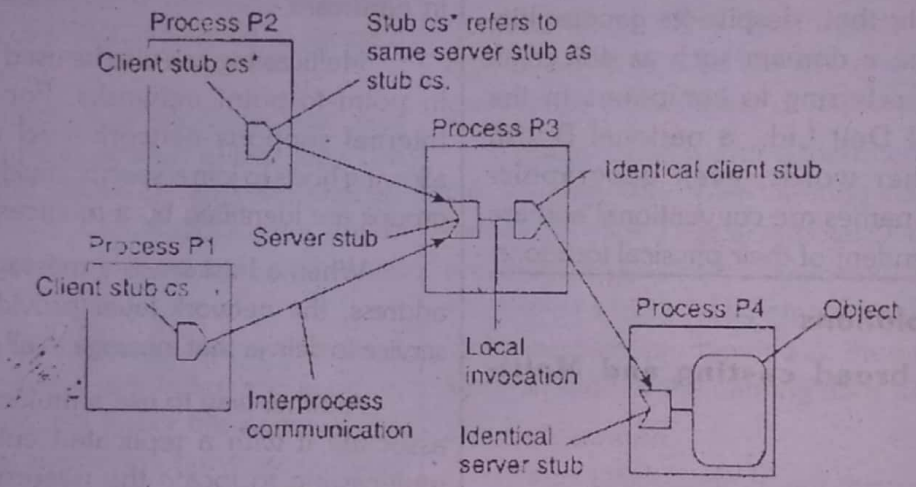


Fig. : The principle of forwarding pointers using (client stub, server stub)

Whenever 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. An interesting aspect of this approach is that migration is completely transparent to a client. The only thing the client sees of an object is a client stub. How, and to which location that client stub forwards its invocations, are hidden from the client. Also note that this use of forwarding pointers is not like looking up an address. Instead, a client's request is forwarded along the chain to the actual object.

To short-cut a chain of (client stub, server stub) pairs, an object invocation carries the identification of the client stub from where that invocation was initiated. A client-stub identification consists of the client's transport-level address, combined with a locally generated number to identify that stub. When the invocation reaches the object at its current location, a response is sent back to the client stub where the invocation was initiated (often without going back up the chain).

The current location is piggybacked with this response, and the client stub adjusts its companion server stub to the one in the object's current location. This principle is shown in Fig. 5-2.

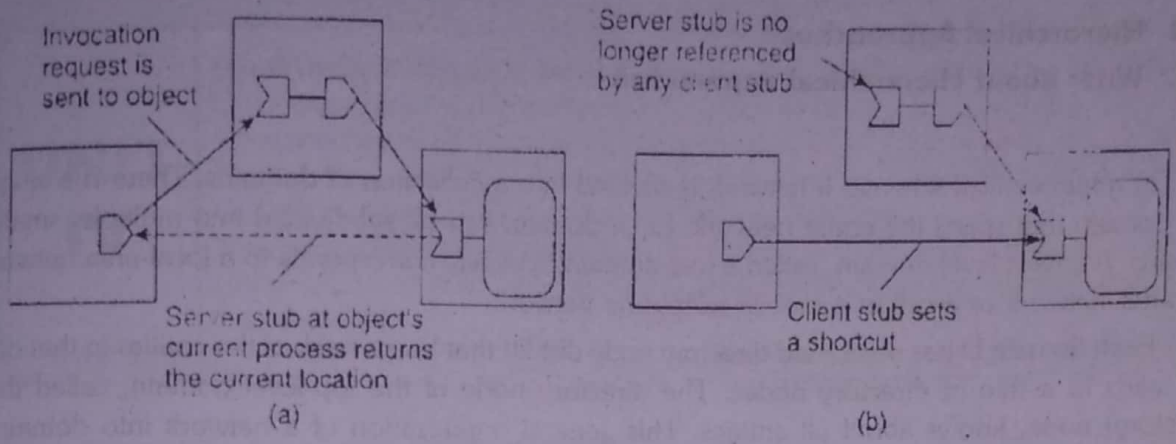


Fig. : Redirecting a forwarding pointer by storing a shortcut in a client stub

3.6.5 Home-Based Approaches

Q16. Write about Home-Based Approach.

Ans :

The use of broadcasting and forwarding pointers imposes scalability problems. A popular approach to supporting mobile entities in large-scale networks is to introduce a home location, which keeps track of the current location of an entity.

Special techniques may be applied to safeguard against network or process failures.

In practice, the home location is often chosen to be the place where an entity was created. The home-based approach is used as a fall-back mechanism for location services based on forwarding pointers.

Another example where the home-based approach is followed is in Mobile IP. Each mobile host uses a fixed IP address. All communication to that IP address is initially directed to the mobile host's home agent. This home agent is located on the LAN corresponding to the network address contained in the mobile host's IP address.

When the home agent receives a packet for the mobile host, it looks up the host's current location. If the host is on the current local network, the packet is simply forwarded. Otherwise, it is tunneled to the host's current location.

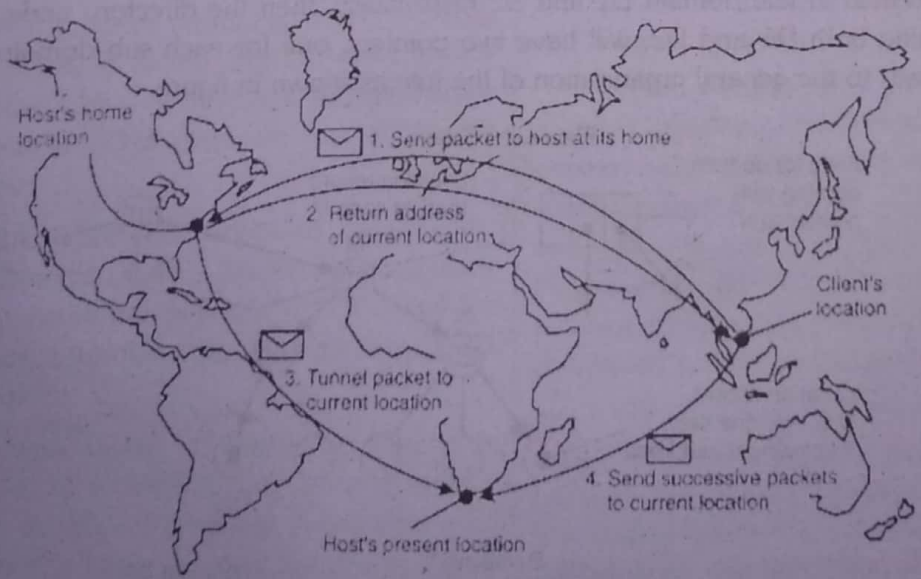


Fig. : The Principle of Mobile IP

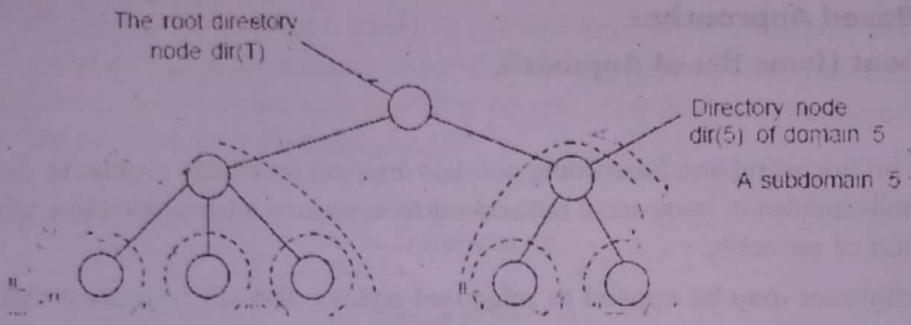
3.6.4 Hierarchical Approaches

Q17. Write about Hierarchical approaches.

Ans :

In a hierarchical scheme, a network is divided into a collection of domains. There is a single top-level domain that spans the entire network. Each domain can be subdivided into multiple, smaller sub domains. A lowest-level domain, called a leaf domain, typically corresponds to a local-area network in a computer network or a cell in a mobile telephone network.

Each domain D has associated directory node dirt D) that keeps track of the entities in that domain. This leads to a tree of directory nodes. The directory node of the top-level domain, called the root (directory) node, knows about all entities. This general organization of a network into domains and directory nodes is illustrated in Figure.



To keep track of the whereabouts of an entity, each entity currently located in a domain D is represented by a location record in the directory node dir(D).

A location record for entity E in the directory node N for a leaf domain D contains the entity's current address in that domain. In contrast, the directory node N' for the next higher-level domain D' that contains D, will have a location record for E containing only a pointer to N.

Likewise the parent node of N' will store a location record for E containing only a pointer to N'. Consequently, the root node will have a location record for each entity, where each location record stores a pointer to the directory node of the next lower-level sub domain where that record's associated entity is currently located. An entity may have multiple addresses, for example if it is replicated. If an entity has an address in leaf domain D1 and D2 respectively, then the directory node of the smallest domain containing both D1 and D2, will have two pointers, one for each sub domain containing an address. This leads to the general organization of the tree as shown in figure.

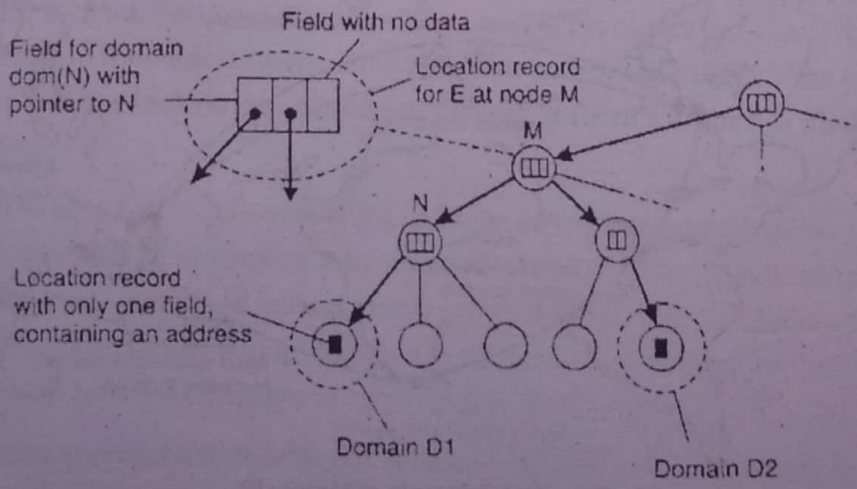


Fig. : An example of storing information of any entity having two addresses in different leaf domains.

UNIT IV

Distributed Object based Systems: CORBA: Overview of CORBA, Communication,Processes, Naming, Synchronization, Caching and Replication, Fault Tolerance, Security, Distributed COM: Overview of DCOM, Communication, Processes, Naming, Synchronization, Replication, Fault Tolerance, Security, GLOBE: Overview of GLOBE, Communication, Process, Naming, Synchronization, Replication, Fault Tolerance, Security, Comparison of COREA, DCOM, and Globe: Philosophy. Communication. Processes. Naming. Synchronization. Caching and Replication, Fault Tolerance. Security.

4.1 Distributed Object Systems

Q1. Explain the role of distributed object systems.

Ans :

Distributed objects form an important paradigm because it is relatively easy to hide distribution aspects behind an object's interface. Furthermore, because an object can be virtually anything, it is also a powerful paradigm for building systems.

