DISTRIBUTED OPERATING SYSTEMS

OBJECTIVES :

- **DISTRIBUTED COMPUTING ENVIRONMENT (DCE)**
- DCE Components
- DCE Cells
- **Computer Networks**
- **COMMUNICATION PROTOCOLS**
- **INTERNETWORKING**
- **ATM TECHNOLOGY**

Dr. P.Selvakumar

Asssistan Professor

Department of Computer Science

Government Arts College

Ariyalur – 621 703

INTRODUCTION TO DISTRIBUTED COMPUTING ENVIRONMENT (DCE)

- Ethernet, IEEE Token Ring, the Internet Protocol suite, and the Internet are
- presented as case studies of networking technologies in Chapter 2.
- The 4.3BSD UNIX interprocess communication mechanism is presented as a case
- study of message-passing technology in Chapter 3.
- SUN RPC and DeE RPC are presented as case studies of Remote Procedure Call
- (RPC) technology in Chapter 4.
- • IVY and Munin are presented as case studies of Distributed Shared Memory
- (DSM) technology in Chapter 5.
- • DCE Distributed Time Service (DTS) is presented as a case study of clock
- synchronization technology in Chapter 6.
- • The DCE threads package is presented as a case study of threads technology in
- Chapter 8.
- • DeE Distributed File Service (DFS) is presented as a case study of distributed file
- system technology in Chapter 9.
- • The various components of DeE naming facility are presented as case studies of
- naming technology in Chapter 10.
- • The Kerberos authentication system and DeE Security Service are presented as
- case studies of security technology in Chapter 11.

What is DCE ?

DCE , it is an integrated set of services and tools that can be installed as a coherent environment on top of existing operating systems and serve as a platform for building and running distributed applications.

A primary goal of DCE is vendor independence. It runs on many different kinds of computers, operating systems, and networks produced by different vendors. For example, some operating systems to which DCE can be easily ported include OSF/I, AIX, DOMAIN OS, ULTRIX, HP-UX, SINIX, SunOS, UNIX System V, VMS, WINDOWS, and OS/2.

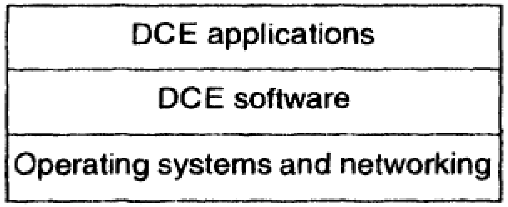


Fig. 1.7 Position of DCE software in a DCE-based distributed system.

As shown in Figure 1.7, DCE is a middleware software layered between the DCE applications layer and the operating system and networking layer. The basic idea is to take a collection of existing machines (possibly from different vendors), interconnect them by a communication network, add the DCE software platform on top of the native operating systems of the machines, and then be able to build and run distributed applications.

DCE Components :

- The main components of DeE are as follows:

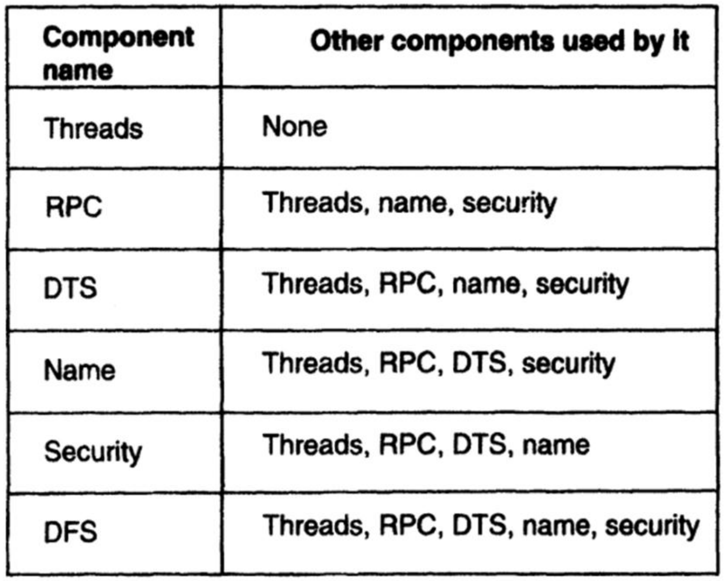
1. Threads package. It provides a simple programming model for building concurrent applications. It includes operations to create and control multiple threads of execution in a single process and to synchronize access to global data within an application. Details are given in Chapter 8.

2. Remote Procedure Call (RPC)facility. It provides programmers with a number of powerful tools necessary to build client-server applications. In fact, the DCE RPC facility is the basis for all communication in DCE because the programming model underlying all of DCE is the client-server model. It is easy to use, is network- and protocol-independent, provides secure communication between a client and a server, and hides differences in data requirements by automatically converting data to the appropriate forms needed by clients and servers. Details are given in Chapter 4.

3. Distributed lime Service (DTS). It closely synchronizes the clocks of all the computers in the system. It also permits the use of time values from external time sources, such as those of the u.s. National Institute for Standards and Technology (NIST), to synchronize the clocks of the computers in the system with external time. This facility can also be used to synchronize the clocks of the computers of one distributed environment with the clocks of the computers of another distributed environment. Details are given in Chapter 6.