Examples of Distributed Object Systems

- **Java/RMI** – Java/RMI relies on a protocol called the Java Remote Method Protocol – Portable across operating systems – Requires Java Virtual Machine (JVM) implementation – Uses TCP/IP for communication.
- **DCOM** – Supports remote objects by running on a protocol called the Object Remote Procedure Call – Is language independent – Requires a COM platform, i.e. Windows machine.
- **CORBA** – Uses a protocol called Internet Inter-ORB Protocol (IIOP) – Platform independent and language independent – Well-suited for complex heterogeneous systems (e.g. DoD systems).

Middleware based on distributed objects is designed to provide a programming model based on object-oriented principles and therefore to bring the benefits of the object oriented approach to distributed programming.

Such distributed objects as a natural evolution from three strands of activity :

- In distributed systems, earlier middleware was based on the client-server model and there was a desire for more sophisticated programming abstractions.
- In programming languages, earlier work in object-oriented languages.
- Simula-67 and Smalltalk led to the emergence of more mainstream and heavily used programming languages such as Java and C++ (languages used extensively in distributed systems).
- In software engineering, significant progress was made in the development of object-oriented design methods, leading to the emergence of the UnifiedModelling Language (UML) as an industrial-standard notation for specifying (potentially distributed) object-oriented software systems.

In other words, through adopting an object-oriented approach, distributed systems developers are not only provided with richer programming abstractions (using familiar programming languages such as C++ and Java) but are also able to use object-oriented design principles, tools and techniques (including UML) in the development of distributed systems software.

This represents a major step forward in an area where, previously, such design techniques were not available. It is interesting to note that the OMG, the organization that developed CORBA (see Section 8.3), also manages the standardization of UML.

Distributed object middleware offers a programming abstraction based on object oriented principles. Leading examples of distributed object middleware include Java RMI and CORBA .While Java RMI and CORBA share a lot in common, there is one important difference: the use of Java RMI is restricted to Java-based development, whereas CORBA is a multi-language solution allowing objects written in a variety of languages to interoperate. (Bindings exist for C++, Java, Python and several others.)

Objects	*Distributed objects*	*Description of distributed object*
Object references	Remote object references	Globally unique reference for a distributed object: may be passed as a parameter.
Interfaces	Remote interfaces	Provides an abstract specification of the methods that can be invoked on the remote object: specified using an interface definition language (IDL).
Actions	Distributed actions	Initiated by a method invocation, potentially resulting in invocation chains: remote invocations use RMI.
Exceptions	Distributed exceptions	Additional exceptions generated from the distributed nature of the system, including message loss or process failure.
Garbage collection	Distributed garbage collection	Extended scheme to ensure that an object will continue to exist if at least one object reference or remote object reference exists for that object, otherwise, it should be removed. Requires a distributed garbage collection algorithm.

Fig. : Distributed Systems

4.2 Introduction to CORBA

Q2. What is CORBA? Where does CORBA Fit in Distributed Systems ?

Ans :

CORBA is a software standard that is defined and maintained by the Object Management Group (OMG). The OMG was a non-profit organization founded in 1989 by eight companies. This consortium now compromises of over 800 members.

OMG products specifications, not implementations. Implementations of OMG specifications can be found on over 50 operating systems.

CORBA is the acronym for Common Object Request Broker Architecture. It consists of a standard framework for developing and maintaining distributed software systems.

CORBA is defined as an architecture and specification for creating, distributing and managing distributed program objects in a network.

In Common Object Request Broker Architecture, the programs that are written by different users at different locations are allowed to communicate with each other in a network.

SUN and W3Open have sanctioned CORBA as a standard architecture for distributed systems.

Where does CORBA Fit in Distributed Systems?

As we have studied in Unit 1 "A distributed architecture is an architecture that supports the development of applications and services that can exploit a physical architecture. The physical architecture consists of multiple, autonomous processing elements. These elements communicate with each other by sending messages over the network."

We had also discussed on the different issues related to distributed systems, the main issue of transparency and various transparency features of the distributed systems was also explained.

The different types of transparency that a distributed system should provide are

- Scalability transparency
- Migration transparency
- Access transparency
- Performance transparency
- Location transparency
- Failure transparency
- Concurrency transparency

We also have seen the concept of a "middleware". A middleware is software that enables interprocess communication. It also provides an API that isolates the application code from the underlying network. Middleware systems implement the various forms of distribution transparencies. These transparencies are maintained by creating the illusion of unity and homogeneity within the network i.e. that provide a "single-system image".

Altogether a distributed system consists of the following structure:

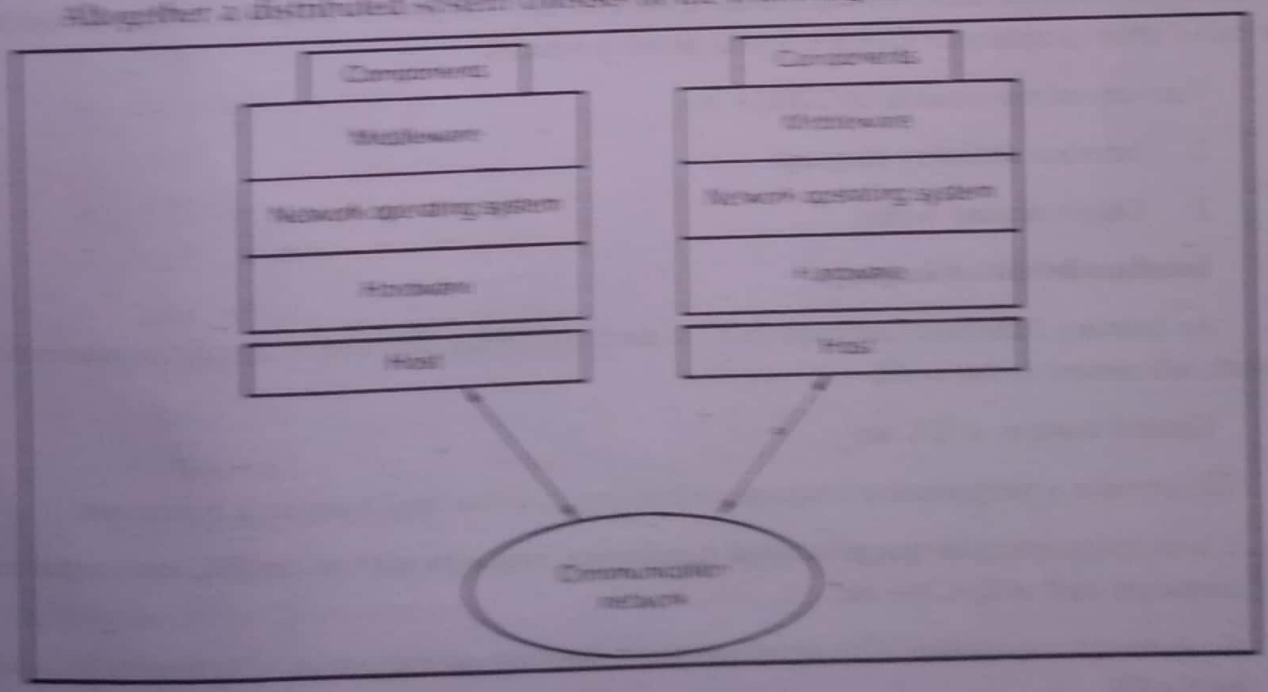


Fig. Distributed System

The MIDDLEWARE acts as glue between autonomous components and processes (e.g. clients, server). It does this by providing generic services on top of the OS.

There are three kinds of middleware systems:

- **Transaction-oriented middleware :** Transaction-oriented middleware supports distributed computing involving database applications.
- **Message-oriented middleware :** Message-oriented middleware supports reliable, asynchronous communications among distributed components
- **Object-oriented middleware :** Object-oriented middleware systems are based on object-oriented paradigm, and primarily supports synchronous communications among distributed components.

The most popular **object-oriented middleware include CORBA,** DCOM, Dot NET and EJB (which is based on RMI).

The main feature of CORBA are

i) **It uses RPC mechanisms** for the invocation of operations across different programming languages, hardware and operating system platforms. **Portability** and **interoperability** are the main features of RPC.

ii) **A component model,** the CORBA Component Model (CCM), for reusable component development.

4.3 Components Involved in CORBA : CORBA-IDL

Q3. What are the basic components of CORBA ?

Ans :

CORBA consists of wrapped objects. The wrapped objects consists of program code into a bundle containing information about the capabilities of the code and also how to call it. The wrapped objects can be called other programs or CORBA objects across a network.

The various components of CORBA are

1. Interface definition language.
2. Object request broker.

1. Interface definition language

An Interface Definition Language (IDL) is used by CORBA, this used to specify the interfaces that objects will present to the world.

General features of IDL are

i) IDL provides a programming language neutral way to define how a service is implements.

ii) It is an intermediary language between specification languages such as the UML and programming languages such as C, C++ etc.

iii) An abstract representation of the interfaces that a client will use and a serve will implement is provided by the IDL.

Clients and object implementation are isolated by three mechanisms.

- An IDL stub on the client end
- An ORB
- A skeleton on the object implementation end.

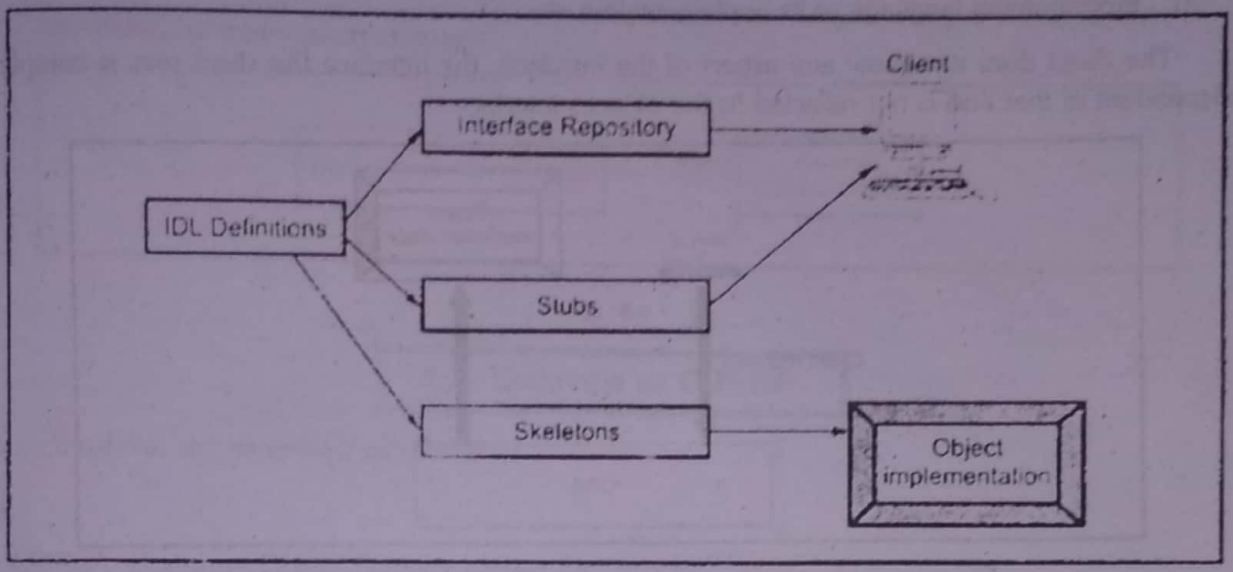


Fig. : Interface Definition Language

A "mapping" from IDL to a specific implementation language like C++ or Java is also provided by CORBA Standard mappings are available for

- Ada
- C
- C++
- Lisp
- Smalltalk
- Java
- COBOL
- PL/I
- And Python

Non-standard mappings are available for

- Perl
- Visual basic

1. The object request broker

A communication infrastructure called as Object Request Broker (ORB) is used for the object to communicate across the network.

The features of the object request broker are:

i) IDL interface isolates both client and object implementation from the ORB.

ii) Clients see only the object's interface, never the implementation.

iii) For communication, every request is passed to the client's local ORB.

The features of the interface like :

i) Location of the object

ii) Programming language of its implementation etc.

The client does not know any aspect of the interface, the interface the client sees is completely independent of that and is not reflected in the object's interface.

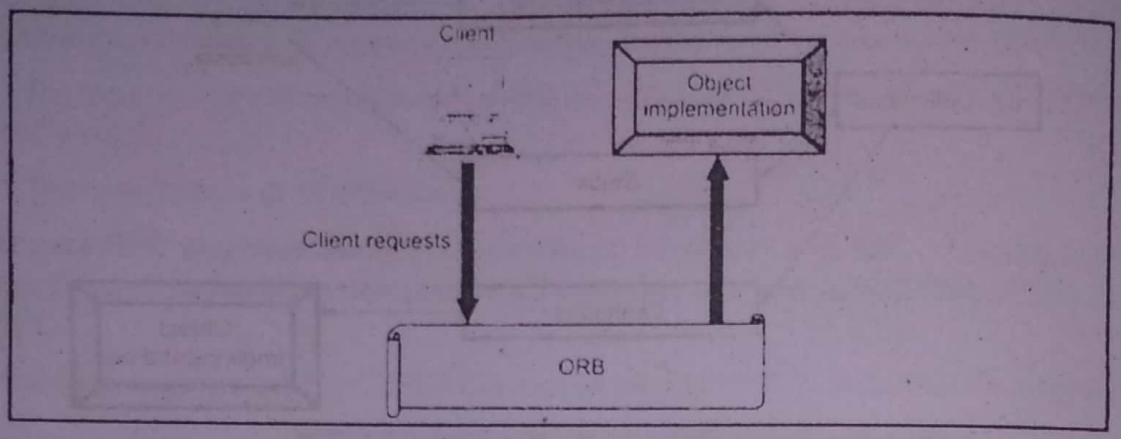


Fig.: Object request broker

The object request brokers communicated with GIOP **(General Inter-ORB Protocol)**.

The object management group defines three parts of GIOP :

1) The Common Data Representation (CDR)

2) The Interoperable Object Reference (IOR)

3) The defined message formats

We'll elaborate on each of the above

1) The Common Data Representation (CDR)

Whenever there is on the wire transfer between ORB's and inter – ORB bridges,r syntax mapping OMG IDL data types are transferred into a low level representation. This is done by the Common Data Representation CDR.

2) The Interoperable Object Reference (IOR)

The format of a reference to a remote object is defined by the Interoperable Object Reference (IOR).

The IOR consists of tagged profiles and their components. These components may carry various needed information regarding the tagged profiles. The typical IOR normally contains the following components.

- Protocol version ,Server address and A byte sequence that indentifies the remote object (object key).

3) The defined message formats

Agents exchange message for

- Facilitation object requests
- Locating object implementations

Managing communication channels.

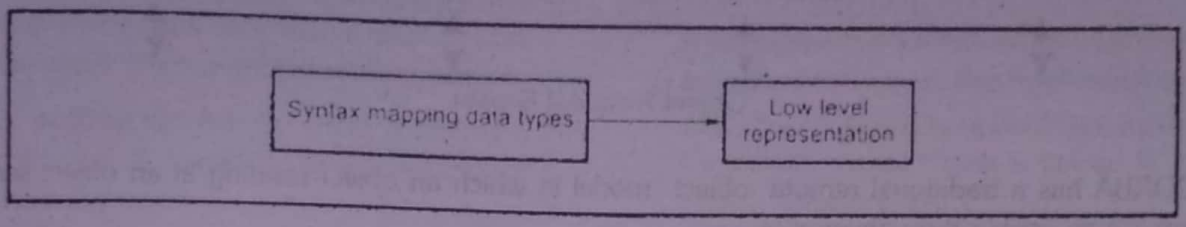


Fig.: 6A

4.3 Overview of CORBA

Q4. Explain, the overview of COBRA.

Ans :

- CORBA: Common Object Request Broker Architecture
- Background:
  - Developed by the Object Management Group (OMG) in response to industrial demands for object- based middleware
  - Currently in version #2.4 and #3
  - CORBA is a specification: different implementations of CORBA exist
  - Very much the work of a committee: there are over 800 members of the OMG and many of them have a say in what CORBA should look like
- CORBA provides a simple distributed -object model, with specifications for many supporting services it may be here to stay (for a couple of years)

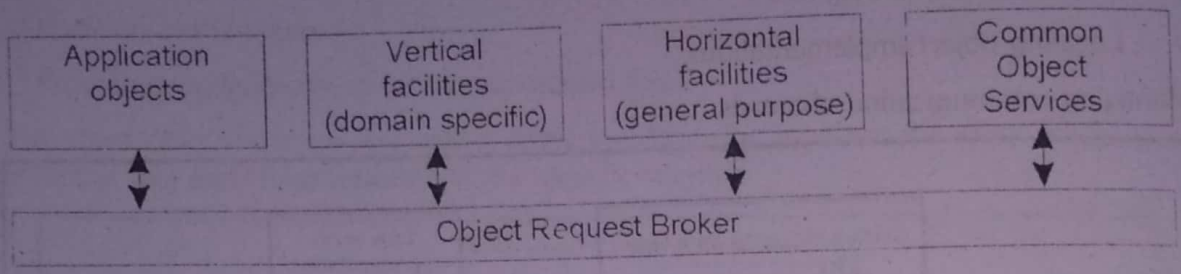
4.3.1 Architecture of CORBA

Q5. Explain about COBRA architecture.

Ans :

- The **Object Request Broker** (**ORB**) forms the core of any CORBA distributed system.
- **Horizontal facilities** consist of general-purpose high-level services that are independent of application domains.
  - User interface
  - Information management
  - System management
  - Task management

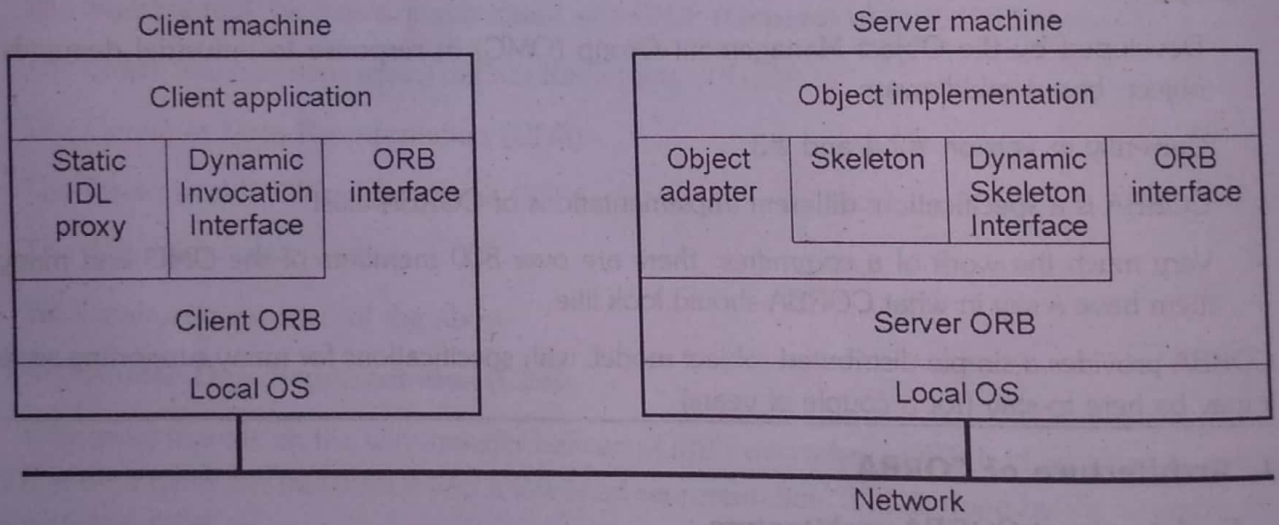
- [Object too big for pasting as inline graphic. | In-line.PNG *] **Vertical facilities** consist of high-level services that are targeted to a specific application domain such as electronic commerce, banking, manufacturing.



- CORBA has a traditional remote -object model in which an object residing at an object server is remote accessible through proxies
- All CORBA specifications are given by means of interface descriptions, expressed in an **Interface Definition Language (IDL)**.
- An interface is a collection of **methods**, and **objects** specify which **interfaces** they implement.
- It provides a precise syntax for expressing methods and their parameters.

[* Object too big for pasting as inline graphic. | In-line.PNG

Object Model



Objective Invocation Models :

Request type	Failure semantics	Description
Synchronous	At-most-once	Caller blocks until a response is returned or an exception is raised
One-way	Best effort delivery	Caller continues immediately without waiting for any response from the server
Deferred synchronous	At-most-once	Caller continues immediately and can later block until response is delivered

Communication Models

- CORBA supports the message-queuing model through the **messaging service**.
  - In **callback model**, A client provides an object with an interface containing callback methods which can be called by the underlying communication system to pass the result of an asynchronous invocation.
  - In **polling model**, the client is offered a collection of operations to poll its ORB for incoming result.
- **General Inter-ORB Protocol (GIOP)** is a standard communication protocol between the client and server.
- **Internet Inter-ORB Protocol** (IIOP) is a GIOP on top of TCP.

4.3.2 Processes

Q6. Explain the process in COBRA.

Ans :

- CORBA distinguishes two types of processes: clients and servers.
- An **interceptor** is a mechanism by which an invocation can be intercepted on its way from client to server, and adapted as necessary before letting it continue.
  - It is designed to allow proxies to adapt the client-side software.
  - **Request-level**: Allows you to modify invocation semantics (e.g., multicasting)
  - **Message-level**: Allows you to control message-passing between client and server (e.g., handle reliability and fragmentation)

4.3.3 Naming

Q7. Explain the naming in COBRA.

Ans :

- In CORBA, it is essential to distinguish specification- level and implementation- level object references.
  - **Specification level**: An object reference is considered to be the same as a proxy for the referenced object. è Having an object reference means you can directly invoke methods. There is no separate client- to -object binding phase
  - **Implementation level**: When a client gets an object reference, the implementation ensures that, one way or the other, a proxy for the referenced object is placed in the client's address space.

4.3.4 Synchronization

Q8. Explain synchronization in COBRA.

Ans :

- The two most important services that facilitate synchronization in CORBA are its **concurrency control service** and its **transaction service**.
- The two services collaborate to implement distributed and nested transactions using two-phase locking.
- There are two types of objects that can be part of transaction:
  - A **recoverable object** is an object that is executed by an object server capable of participating in a two-phase commit protocol.
  - The **transactional objects** are executed by servers that do not participate in a transaction's two-phase commit protocol.

4.3.5 Caching and Replication

Q9. Explain Caching & Replications in COBRA.

Ans :

- CORBA offers no support for generic caching and replication.
- CASCADE is built to provide a generic, scalable mechanism that allows any kind of CORBA object to be cached.

- CASCADE offers a caching service implemented as a large collection of object servers referred to as a **Domain Caching Server** (**DCS**).
- Each DCS is an object server running on a CORBA ORB. The collection of DCSs may be spread across the Internet.

4.3.6 Fault Tolerance

Q10. Explain the Fault Tolerance in COBRA.

Ans :

- In CORBA version 3, fault tolerance is addressed. The basic approach for fault tolerance is to replicate objects into **object groups**.
- Masking failures is achieved through replication by putting objects into object groups. Object groups are transparent to clients. They appear as normal objects.
- This approach requires a separate type of object reference: **Interoperable Object Group Reference** (**IOGR**).