4. Name services. The name services of DCE include the Cell Directory Service (CDS), the Global Directory Service (GDS), and the Global Directory Agent (GDA). These services allow resources such as servers, files, devices, and so on, to be uniquely named and accessed in a location-transparent manner. Details are given in Chapter 10.

5. Security Service. It provides the tools needed for authentication and authorization to protect system resources against illegitimate access. Details are given in Chapter 11.
6. Distributed File Service (DFS). It provides a system wide file system that has such characteristics as location transparency, high performance, and high availability. A unique feature of DeE DFS is that it can also provide file services to clients of other file systems. Details are given in Chapter 9.
The DCE components listed above are tightly integrated. It is difficult to give a pictorial representation of their interdependencies because they are recursive. For example, the name services use RPC facility for internal communication among its
Sec. 1.7 • Introduction to Distributed Computing Environment (DeE) 37
various servers, but the RPC facility uses the name services to locate the destination. Therefore, the interdependencies of the various DeE components can be best depicted in tabular form, as shown in Figure 1.8.



Component name	Other components used by it
Threads	None
RPC	Threads, name, security
DTS	Threads, RPC, name, security
Name	Threads, RPC, DTS, security
Security	Threads, RPC, DTS, name
DFS	Threads, RPC, DTS, name, security

DCE Cells

In a DCE system, a *cell* is a group of users, machines, or other resources that typically have a common purpose and share common DCE services. The minimum cell configuration requires a cell directory server, a security server, a distributed time server, and one or more client machines. Each DCE client machine has client processes for security service, cell directory service, distributed time service, RPC facility, and threads facility.

four factors :

1. Purpose. The machines of users working on a common goaL should be put in the same cell, as they need easy access to a common set of system resources. That is, users of machines in the same cell have closer interaction with each other than with users of machines in different cells.

2. Administration. Each system needs an administrator to register new users in the system and to decide their access rights to the system's resources.To perform his or her job properly, an administrator must know the users and the resourcesof the system. Therefore, to simplify administration jobs, all the machines and their users that are known to and manageable by an administrator should be put in a single cell.

3. Security. Machines of those users who have greater trust in each other should be put in the same cell. That is, users of machines of a cell trust each other more than they trust the users of machines of other cells. In such a design, cell boundaries act like firewalls in the sense that accessing a resource that belongs to another cell requires more sophisticated authentication than accessing a resource that belongs to a user's own cell.

4. Overhead. Several DeE operations, such as name resolution and user authentication, incur more overhead when they are performed between cells than when they are performed within the same cell.

Computer Networks

A computer network is a communication system that links end systems by communication lines and software protocols to exchange data between two processes running on different end systems of the network. The end systems are often referred to as nodes, sites, hosts, computers, machines, and so on. The nodes may vary in size and function. Sizewise, a node may be a small microprocessor, a workstation, a minicomputer, or a large supercomputer. Functionwise, a node may be a dedicated system (such as a print server or a file server) without any capability for interactive users, a single-user personal computer, or a general-purpose time-sharing system.

NETWORKS TYPES

Networks arc broadly classified into two types: *local area networks (LANs)* and *wide-area networks (WANs).* differentiate between these two types of networks are as follows :

1. *Geographic distribution.*
2. *Data rate.* Data transmission rates are usually much higher in LANs than in WANs.
3. *Error rate.* Local area networks generally experience fewer than WAN
4. *Communication link.* The most common communication links used in LANs are twisted pair, coaxial cable, and fiber optics. On the other hand, since the sites in a WAN are physically distributed over a large geographic area, the communication links used are by default relatively slow and unreliable. WANs are telephone lines, microwave links, and satellite Channels.
5. *Ownership.* A LAN is owned by a single. A WAN formed by interconnecting multiple LANs each of which may belong to a different organization
6. *Communication cost.* The overall communication costs of a LAN is usually much lower than that of a WAN.

LAN TECHNOLOGIES

This section presents a description of topologies, principles of operation of popular LANs.

LAN Topologies :

The two commonly used network topologies for constructing LANs are multiaccess bus and ring. In a simple *multiaccess bus network*, all sites are directly connected to a single transmission medium (called the bus) that spans the whole length of the network (Fig. 2.1).

In such a network, two or more simple multiaccess bus networks are interconnected by using repeaters (Fig. 2.2). *Repeaters* are hardware devices used to connect cable segments. They simply amplify and copy electric signals from one segment of a network to its next segment.

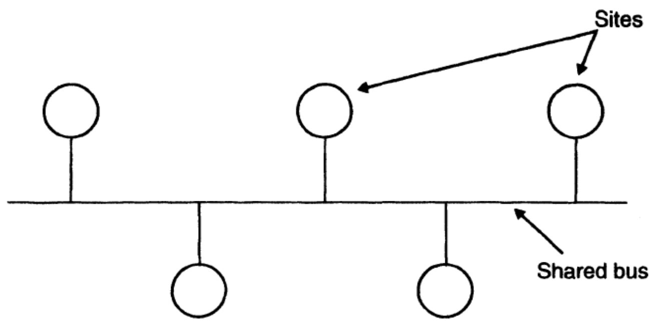


Fig. 2.1 Simple multiaccess bus network topology.