4.3.7 Security

Q11. Explain the Security in COBRA.

Ans :

- The underlying idea is to allow the client and object to be mostly unaware of all the security policies, except perhaps at binding time. The ORB does the rest.
- Specific policies are passed to the ORB as (local) **policy objects** and are invoked when necessary.
  - Examples: Type of message protection, lists of trusted parties.
- A **replaceable** security service is a service which can be specified by means of standard interfaces that hide the implementation.

4.4 CORBA Services

Q12. What are the services offered by COBRA?

Ans :

The various services offered by CORBA are

- **Collection service :** As the name suggests it groups the object in either lists, queues, stacks etc. Access mechanisms are offered depending on the nature of the group.
- **Query service :** This service provides the collection of object which are queried using a declarative query language.
- **Concurrency control service :** Clients need certain advanced locking mechanisms to access shared object. This is done by providing concurrency control service.
- **Transaction service :** A series of method invocations across multiple objects in a single transaction is provided by the transaction service.
- **Notification service :** Facilities related to asynchronous communication are provided by the notification service.
- **Event service :** The clients and servers are interrupted on the occurrence of a certain event. This is the facility provided by an event server.
- **Life cycle service :** The objects can be created, destroyed, copied and move with the help of life cycle service.
- **Licensing service :** An object can attach a license and enforce a licensing policy under licensing service offered by CORBA.
- **Naming service :** Objects are being named for its reference in the system. The naming service provided this facility.
- **Property service :** The objects can be described because of the property service provided by CORBA.

- **Persistence service :** Information can be stored and retrieved from the disk because of the persistence service.
- **Relationship service :** Two or more objects can be related due to relationship services.
- **Security service :** Facilities for authorization, qualiting, authorization etc. are provided under security services.
- **Trading service :** Objects can advertise themselves under the provision of trading services offered by CORBA.

4.5 COM & DCOM

Q13. Write a note on DCOM and Object model.

Ans :

- Microsoft COM (Component Object Model) technology in the Microsoft Windows-family of Operating Systems enables software components to communicate.
- COM is used by developers to create re-usable software components, link components together to build applications, and take advantage of Windows services.
- The family of COM technologies includes COM+, Distributed COM (DCOM) and ActiveX® Controls.

4.5.1 DCOM: Distributed Component Object Model

Q14. What is DCOM ?

Ans :

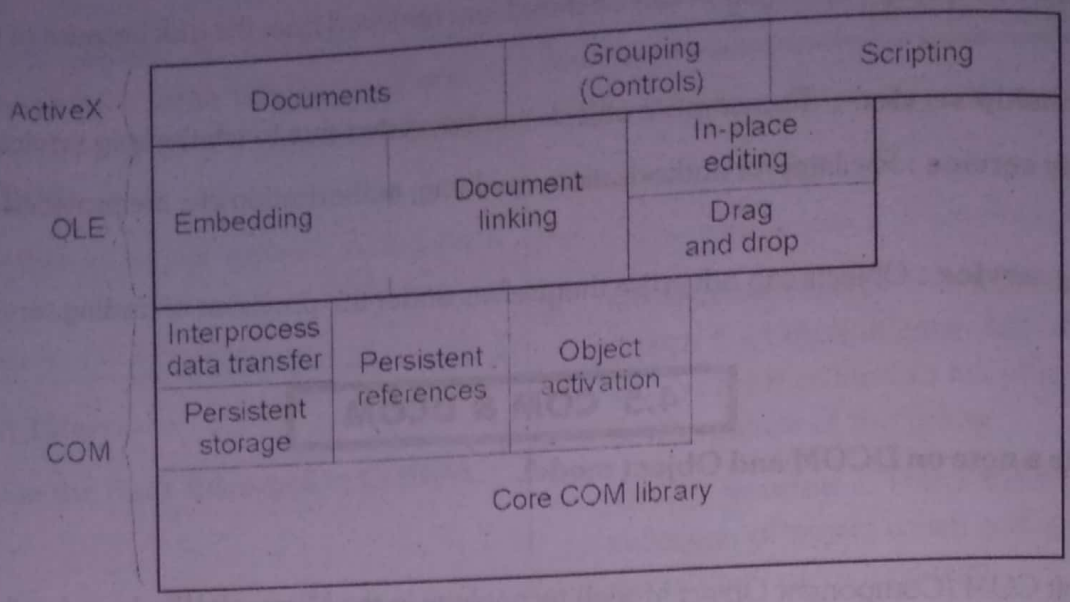
- Microsoft's solutions to establishing inter- process communication, possibly across machine boundaries.
- DCOM uses the RPC mechanism to transparently send and receive information between COM components (i.e., clients and servers) on the same network.
- Supports a primitive notion of distributed objects
- Evolved from early Windows versions to current NT- based systems (including Windows 2000/XP)
- Comparable to CORBA's object request broker (Microsoft's CORBA).

4.5.2 DCOM Overview

Q15. Discuss the Overview of DCOM ?

Ans :

- DCOM is related to many things that have been introduced by Microsoft in the past couple of years :
  - **DCOM** : Adds facilities to communicate across process and machine boundaries.
  - **SCM** : Service Control Manager, responsible for activating objects (cf., to CORBA's implementation repository).
- **Proxy marshaler** : handles the way that object references are passed between different machines.



Communication Models :

- Object invocations: Synchronous remote- method calls with at -most -once semantics. Asynchrono invocations are supported through a polling model, as in CORBA.
- Event communication: Similar to CORBA's push- style model:
- Messaging: Completely analogous to CORBA messaging.
- Observation: Objects are referenced by means of a local interface pointer. The question is how su pointers can be passed between different machines:

Naming:

- Observation: DCOM can handle only objects as temporary instances of a class. To accommoda objects that can outlive their client, something else is needed.
- Moniker: A name that uniquely identifies a Microsoft's COM (persistent) object similar to a directo path name.
  - A moniker associates data (e.g., a file), with an application or program.
  - Monikers can be stored.
  - A moniker can contain a binding protocol, specifying how the associated program should 'launched' with respect to the data.

Fault Tolerance

- Automatic transactions: Each class object (from which objects are created), has a transaction attrib that determines how its objects behave as part of a transaction.
- Note: Transactions are essentially executed at the level of a method invocation.

Security

- Declarative security: Register per object what the system should enforce with respect to authenticati Authentication is associated with users and user groups. There are different authentication levels.
- Delegation: A server can impersonate a client depending on a level.
- **Note:** There is also support for programmatic security by which security levels can be set by application, as well as the required security services (see book).

4.6 GLOBE

Q16. Explain about Globe object model.

Ans :

GLOBE is a completely different type of object-based distributed system. Globe is a system in which scalability plays a central role. All aspects that deal with constructing a large-scale wide-area system that can support huge numbers of users and objects drive the design of Globe. Fundamentally, this approach is the way objects are viewed. Like other object-based systems. , Objects in Globe are expected to encapsulate state and operations on that state.

Important difference with other object-based systems is that objects are also expected to encapsulate the implementation of policies that prescribe the distribution of an object's state across multiple machines. In other words, each object determines how its state will be distributed over its replicas. Each object also controls its own policies in other areas as well.

By and large, objects in Globe are put in charge as much as possible. For example, an object decides how, when, and where its state should be migrated. Also, an object decides if its state is to be replicated, and if so, how replication should take place. In addition, an object may also determine its security policy and implementation. Below, we describe how such encapsulation is achieved.

Object Model

Unlike most other object-based distributed systems, Globe does not adopt the remote-object model Instead, objects in Globe can be physically distributed, meaning that the state of an object can be distributed and replicated across multiple processes. This organization is shown in Fig. 10-3, which shows an object that is distributed across four processes, each running on a different machine. Objects in Globe are referred to as distributed shared objects, to reflect that objects are normally shared between several processes. The object model originates from the distributed objects used in Orca. Similar approaches have been followed for fragmented objects.

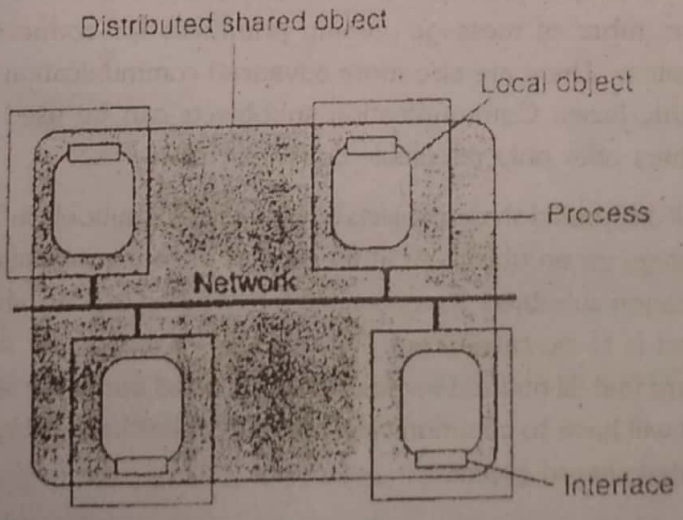


Fig. : The Organisation of a Globe distributed shared object

A process that is bound to a distributed shared object is offered a local implementation of the interfaces provided by that object. Such a local implementation is called a local representative, or simply local object. In principle, whether or not a local object has state is completely transparent to the bound

process. All implementation details of an object are hidden behind the interfaces offered to a process. The only thing visible outside the local object are its methods.

Globe local objects come in two flavors. A primitive local object is a local object that does not contain any other local objects. In contrast, a composite local object is an object that is composed of multiple (possibly composite) local objects. Composition is used to construct a local object that is needed for implementing distributed shared objects. This local object is shown in Fig. 10-4 and consists of at least four sub objects.

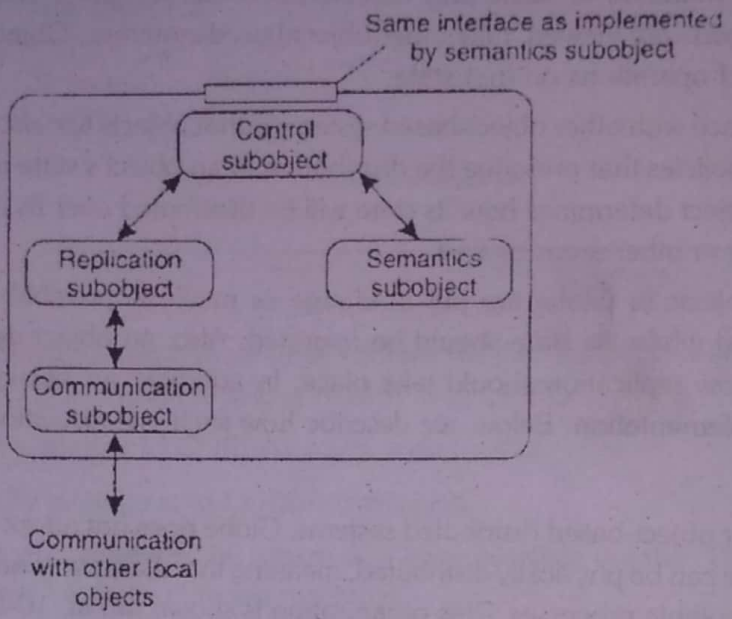


Fig. : The general organisation of a local object for distributed objects in Globe

The semantics sub object implements the functionality provided by a distributed shared object. In essence, it corresponds to ordinary remote objects, similar in flavor to EIBs.