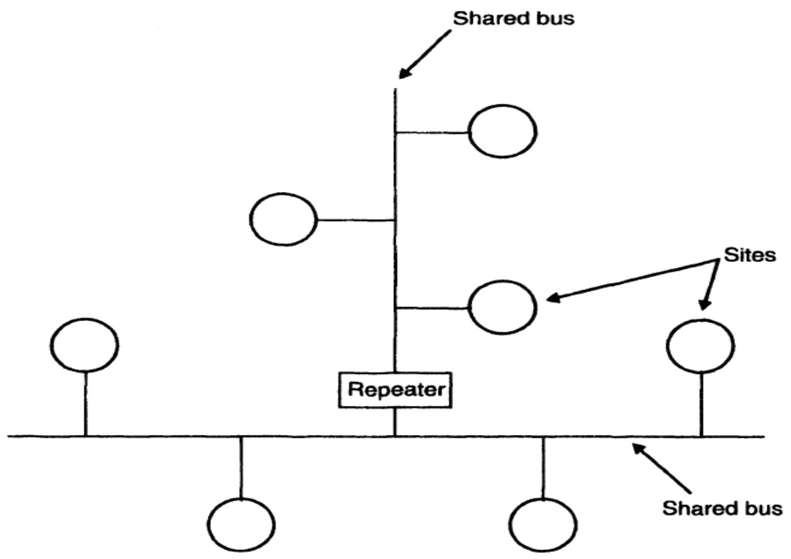


Fig. 2.2 Multiaccess branching bus network topology.

In a **ring network**, each site is connected to exactly two other sites so that a loop is formed (Fig. 2.3). A separate link is used to connect two sites. The links are interconnected by using repeaters. Data is transmitted in one direction around the ring by signaling between sites.

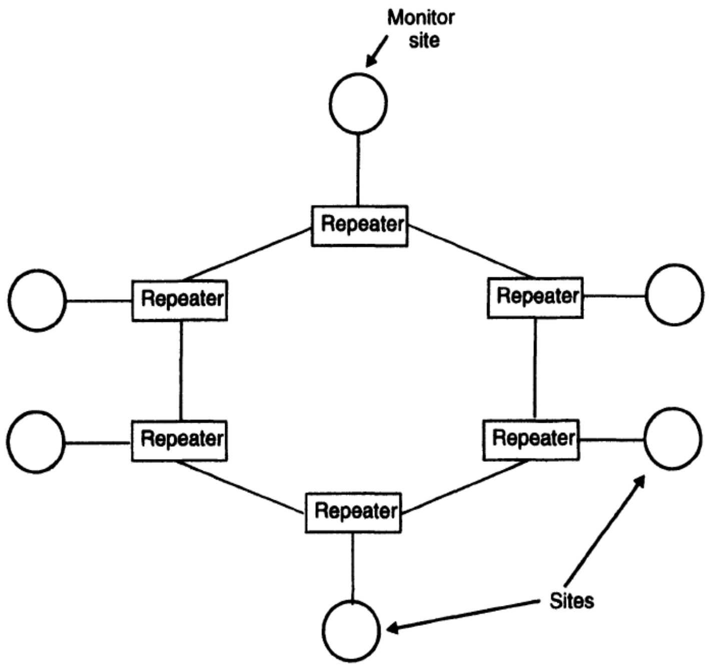


Fig. 2.3 Ring network topology.

The connection cost of a ring network is low and grows only linearly with the increase in number of sites. The average communication cost is directly proportional to the number of sites in the network. If there are *n* sites, at most (n-l) links have to be traversed by a message to reach its destination.

Medium-Access Control Protocols :

InIn case of both multi access bus and ring networks, are needed in a multi access environment to control the access to a shared channel. These schemes are known as *medium-access control protocols*.

- **The CSMA/CD Protocol :** The *CSMA/CD* scheme employs decentralized control of the shared medium, In this scheme is comprised of the following three mechanisms :

1.Carrier sense and defer mechanism.

2. Collision detection mechanism.

3. Controlled retransmission mechanism.

- **The Token Ring Protocol :** This scheme also employs decentralized control of the shared medium. In this scheme, access to the shared medium is controlled by using a single token that is circulated among the sites in the system. A *token* is a special type of message (having a unique bit pattern) that entitles its holder to use the shared medium for transmitting its messages.
- **The Slotted-Ring Protocol :** In this scheme, a constant number of fixed-length message slots continuously circulate around the ring. Each slot has two parts-control and data. The control part usually has fields to specify whether the slot is full or empty, the source and destination addresses of the message contained in a full slot, and whether the message in it was successfully received at the destination. On the other hand, the data part can contain a fixed-length message data.