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. There are also more advanced communication subobjects available that implement multicasting interfaces. Communication subobjects can be used that implement reliable communication, while others offer only unreliable communication.

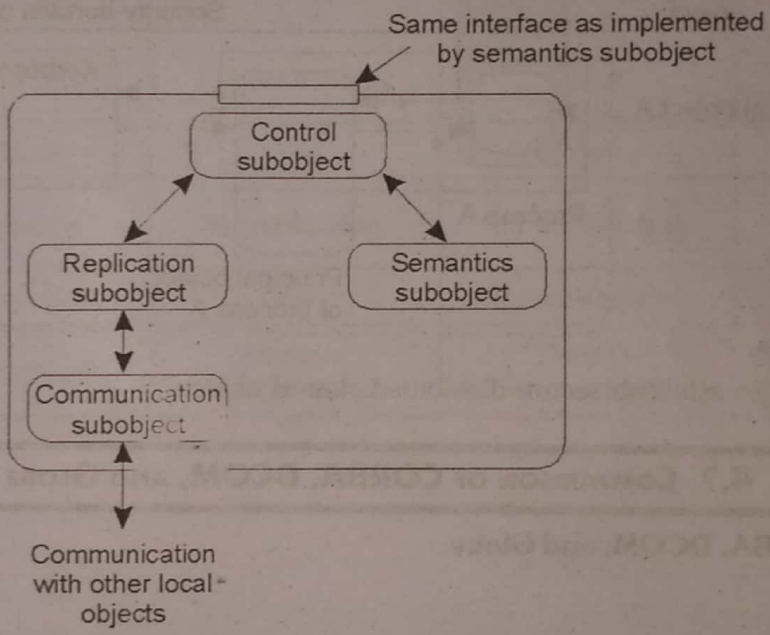
Crucial to virtually all distributed shared objects is the replication subobject. This subobject implements the actual distribution strategy for an object. As in the case of the communication subobject, its interface is standardized. The replication subobject is responsible for deciding exactly when a method as provided by the semantics subobject is to be carried out. For example, a replication subobject that implements active replication will ensure that all method invocations are carried out in the same order at each replica. In this case, the subobject will have to communicate with the replication subobjects in other local objects that comprise the distributed shared object.

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. In addition, it is responsible for exporting the interfaces of the semantics subobject to the process bound to the distributed shared object. All method invocations requested by that process are marshaled by the control subobject and passed to the replication subobject.

The replication subobject will eventually allow the control subobject to carry on with an invocation request and to return the results to the process. Likewise, invocation requests from remote processes are eventually passed to the control subobject as well. Such a request is then unmarshaled, after which the invocation is carried out by the control subobject, passing results back to the replication subobject.

Types Globe Objects

- **Semantics sub object:** Contains the methods that implement the functionality of the distributed shared object
- **Communication sub object:** Provides a (relatively simple), network- independent interface for communication between local objects
- **Replication sub object:** Contains the implementation of an object- specific consistency protocol that controls exactly when a method on the semantics sub object may be invoked
- **Control sub object:** Connects the user -defined interfaces of the semantics sub object to the generic, predefined interfaces of the replication sub object.



Naming

- **Observation:** Objects in Globe have their own object- specific implementations; there is no "standard" proxy that is implemented for all clients
- **Observation:** Globe separates naming from locating objects. The current naming service is based on DNS, using TXT records for storing object handles
- **Observation:** The location service is implemented as a generic, hierarchical tree.

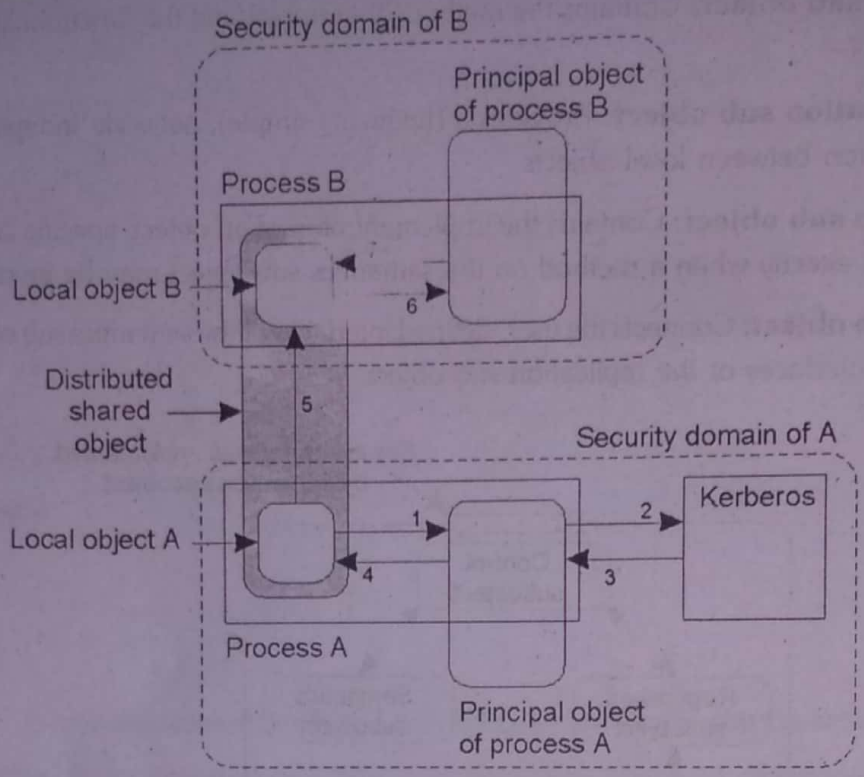
Caching and Replication

- **Observation:** Here's where Globe differs from many other systems:
  - The organization of a local object is such that replication is inherently part of each distributed shared object.

- All replication sub objects have the same interface:
- This approach allows to implement any object -specific caching/replication strategy

Security

- **Essence:** Additional security sub object checks for authorized communication, invocation, and parameter values. Globe can be integrated with existing security services.



Using Kerberos to establish secure distributed shared objects.

4.7 Comparison of CORBA, DCOM, and Globe

Q17. Compare CORBA, DCOM, and Globe.

Ans :

Issue	CORBA	DCOM	Globe
Design goals	Interoperability	Functionality	Scalability
Object model	Remote objects	Remote objects	Distributed objects
Services	Many of its own	From environment	Few
Interfaces	IDL based	Binary	Binary
Sync. communication	Yes	Yes	Yes
Async. communication	Yes	Yes	No
Callbacks	Yes	Yes	No

Events	Yes	Yes	No
Messaging	Yes	Yes	No
Object server	Flexible (POA)	Hard-coded	Object dependent
Directory service	Yes	Yes	No
Trading service	Yes	No	No
Naming service	CORBA	DCOM	Globe
Naming service	Yes	Yes	Yes
Location service	No	No	Yes
Object reference	Object's location	Interface pointer	True identifier
Synchronization	Transactions	Transactions	Only intra-object
Replication support	Separate server	None	Separate subobject
Transactions	Yes	Yes	No
Fault tolerance	By replication	By transactions	By replication
Recovery support	Yes	By transactions	No
Security	Various mechanisms	Various mechanisms	More work needed

UNIT V

Distributed Multimedia Systems : Introduction. Characteristics of Multimedia Data. Quality of Service Management: Quality of Service negotiation. Admission Control. Resource Management: Resource Scheduling.

5.1 Distributed Multimedia Systems

Q1. Write a short note on distributed multimedia systems.

Ans :

Modern computers can handle streams of continuous, time-based data such as digital audio and video. This capability has led to the development of distributed multimedia applications such as networked video libraries, Internet telephony and video conferencing. Such applications are viable with current general-purpose networks and systems, although the quality of the resulting audio and video is often less than satisfactory. More demanding applications such as large-scale video conferencing, digital TV production, interactive TV and video surveillance systems are beyond the capabilities of current networking and distributed system technologies.

Multimedia applications demand the timely delivery of streams of multimedia data to end users. Audio and video streams are generated and consumed in real time, and the timely delivery of the individual elements (audio samples, video frames) is essential to the integrity of the application. In short, multimedia systems are real-time systems:they must perform tasks and deliver results according to a schedule that is externally determined. The degree to which this is achieved by the underlying system is known as the quality of service (QoS) enjoyed by an application.

Although the problems of real-time system design had been studied before the advent of multimedia systems, and many successful real-time systems were developed, they have not generally been integrated into more general-purpose operating systems and networks. The nature of the tasks performed by these existing real-time systems, such as avionics, air traffic control, manufacturing process control and telephone switching, differs from those performed in multimedia applications.

The former generally deal with relatively small quantities of data and have relatively infrequent hard deadlines, but failure to meet any deadline can have serious or even disastrous consequences. In such cases, the solution adopted has been to over-specify the computing resources and to allocate them on a fixed schedule that ensures that worst-case requirements are always met. This type of solution is not available for most Internet multimedia streaming applications on desktop computers, resulting in a 'best-effort' quality of service using the available resources.

The planned allocation and scheduling of resources to meet the needs of multimedia and other applications is referred to as quality of service management. Most current operating systems and networks do not include the QoS management facilities needed for a guaranteed quality of service for multimedia applications.

The consequences of failure to meet deadlines in multimedia applications can be serious, especially in commercial environments such as video-on-demand services, business conferencing

applications and remote medicine, but the requirements differ significantly from those of other real-time applications :

- Multimedia applications are often highly distributed and operate within general purpose distributed computing environments. They therefore compete with other distributed applications for network bandwidth and for computing resources at users' workstations and servers.
- The resource requirements of multimedia applications are dynamic. A video conference will require more or less network bandwidth as the number of participants grows or shrinks. Its use of computing resources at each user's
- workstation will also vary, since, for example, the number of video streams that have to be displayed varies. Multimedia applications may involve other variable or intermittent loads as well. For example, a networked multimedia lecture might include a processor-intensive simulation activity.
- Users often wish to balance the resource costs of a multimedia application with other activities. Thus they may be willing to reduce their demands for video bandwidth in a conferencing application in order to allow a separate voice conversation to proceed, or they may wish a program development or word processing activity to proceed while they are participating in a conference.

QoS management systems are intended to meet all of these needs, managing the available resources dynamically and varying the allocations in response to changing demands and user priorities. A QoS management system must manage all of the computing and communication resources needed to acquire, process and transmit multimedia data streams, especially where the resources are shared between applications.

Figure 20.1 illustrates a typical distributed multimedia system capable of supporting a variety of applications, such as desktop conferencing or providing access to stored video sequences, broadcast digital TV and radio. The resources for which QoS management is required include network bandwidth, processor cycles and memory capacity.

Disk bandwidth at the video server may also be included. We shall adopt the generic term resource bandwidth to refer to the capacity of any hardware resource (network, central processor, disk subsystem) to transmit or process multimedia data.