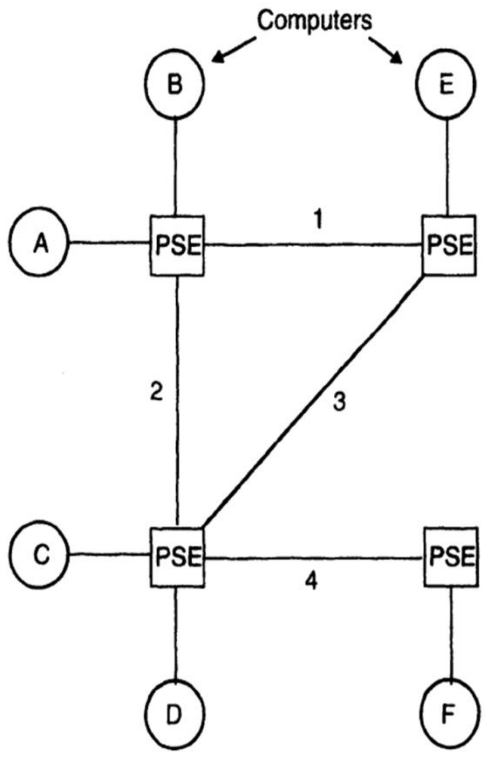
WAN Technologies

WAN, computers located in the same country may be interconnected by coaxial cables (telephone lines), but communications satellites may be used to interconnect two computers that are located in different countries. The computers of a WAN are not connected directly to the communication channels but are connected to hardware devices caned *packet-switching exchanges (PSEs),* which are special-purpose computers dedicated to the task of data communication. Therefore, the communication channels of the network interconnect the PSEs, which actually perform the task of data communication across the network (Fig. 2.6).

We saw that in a WAN communication is achieved by transmitting a packet from its source computer to its destination computer through two or more PSEs. The PSEs provide switching facility to move a packet from one PSE to another until the packet reaches its destination. The two most commonly used schemes are circuit switching and packet switching.

circuit switching : a physical circuit is constructed between the sender and receiver computers during the circuit establishment phase.ex : Public Switched Telephone Network (PSTN).

packet switching : sender-receiver pair only while transmitting a single packet of the message of that pair. Ex : X.25 public packet network and the Internet.



COMMUNICATION PROTOCOLS

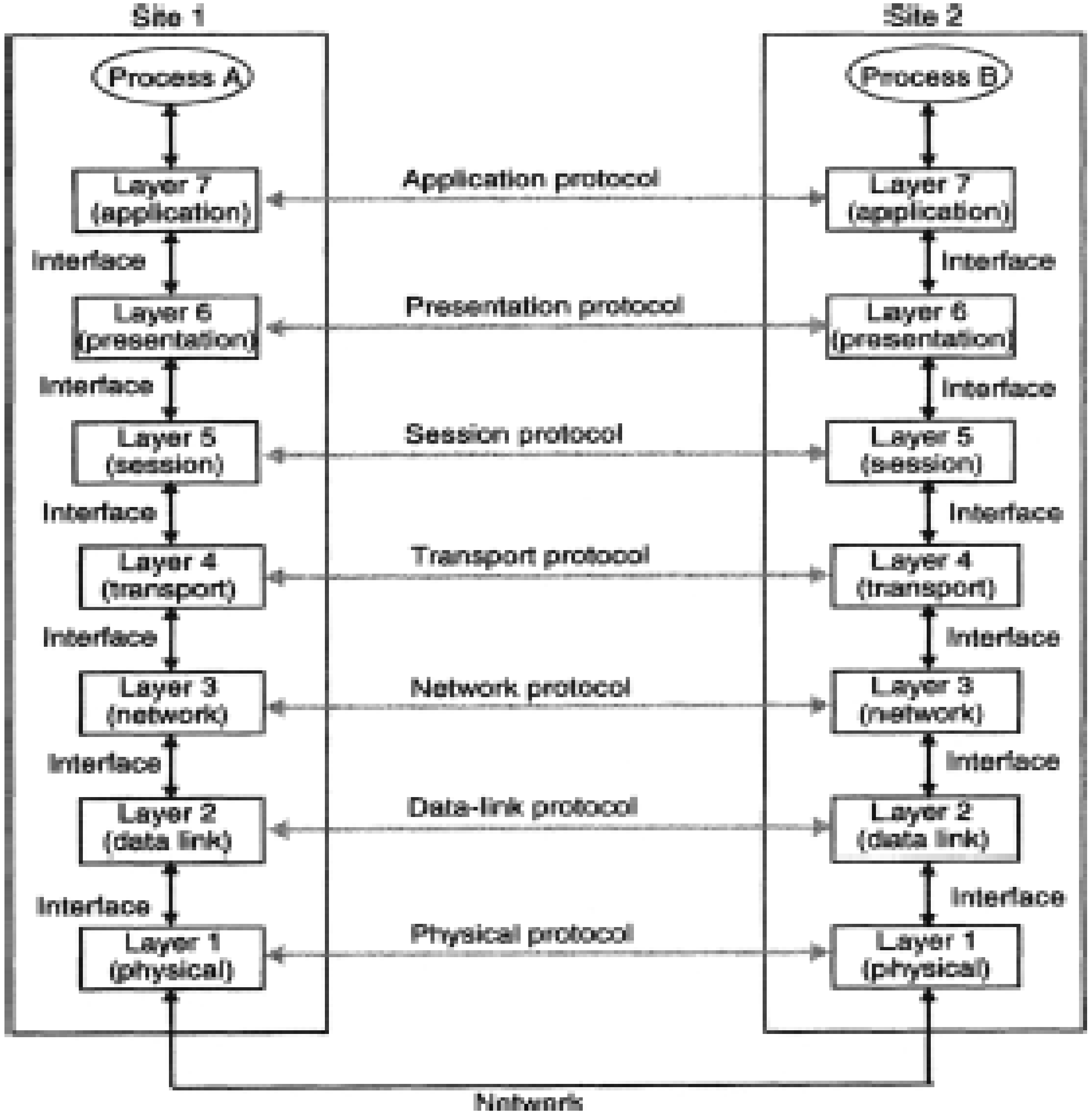
The term **protocol** is used to refer to a set of such rules and conventions. Computer networks are implemented using the concept of *layered protocols.*

network are organized into a series of layers in such a way that each layer contains protocols for exchanging data and providing functions in a logical sense with peer entities at other sites in the network. The terms *protocol suite, protocol family,* or *protocol stack* are used to refer to the collection of protocols (of all layers) of a particular network system.

Protocols for Network Systems

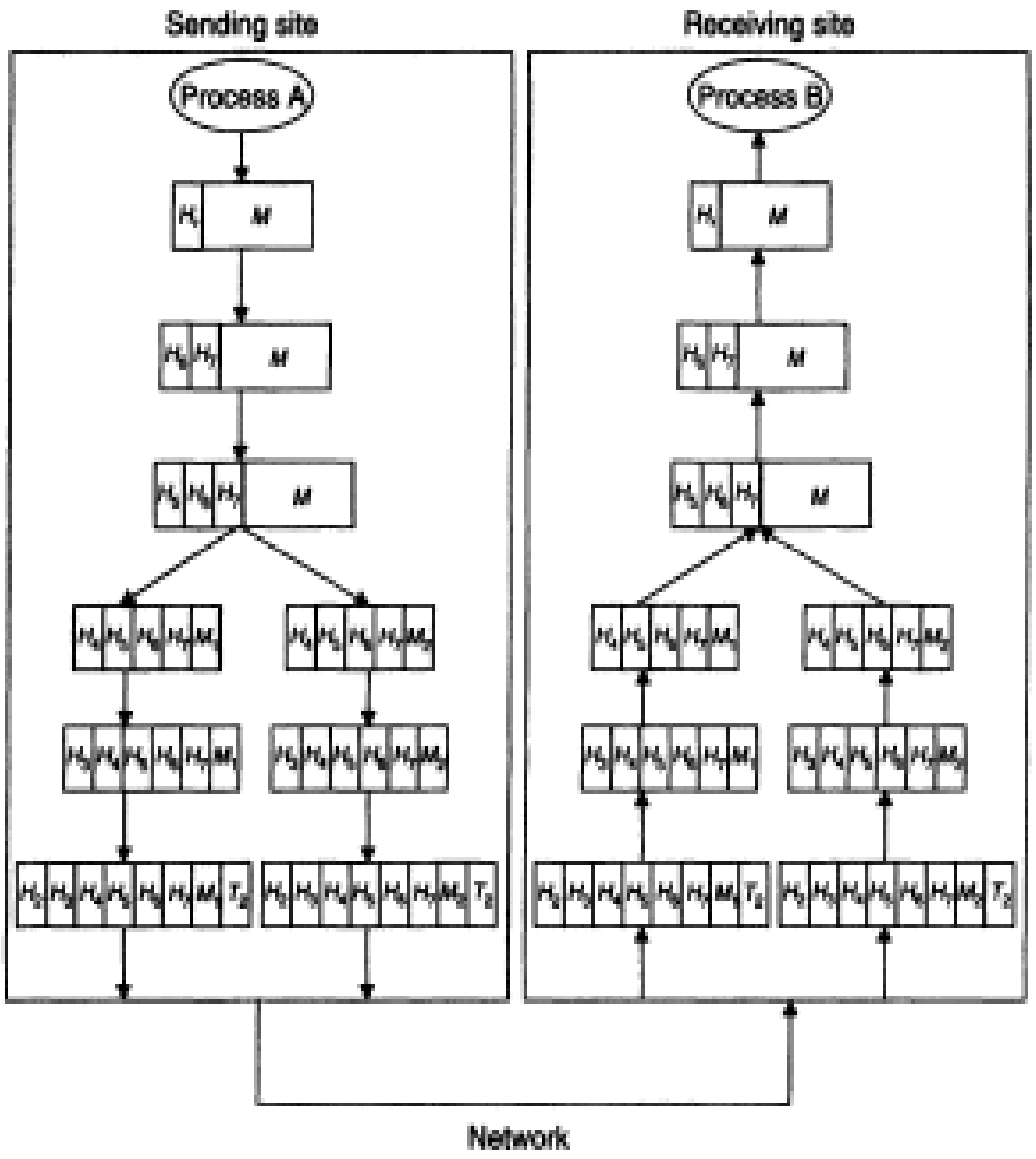
International Standardization Organization (ISO) has developed a reference model that identifies seven standard layers and defines the jobs to be performed at each layer. This model is called the *Open System International Reference Model* (*OSI model)*

The architecture of the **OSI model is shown in Figure 2.7**. It is a seven-layer architecture in which a separate set of protocols is defined for each layer. Thus each layer has an independent function and deals with one or more specific aspects of the communication.