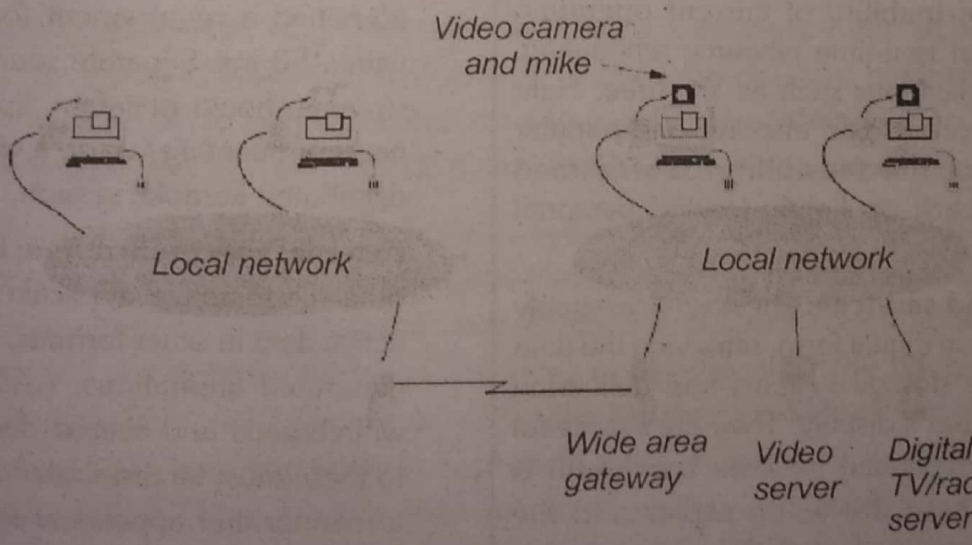


Fig. : A distributed multimedia system

5.2 Multimedia Applications

Q2. Write a short note on Multimedia applications.

Ans :

- In an open distributed system, multimedia applications can be started up and used without prior arrangement.
- Several applications may coexist in the same network and even on the same workstation.
- The need for QoS management therefore arises regardless of the total quantity of resource bandwidth or memory capacity in the system.
- QoS management is needed in order to guarantee that applications will be able to obtain the necessary quantity of resources at the required times, even when other applications are competing for the resources.
- Some multimedia applications have been deployed even in today's QoS-less, best efforts computing and network environments. These include :

Web-based multimedia: These are applications that provide best-efforts quality of service for access to streams of audio and video data published via the Web. They have been successful when there is little or no need for the synchronization of the data streams at different locations. Their performance varies with the bandwidth and latencies in networks and is hampered by the inability of current operating systems to support real-time resource scheduling. Nevertheless, applications such as YouTube, Hulu and BBC iPlayer provide an effective and popular demonstration of the feasibility of streamed multimedia playback on lightly loaded personal computers.

Video-on-demand services: These services supply video information in digital form, retrieving the data from large online storage systems and delivering them to the end user's display. They are successful where sufficient dedicated network bandwidth is available and where the video server and the receiving stations are dedicated. They also employ considerable buffering at the destination.

Many multimedia applications are cooperative (involving several users) and synchronous (requiring the users' activities to be closely coordinated). They span a wide spectrum of application contexts and scenarios. For example:

- Internet telephony. See the box on the next page.
- A simple video conference involving two or more users, each using a workstation equipped with a digital video camera, microphone, sound output and video display capability.(for example: Skype, Net Meeting [www.microsoft.com III], iChat AV [www.apple.com II]).
- A music rehearsal and performance facility enabling musicians at different locations to perform in an ensemble.

Applications such as these require :

- **Low-latency communication:** Round-trip delays should not exceed 100–300 ms, so that interaction between users appears to be synchronous.
- **Synchronous distributed state:** If one user stops a video on a given frame, the other users should see it stopped at the same frame.
- **Media synchronization:** All participants in a music performance should hear the performance at approximately the same time identified a requirement for synchronization within 50 ms. Separate soundtrack and video streams should maintain 'lip sync', e.g., for a user commenting live on a video playback or a distributed karaoke session.
- **External synchronization:** In conferencing and other cooperative applications, there may be active data in other formats, such as computer-generated animations, CAD data, electronic whiteboards and shared documents. Updates to these must be distributed and acted upon in a manner that appears at least approximately synchronized with the time-based multimedia streams.

5.3 CHARACTERISTIC OF MULTIMEDIA DATA

Q3. What are the different characteristics of Multimedia data ?

Ans :

We have referred to video and audio data as continuous and time-based. How can we define these characteristics more precisely? The term 'continuous' refers to the user's view of the data. Internally, continuous media are represented as sequences of discrete values that replace each other over time. For example, the value of an image array is replaced 25 times per second to give the impression of a TV-quality view of a moving scene; a sound amplitude value is replaced 8000 times per second to convey telephone quality speech.

Multimedia streams are said to be time-based (or isochronous) because timed data elements in audio and video streams define the semantics or 'content' of the stream. The times at which the values are played or recorded affect the validity of the data. Hence systems that support multimedia applications need to preserve the timing when they handle continuous data.

Multimedia streams are often bulky. Hence systems that support multimedia applications need to move data with greater throughput than conventional systems. Figure 20.2 shows some typical data rates and frame/sample frequencies. We note that the resource bandwidth requirements for some are very large. This is especially so for video of reasonable quality. For example, an uncompressed standard TV video stream requires more than 120 Mbps, which exceeds the capacity of a 100-Mbps Ethernet network.

A program that copies or applies a simple data transformation to each frame of a standard TV video stream requires less than 10% of the CPU capacity of a PC. The figures for high-definition television streams are higher, and we should note that in many applications, such as video conferencing, there is a need to handle multiple video and audio streams concurrently. The use of compressed representations to overcome these problems is therefore essential, although transformations such as video mixing and editing are difficult to accomplish with compressed streams.

Compression can reduce bandwidth requirements by factors of between 10 and 100, but the timing requirements of continuous data are unaffected. There is intensive research and standardization activity aimed at producing efficient, general-purpose representations and compression methods for multimedia data streams. This work has resulted in various compressed data formats, such as GIF, TIFF and JPEG for still images and MPEG-1, MPEG-2 and MPEG-4 for video sequences.

Although the use of compressed video and audio data reduces bandwidth requirements in communication networks, it imposes substantial additional loads on processing resources at the source and destination. This need has often been met through the use of special-purpose hardware to process and dispatch video and audio information, i.e., the video and audio coders/decoders (codecs) found on video cards manufactured for personal computers. But the increasing power of personal computers and multiprocessor architectures now enable them to perform much of this work in software using software coding and decoding filters. This approach offers greater flexibility, with better support for application-specific data formats, special-purpose application logic and the simultaneous handling of multiple media streams.

	Data rate (approximate)	Sample or frame size	frequency
Telephone speech	64 kbps	8 bits	8000 sec
CD-quality sound	1.4 Mbps	16 bits	44,000 sec
Standard TV video (uncompressed)	120 Mbps	up to 640 × 480 pixels × 16 bits	24 sec
Standard TV video (MPEG-1 compression)	1.5 Mbps	variable	24 sec
HDTV video (uncompressed)	1000 3000 Mbps	up to 1920 × 1080 pixels × 24 bits	24 60 sec
HDTV video (MPEG-2 MPEG-4 compression)	6 20 Mbps	variable	24 60 sec

Fig. : Characteristics of typical multimedia streams

5.4 Quality of Service Management

Q4. Write about quality of service management.

Ans :

When multimedia applications run in networks of personal computers, they compete for resources at the workstations running the applications (processor cycles, bus cycles, buffer capacity) and in the networks (physical transmission links, switches, gateways).

Workstations and networks may have to support several multimedia and conventional applications. There is competition between the multimedia and conventional applications, between different multimedia applications and even between the media streams within individual applications.

The concurrent use of physical resources for a variety of tasks has long been possible with multitasking operating systems and shared networks. In multitasking operating systems, the central processor is allocated to individual tasks (or processes) in a round-robin or other scheduling scheme that shares the processing resources on a best effort basis among all of the tasks currently competing for the central processor.

Networks are designed to enable messages from different sources to be interleaved, allowing many virtual communication channels to exist on the same physical channels. The predominant local area network technology, Ethernet, manages a shared transmission medium in a best-effort manner. Any node may use the medium when it is quiet, but packet collisions can occur, and when they do sending nodes wait for random back off periods in order to prevent repeated collisions. Collisions are likely to occur when the network is heavily loaded, and this scheme cannot provide any guarantees regarding the bandwidth or latency in such situations.

The key feature of these resources allocation schemes is that they handle increases in demand by spreading the available resources more thinly between the competing tasks. Round-robin and other best-efforts methods for sharing processor cycles and network bandwidth cannot meet the needs of multimedia applications. As we have seen, the timely processing and transmission of multimedia streams is crucial for them. Late delivery is valueless. In order to achieve timely delivery, applications need guarantees that the necessary resources will be allocated and scheduled at the required times.

The management and allocation of resources to provide such guarantees is referred to as quality of service management. Figure shows the infrastructure components for a simple multimedia conferencing application running on two personal computers, using software data compression and format conversion. The white boxes represent software components whose resource requirements may affect the quality of service of the application.

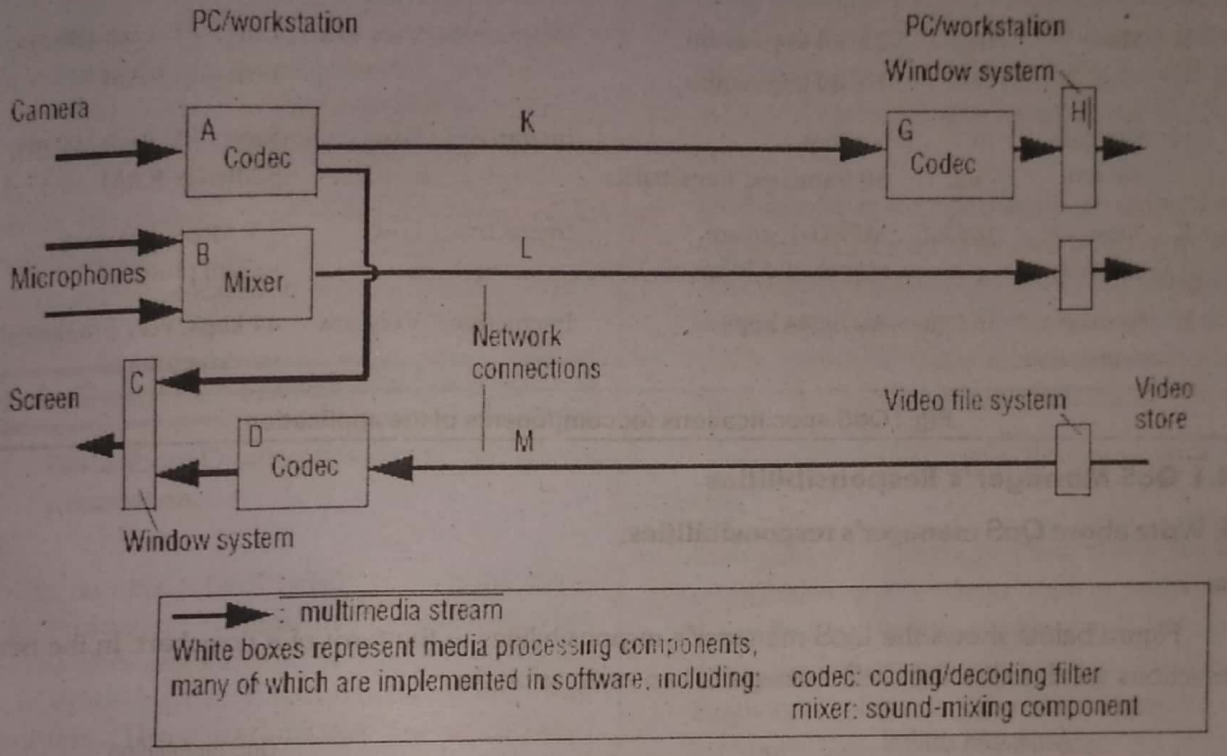


Fig. : Typical infrastructre components for multimedia application

The figure shows the most commonly used abstract architecture for multimedia software, in which continuously flowing streams of media data elements (video frames, audio samples) are processed by a collection of processes and transferred between the processes by interprocess connections. The processes produce, transform and consume continuous streams of multimedia data.

The connections link the processes in a sequence from a source of media elements to a target, at which it is rendered or consumed. The connections between the processes may be implemented by networked connections or by in-memory transfers when processes reside on the same machine. For the elements of multimedia data to arrive at their target on time, each process must be allocated adequate CPU time, memory capacity and network bandwidth to perform its designated task and must be scheduled to use the resources sufficiently frequently to enable it to deliver the data elements in its stream to the next process on time.

In figure below we have resource requirements for the main software components and network connections. Clearly, the required resources can be guaranteed only if there is a system component responsible for the allocation and scheduling of those resources. We shall refer to that component as the quality of service manager.

	Component		Bandwidth	Latency	Loss rate	Resources required
	Camera	Out:	10 frames/sec, raw video 640x480x16 bits	–	Zero	–
A	Codec	In:	10 frames/sec, raw video	Interactive	Low	10 ms CPU each 100 ms;
		Out:	MPEG-1 stream			10 Mbytes RAM
B	Mixer	In:	2 × 44 kbps audio	Interactive	Very low	1 ms CPU each 100 ms;
		Out:	1 × 44 kbps audio			1 Mbytes RAM
H	Window system	In:	Various	Interactive	Low	5 ms CPU each 100 ms;
		Out:	50 frame/sec framebuffer			5 Mbytes RAM
K	Network connection	In/Out:	MPEG-1 stream, approx. 1.5 Mbps	Interactive	Low	1.5 Mbps, low-loss stream protocol
L	Network connection	In/Out:	Audio 44 kbps	Interactive	Very low	44 kbps, very low-loss stream protocol

Fig. : QoS specifications for components of the application

5.4.1 QoS Manager's Responsibilities

Q5. Write about QoS manager's responsibilities.

Ans :

Figure below shows the QoS manager's responsibilities in the form of a flowchart. In the next two subsections we describe the QoS manager's two main subtasks :

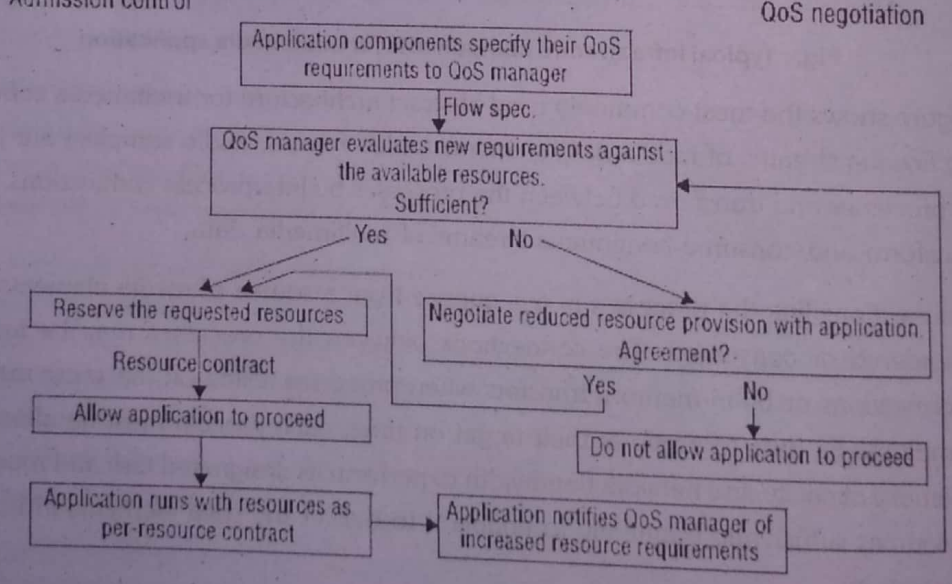


Fig. : The OoS manager's task

Quality of Service Negotiation

The application indicates its resource requirements to the QoS manager. The QoS manager evaluates the feasibility of meeting the requirements against a database of the available resources and current

resource commitments and gives a positive or negative response. If it is negative, the application may be reconfigured to use reduced resources, and the process is repeated.

Admission control

If the result of the resource evaluation is positive, the requested resources are reserved and the application is given a resource contract, stating the resources that have been reserved. The contract includes a time limit. The application is then free to run. If it changes its resource requirements it must notify the QoS manager. If the requirements decrease, the resources released are returned to the database as available resources. If they increase, a new round of negotiation and admission control is initiated.

5.5 Quality of Service Negotiation

Q6. Write about Quality of Service Negotiation parameters.

Ans :

To negotiate QoS between an application and its underlying system, the application must specify its QoS requirements to the QoS manager. This is done by the transmission of a set of parameters. Three parameters are of primary interest when it comes to processing and transporting multimedia streams:

Bandwidth: The bandwidth of a multimedia stream or component is the rate at which data flows through it.

Latency: Latency is the time required for an individual data element to move through a stream from the source to the destination. This may vary depending on the volume of other data in the system and other characteristics of the system load. This variation is termed jitter – formally, jitter is the first derivative of the latency.

Loss rate: Since the late delivery of multimedia data is of no value, data elements will be dropped when it is impossible to deliver them before their scheduled delivery time. In a perfectly managed QoS environment, this should never happen, but as yet few such environments exist, for reasons outlined earlier.

The three parameters can be used :

1. To describe the characteristics of a multimedia stream in a particular environment. For example, a video stream may require an average bandwidth of 1.5 Mbps, and because it is used in a conferencing application it needs to be transferred with at most 150 ms delay to avoid conversation gaps. The decompression algorithm used at the target may still yield acceptable pictures with a loss rate of one frame out of 100.
2. To describe the capabilities of resources to transport a stream. For example, a network may provide connections of 64 kbps bandwidth, its queuing algorithms may guarantee delays of less than 10 ms and the transmission system may guarantee a loss rate smaller than 1 in 106.

The parameters are interdependent. For example :

- Loss rate in modern systems rarely depends on actual bit errors due to noise or malfunction; it results from buffer overflow and from time-dependent data arriving too late. Hence, the larger bandwidth and the lower delay can be, the more likely is a low loss rate.
- The smaller the overall bandwidth of a resource is compared with its load, the more messages will accumulate in front of it and the larger the buffers for this accumulation will need to be to avoid loss. The larger the buffers become, the more likely it is that messages will need to wait for other messages in front of them to be serviced – that is, the larger the delay will become.

5.5.1 Specifying the QoS Parameters for Streams

Q7. Write about Specifying the QoS parameters for streams.

Ans :

The values of QoS parameters can be stated explicitly (e.g., for the camera output stream in Figure below we might require bandwidth: 50 Mbps, delay: 150 ms, loss: < 1 frame in 103) or

implicitly (e.g., the bandwidth of the input stream to the network connection K is the result of applying MPEG-1 compression to the camera output).But the more usual case is that we need to specify a value and a range of permissible variation. Here we consider this requirement for each of the parameters:

Bandwidth

Most video compression techniques produce a stream of frames of differing sizes depending on the original content of the raw video. For MPEG, the average compression ratio is between 1:50 and 1:100, but this will vary dynamically depending on content; for example, the required bandwidth will be highest when the content is changing most rapidly. Because of this, it is often useful to quote QoS parameters as maximum, minimum or average values, depending on the type of QoS management regime that will be used.

Another problem that arises in the specification of the bandwidth is the characterization of burstiness. Consider three streams of 1 Mbps. One stream transfers a single frame of 1 Mbit every second, the second is an asynchronous stream of computer-generated animation elements with an average bandwidth of 1 Mbps and the third sends a 100-bit sound sample every microsecond. Whereas all three streams require the same bandwidth, their traffic patterns are very different.

One way to take care of irregularities is to define a burst parameter in addition to rate and frame size. The burst parameter specifies the maximum number of media elements that may arrive early – that is, before they should arrive according to the regular arrival rate.

The model of linear-bounded arrival processes (LBAP) used in Anderson [1993] defines the maximum number of messages in a stream during any time interval t as Rt + B, where R is the rate and B is the maximum size of burst. The advantage of using this model is that it nicely reflects the characteristics of multimedia sources: multimedia data read from disks are usually delivered in large blocks, and data received from networks often arrive in the form of sequences of smaller packets. In this case, the burst parameter defines the amount of buffer space required to avoid loss.

Latency

Some timing requirements in multimedia result from the stream itself: if a frame of a stream does not get processed with the same rate at which frames arrive, a backlog builds up and buffer capacity may be exceeded. If this is to be avoided, a frame must on average not remain in a buffer for longer than 1/R, where R is the frame rate of a stream. If backlogs do occur, the number and size of the backlogs will affect the maximum end-to-end delay of a stream, in addition to the processing and propagation times.

Other latency requirements arise from the application environment. In conferencing applications, the need for apparently instantaneous interaction between the participants makes it necessary to achieve absolute end-to-end delays of no more than 150 ms to avoid problems in the human perception of the conversation, whereas for the replay of stored video, to ensure a proper system response to commands such as Pause and Stop, the maximum latency should be on the order of 500 ms.

A third consideration for the delivery time of multimedia messages is jitter –variation in the period between the delivery of two adjacent frames. Whereas most multimedia devices make sure that they present data at its regular rate without variation, software presentations (for example, in a software decoder for video frames) need to take extra care to avoid jitter. Jitter is essentially solved by buffering, but the scope for jitter removal is limited, because total end-to-end delay is constrained by the consideration mentioned above. Thus the playback of media sequences also requires media elements to arrive before fixed deadlines.

Loss Rate

Loss rate is the most difficult QoS parameter to specify. Typical loss rate values result from probability calculations about overflowing buffers

and delayed messages. These calculations are based either on worst-case assumptions or on standard distributions. Neither of these is necessarily a good match for practical situations.

However, loss rate specifications are necessary to qualify the bandwidth and latency parameters: two applications may have the same bandwidth and latency characteristics; but they will look dramatically different if one application loses every fifth media element and the other loses only one in a million.

As with bandwidth specifications, where not just the volume of data sent in a time interval but its distribution over the time interval is important, a loss rate specification needs to determine the time interval during which to expect a certain level of loss. In particular, loss rates given for infinite time spans are not useful, as any loss over a short time may exceed the long-term rate significantly.

Traffic Shaping

Traffic shaping is the term used to describe the use of output buffering to smooth the flow of data elements. The bandwidth parameter of a multimedia stream typically provides an idealistic approximation of the actual traffic pattern that will occur when the stream is transmitted. The closer the actual traffic pattern matches the description, the better a system will be able to handle the traffic, in particular when it uses scheduling methods that are designed for periodic requests.

The LBAP model of bandwidth variations calls for regulation of the burstiness of multimedia streams. Any stream can be regulated by inserting a buffer at the source and by defining a method by which data elements leave the buffer. A good illustration of this method is the image of a leaky bucket (Figure 20.6a): the bucket can be filled arbitrarily with water until it is full; through a leak at the bottom of the bucket water will flow continuously. The leaky bucket algorithm ensures that a stream will never flow with a rate higher than R. The size of the buffer B defines the maximum burst a stream can incur without losing elements. B also bounds the time for which an element will remain in the bucket.