Site 1
Site 2
Process A
Process B
Layer 7 (application)
Layer 6 (presentation)
Layer 5 (session)
Layer 4 (transport)
Layer 3 (network)
Layer 2 (data link)
Layer 1 (physical)
Application protocol
Presentation protocol
Session protocol
Transport protocol
Network protocol
Data-link protocol
Physical protocol
Interface
Network

Example of Message Transfer in the OSI Model.

To illustrate the functions of the various layers of the OSI model, let us consider a simple example of message transmission. With reference to Figure 2.8,



IEEE 802 LAN Reference Model

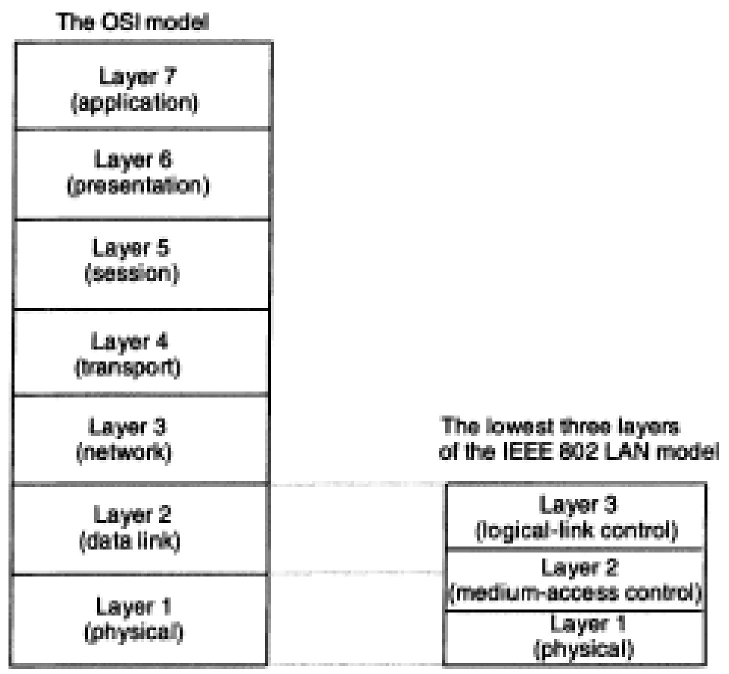
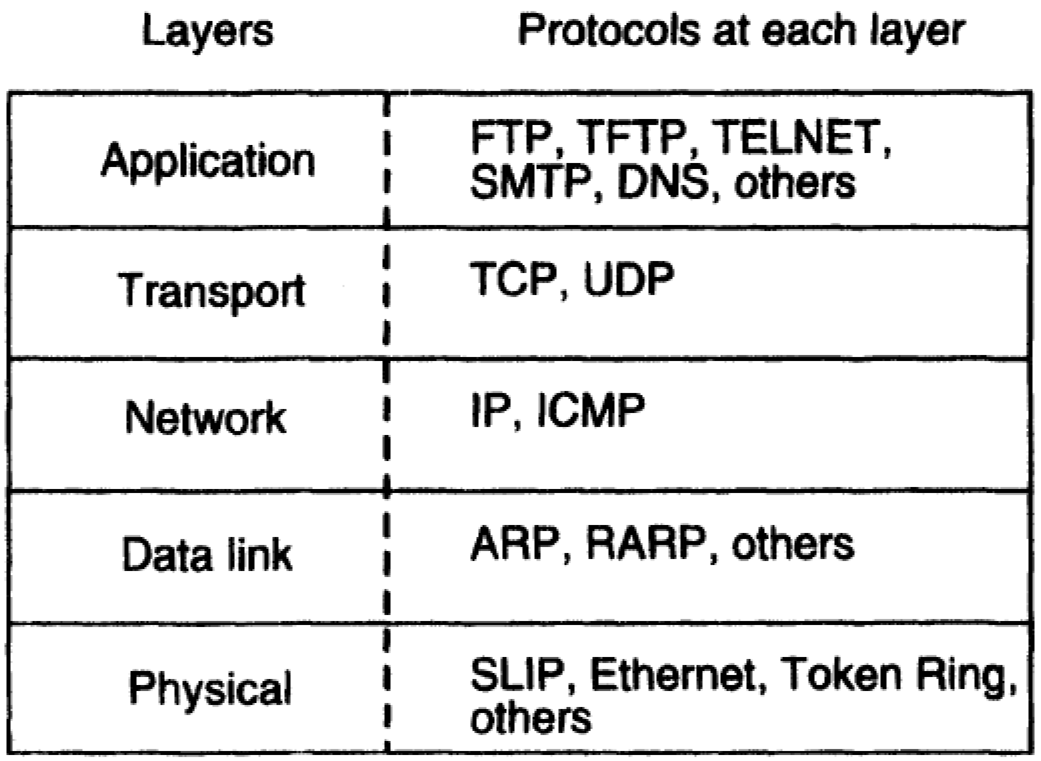


Fig. 2.9 Relationship between the IEEE 802 LAN model and the OSI model.