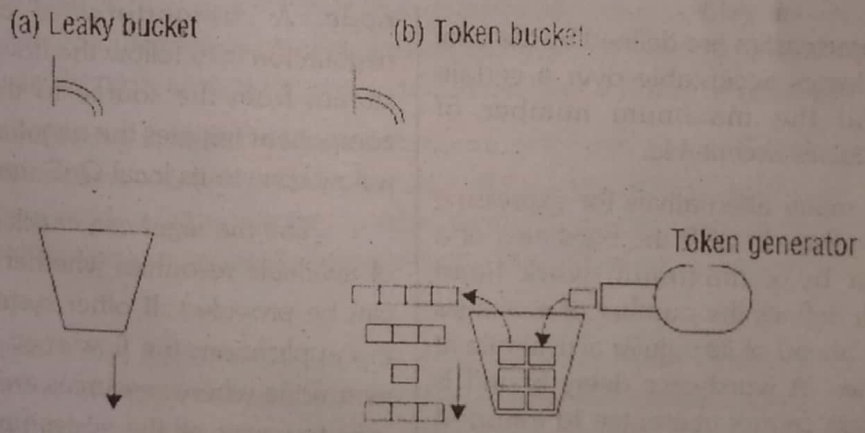


Fig. : Traffic Shaping algorithms

The leaky bucket algorithm completely eliminates bursts. Such elimination is not always necessary as long as bandwidth is bounded over any time interval. The token bucket algorithm achieves this while allowing larger bursts to occur when a stream has been idle for a while (Figure 20.6b). It is a variation of the leaky bucket algorithm in which tokens to send data are generated at a fixed rate, R. They are collected in a bucket of size B. Data of size S can be sent only if at least S tokens are in the bucket. The send process then removes these S tokens. The token bucket algorithm ensures that over any interval t the amount of data sent is not larger than Rt + B. It is, hence, an implementation of the LBAP model.

High peaks of size B can only occur in a token bucket system when a stream has been idle for a while. To avoid these bursts, a simple leaky bucket can be placed behind the token bucket. The flow rate F of this bucket needs to be significantly larger than R for this scheme to make sense. Its only purpose is to break up really large bursts.

Flow specifications

A collection of QoS parameters is typically known as a flow specification, or flow spec for short. Several examples of flow specs exist and all are similar. In Internet RFC 1363, a flow spec is defined as 11 16-bit numeric values (Figure 20.7), which reflect the QoS parameters discussed above in the following way :

- The maximum transmission unit and maximum transmission rate determine the maximum bandwidth required by the stream.
- The token bucket size and rate determine the burstiness of the stream.
- The delay characteristics are specified by the minimum delay that an application can notice (since we wish to avoid over optimization for short delays) and the maximum jitter it can accept.
- The loss characteristics are defined by the total number of losses acceptable over a certain interval and the maximum number of consecutive losses acceptable.

There are many alternatives for expressing each parameter group. In SRP, the burstiness of a stream is given by a maximum work head parameter, which defines the number of messages a stream may be ahead of its regular arrival rate at any point in time. A worst-case delay bound is given: if the system cannot guarantee to transport data within this time span, the data transport will be useless for the application. In RFC 1190, the specification of the ST-II protocol, loss is represented as the probability of each packet being dropped.

All the above examples provide a continuous spectrum of QoS values. If the set of applications and streams to be supported is limited, it may be sufficient to define a discrete set of QoS classes: for example, telephone-quality and high-fidelity audio, live and playback video, etc. The requirements of all classes must be implicitly known by all system components; the system may even be configured for a certain traffic mix.

	Protocol version
	Maximum transmission unit
Bandwidth:	Token bucket rate
	Token bucket size
	Maximum transmission rate
	Minimum delay noticed
Delay:	Maximum delay variation
	Loss sensitivity
Loss:	Burst loss sensitivity
	Loss interval
	Quality of service guarantee

Fig. : The RFC 1363 Flow Spec

Negotiation Procedures

For distributed multimedia applications, the components of a stream are likely to be located in several nodes. There will be a QoS manager at each node. A straightforward approach to QoS negotiation is to follow the flow of data along each stream from the source to the target. A source component initiates the negotiation by sending out a flow spec to its local QoS manager.

The manager can check against its database of available resources whether the requested QoS can be provided. If other systems are involved in the application, the flow spec is forwarded to the next node where resources are required. The flow spec traverses all the nodes until the final target is reached. Then the information on whether the desired QoS can be provided by the system is passed back to the source. This simple approach to negotiation is satisfactory for many purposes, but it does not consider the possibilities for conflict between concurrent QoS negotiations starting at different nodes. A distributed transactional QoS negotiation procedure would be required for a full solution to this problem.

5.6 Admission Control

Q8. Write about Admission control.

Ans :

Admission control regulates access to resources to avoid resource overload and to protect resources from requests that they cannot fulfill. It involves turning down service requests should the resource requirements of a new multimedia stream violate existing QoS guarantees.

An admission control scheme is based on some knowledge of both the overall system capacity and the load generated by each application. The bandwidth requirement specification for an application may reflect the maximum amount of bandwidth that an application will ever require, the minimum bandwidth it will need to function, or some average value in between. Correspondingly, an admission control scheme may base its resource allocation on any of these values.

For resources that have a single allocator, admission control is straightforward. Resources that have distributed access points, such as many local area networks, require either a centralized admission control entity or some distributed admission control algorithm that avoids conflicting concurrent admissions. Bus arbitration within workstations falls into this category; however, even multimedia systems that perform bandwidth allocation extensively do not control bus admission, as bus bandwidth is not considered to be in the window of scarcity.

Bandwidth Reservation

Because of the potential under-utilization that can occur, it is common to overbook resources. The resulting guarantees, often called statistical or soft guarantees to distinguish them from the deterministic or hard guarantees introduced before, are only valid with some (usually very high) probability.

Statistical guarantees tend to provide better resource utilization as they do not consider the worst case. But just as when resource allocation is based on minimum or average requirements, simul-taneous peak loads can cause drops in service quality; applications have to be able to handle these drops.

Statistical multiplexing is based on the hypothesis that for a large number of streams the aggregate bandwidth required remains nearly constant regardless of the bandwidth of individual streams. It assumes that when one stream sends a large quantity of data, there will also be another stream that sends a small quantity, and overall the requirements will balance out. This, however, is only the case for uncorrelated streams.

As experiments show, multimedia traffic in typical environments does not obey this hypothesis. Given a larger number of bursty streams, the aggregate traffic still remains bursty. The term self-similar has been applied to this phenomenon, meaning that the aggregate traffic shows similarity to the individual streams of which it is composed.

Statistical Multiplexing

Because of the potential under-utilization that can occur, it is common to overbook resources. The resulting guarantees, often called statistical or soft guarantees to distinguish them from the deterministic or hard guarantees introduced before, are only valid with some (usually very high) probability.

Statistical guarantees tend to provide better resource utilization as they do not consider the worst case. But just as when resource allocation is based on minimum or average requirements, simultaneous peak loads can cause drops in service quality; applications have to be able to handle these drops.

Statistical multiplexing is based on the hypothesis that for a large number of streams the aggregate bandwidth required remains nearly constant regardless of the bandwidth of individual streams. It assumes that when one stream sends a large quantity of data, there will also be another stream that sends a small quantity, and overall the requirements will balance out. This, however, is only the case for uncorrelated streams.

5.7 Resource Management

Q9. Explain Resource Scheduling in Resource Management.

Ans :

To provide a certain QoS level to an application, not only does a system need to have sufficient resources (performance), but it also needs to make these resources available to an application when they are needed (scheduling).

Resource Scheduling

Processes need to have resources assigned to them according to their priority.

A resource scheduler determines the priority of processes based on certain criteria.

Traditional CPU schedulers in time-sharing systems often base their priority assignments on responsiveness and fairness: I/O-intensive tasks get high priority to guarantee fast response to user requests, CPU-bound tasks get lower priorities, and overall, processes in the same class are treated equally.

Both criteria remain valid for multimedia systems, but the existence of deadlines for the delivery of individual multimedia data elements changes the nature of the scheduling problem. Real-time scheduling algorithms can be applied to this problem, as discussed below.

As multimedia systems have to handle both discrete and continuous media, it becomes a challenge to provide sufficient service to time-dependent streams without causing starvation of discrete media access and other interactive applications.

Scheduling methods need to be applied to (and coordinated for) all resources that affect the performance of a multimedia application. In a typical scenario, a multimedia stream would be retrieved from disk and then sent through a network to a target station, where it is synchronized with a stream coming from another source and finally displayed. The resources required in this example include disk, network and CPU resources as well as memory and bus bandwidth on all systems involved.

Fair Scheduling

If several streams compete for the same resource, it becomes necessary to consider fairness and to prevent ill-behaved streams taking too much bandwidth.

A straightforward approach ensuring fairness is to apply round-robin scheduling to all streams in the same class. Such a method was introduced on a packet-by-packet basis.

The method is used on a bit-by-bit basis, which provides more fairness with respect to varying packet sizes and packet arrival times. These methods are known as fair queuing.

Packets cannot actually be sent on a bit-by-bit basis, but given a certain frame rate it is possible to calculate for each packet when it should have been sent completely.

If packet transmissions are ordered based on this calculation, one achieves almost the same behavior as with actual bit-by-bit round robin, except that when a large packet is sent, it may block the transmission of a smaller packet, which would have been preferred under the bit-by-bit scheme. However, no packet is delayed for longer than the maximum packet transmission time.

All basic round-robin schemes assign the same bandwidth to each stream. To take the individual bandwidth of streams into account, the

bit-by-bit scheme can be extended so that for certain streams a larger number of bits can be transmitted per cycle. This method is called weighted fair queuing.

Real-time Scheduling

Several real-time scheduling algorithms have been developed to meet the CPU scheduling needs of applications such as avionics industrial process control.

Assuming that the CPU resources have not been over allocated (which is the task of the QoS manager), they assign CPU timeslots to a set of processes in a manner that ensures that they complete their tasks on time.

Traditional real-time scheduling methods suit the model of regular continuous multimedia streams very well. Earliest-deadline-first (EDF) scheduling has almost become a synonym for these methods. An EDF scheduler uses a deadline that is associated with each of its work items to determine the next item to be processed: the item with the earliest deadline goes first.

In multimedia applications, we identify each media element arriving at a process as a work item. EDF scheduling is proven to be optimal for allocating a single resource based on timing criteria: if there is a schedule that fulfils all timing requirements, EDF scheduling will find it.

EDF scheduling requires one scheduling decision per message (i.e., per multimedia element). It would be more efficient to base scheduling on elements that exist for a longer time. Rate-monotonic (RM) scheduling is a prominent technique for real-time scheduling of periodic processes that achieves just this. Streams are assigned priorities according to their rate: the higher the rate of work items on a stream, the higher the priority of a stream. RM scheduling has been shown to be optimal for situations that utilize a given bandwidth by less than 69%. Using such an allocation scheme, the remaining bandwidth could be given to non-real-time applications.

To cope with bursty real-time traffic, the basic real-time scheduling methods should be adjusted to distinguish between time-critical and non-critical continuous media work items.

A deadline/work ahead scheduling is introduced. It allows messages in a continuous stream to arrive ahead of time in bursts but applies EDF scheduling to a message only at its regular arrival time.

5.8 INTERNET TELEPHONY – VoIP

10. What is Internet telephony?

Ans :

The Internet was not designed for real-time interactive applications such as telephony, but it has become possible to use it for that purpose as a result of the increases in the capacity and performance of the Internet's core components – its backbone of network links run at 10–40 Gbps and the routers that interconnect them have comparable performance. These components typically run at low load factors (< 10% bandwidth utilization) and IP traffic is therefore seldom delayed or dropped as a result of contention for resources.

This has resulted in the feasibility of building telephony applications over the public Internet by transmitting streams of digitized voice samples from source to destination as UDP packets with no special provision for quality of service. Voice over IP (VoIP) applications such as Skype and Vonage rely on this technique, as do the voice features of instant-messaging applications such as AOL Instant Messaging, Apple iChat AV and Microsoft NetMeeting.