As shown in the figure, the lowest three layers of the IEEE 802 LAN model are the physical layer, the medium-access-control layer, and the logical-link-control layer. The physical layer defines interface protocols for the following four types of media that are commonly used in LANs: baseband, broadband, fiber optics, and twisted pair. As the name implies, the medium-access-control layer deals with the medium-access-control protocols for LANs.

The Internet Protocol Suite



Layers	Protocols at each layer
Application	FTP, TFTP, TELNET, SMTP, DNS, others
Transport	TCP, UDP
Network	IP, ICMP
Data link	ARP, RARP, others
Physical	SLIP, Ethernet, Token Ring, others

FTP (File Transfer Protocol
TFTP (Trivial File TransferProtocol)
SMTP (Simple Mail Transfer Protocol)
DNS (Domain Name Service)

TCP (Transport Control Protocol)
UDP (User Datagram Protocol)

IP (Internet Protocol)
ICMP (Internet Control Message Protocol)

ARP (Address Resolution Protocol)
RARP (Reverse ARP)

SLIP (Serial Line Internet Protocol)

INTERNETWORKING

Interconnecting of two or more networks to form a single network is called *internetworking*, and the resulting network is called an ***internetwork***. Therefore, a WAN of multiple LANs is an internetwork. Internetworks are often heterogeneous networks composed of several network segments that may differ in topology and protocol.

The three important internetworking issues are how to interconnect multiple (possibly heterogeneous) networks into a single network, which communication medium to use for connecting two networks, and how to manage the resulting internetwork.

Interconnection Technologies

Interconnection technologies enable interconnection of networks that may possibly have different topologies and protocols. Interconnecting two networks having the same topology and protocol is simple because the two networks can easily communicate with each other. However, interconnecting two dissimilar networks that have different topologies and protocols requires an internetworking scheme that provides some common point of reference for the two networks to communicate with each other.

Bridges

Bridges operate at the bottom two layers of the OSI model (physical and data link). Therefore, they are used to connect networks that use the same communication protocols above the data-link layer but mayor may not use the same protocols at the physical and data-link layers. For example, bridges may be used to connect two networks, one of which uses fiber-optic communication medium and the other uses coaxial cable; or one of which uses Ethernet technology and the other uses Token Ring technology. But both networks must use the same high-level protocols (e.g., *TCP/IP* or XNS) to communicate. bridges are also useful in network partitioning.

Routers

Routers are commonly used to interconnect those network segments of large intemetworks that use the same communication protocol. They are particularly useful in controlling traffic flow by making intelligent routing decisions. An internetwork often uses both bridges and routers to handle both routing and multiprotocol issues. This requirement has resulted in the design of devices called *brouters,* which are a kind of hybrid of bridges and routers. They provide many of the advantages of both bridges and routers.

Gateways

Gateways operate at the top three layers of the OSI model (session, presentation, and application). They are the most sophisticated internetworking tools and are used for interconnecting dissimilar networks that use different communication protocols. That is, gateways are used to interconnect networks that are built on totally different communications architectures.

ATM TECHNOLOGY

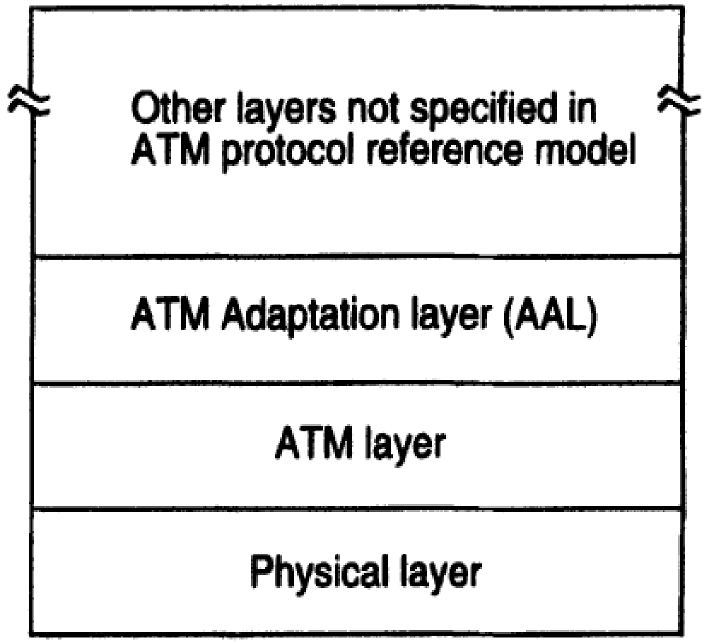
Asynchronous Transfer Mode (ATM) is often escribed as the future computer networking paradigm. It is a high-speed, connection-oriented switching and multiplexing technology that uses short, fixed-length packets (called *cells)* to transmit different types of traffic simultaneously, including voice, video, and data.

Main Features of ATM Technology

- It enables high-bandwidth distributed applications
- It provides high transmission speeds for both local and wide-area networks & services
- It supports both the fundamental approaches to switching (circuit switching and
- packet switching) within a single integrated switching mechanism (called *cell switching).*
- It uses the concept of virtual networking to pass traffic between two locations.
- In addition to point-to-point communication in which there is a single sender and
- a single receiver.
- It enables the use of a single network to efficiently transport a wide range of multimedia data such as text, voice, video, broadcast television, and so on.
- It is flexible in the way it grants access to bandwidth.
- It is a scalable technology.
- It has a fairly solid base of standards.

ATM Protocol Reference Model

The protocol reference model in ATM is divided into three layers-physical layer, ATM layer, and ATM adaptation layer (AAL) (Fig. 2.12). Applications involving data, voice, and video are built on top of these three layers.



Other layers not specified in ATM protocol reference model
ATM Adaptation layer (AAL)
ATM layer
Physical layer

Physical Layer

The physical layer is the bottom-most layer of the ATM protocol suite. It is concerned with putting bits on the wire and taking them off again. It has two sublayers: the physical medium dependent (PMD) sublayer and the transmission convergence (TC) sublayer.

ATM Layer

The ATM layer handles most of the cell processing and routing activities. These include building the cell header, cell multiplexing of individual connections into composite flows of cells, cell demultiplexing of composite flows into individual connections, cell routing, cell payload type marking and differentiation, cell loss priority marking and reduction, cell reception and header validation, and generic flow control of cells.

ATM Adaptation Layer

The functionality of the physical and the ATM layers of the ATM protocol suite is not tailored to any application. We saw that ATM can support various types of traffic, including voice, video, and data.

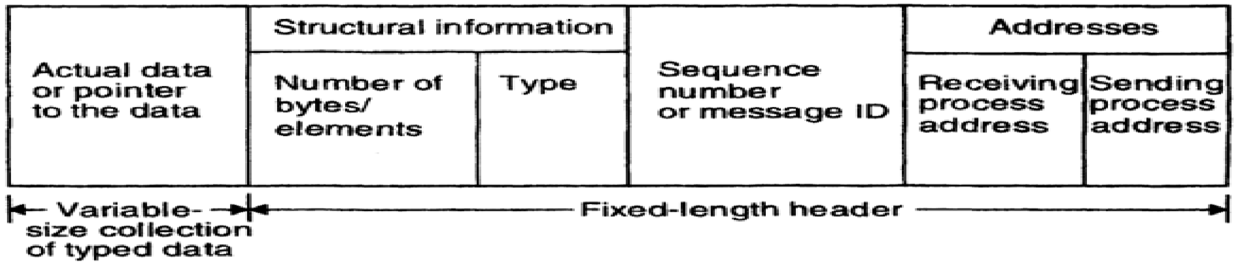
ISSUES IN IPC BY MESSAGE PASSING

A message is a block of information formatted by a sending process in such a manner that it is meaningful to the receiving process. It consists of a fixed-length header and a variable-size collection of typed data objects.

Address. It contains characters that uniquely identify the sending and receiving processes in the network. Thus, this element has two parts-one part is the sending process address and the other part is the receiving process address.

Sequence number. This is the message identifier (ID), which is very useful for identifying lost messages and duplicate messages in case of system failures.

Structural information. This element also has two parts. The *type* part specifies whether the data to be passed on to the receiver is included within the message or the message only contains a pointer to the data, which is stored somewhere outside the contiguous portion of the message. The second part of this element specifies the length of the variable-size message data.



Actual data or pointer to the data
Structural information
Number of bytes/ elements
Type
Sequence number or message ID
Addresses
Receiving process address
Sending process address
Variable-size collection of typed data
Fixed-length header

Fig. 3.2 A typical message structure.

IPC protocol for a message-passing system, the following important issues need to be considered:

- Who is the sender?
- Who is the receiver?
- Is there one receiver or many receivers?
- Is the message guaranteed to have been accepted by its receiver(s)?
- Does the sender need to wait for a reply?
- What should be done if a catastrophic event such as a node crash or a communication link failure occurs during the course of communication?
- What should be done if the receiver is not ready to accept the message: Will the message be discarded or stored in a buffer? In the case of buffering, what should be done if the buffer is full?
- If there are several outstanding messages for a receiver, can it choose the order in
- which to service the outstanding messages?

SYNCHRONIZATION

A central issue in the communication structure is the synchronization imposed on the communicating processes by the communication primitives. The semantics used for synchronization may be broadly classified as *blocking* and *non blocking* types.
The synchronization imposed on the communicating processes basically depends on one of the two types of semantics used for the send and receive primitives.

Two methods is commonly used for this purpose:

1. Polling. In this method, a *test* primitive is provided to allow the receiver to check the buffer status. The receiver uses this primitive to periodically poll the kernel to check if the message is already available in the buffer.
2. Interrupt. In this method, when the message has been filled in the buffer and is ready for use by the receiver, a software interrupt is used to notify the receiving process. This method permits the receiving process to continue with its execution without having to issue unsuccessful *test* requests. Although this method is highly efficient and allows maximum parallelism, its main drawback is that user-level interrupts make programming difficult [Tanenbaum 1995].

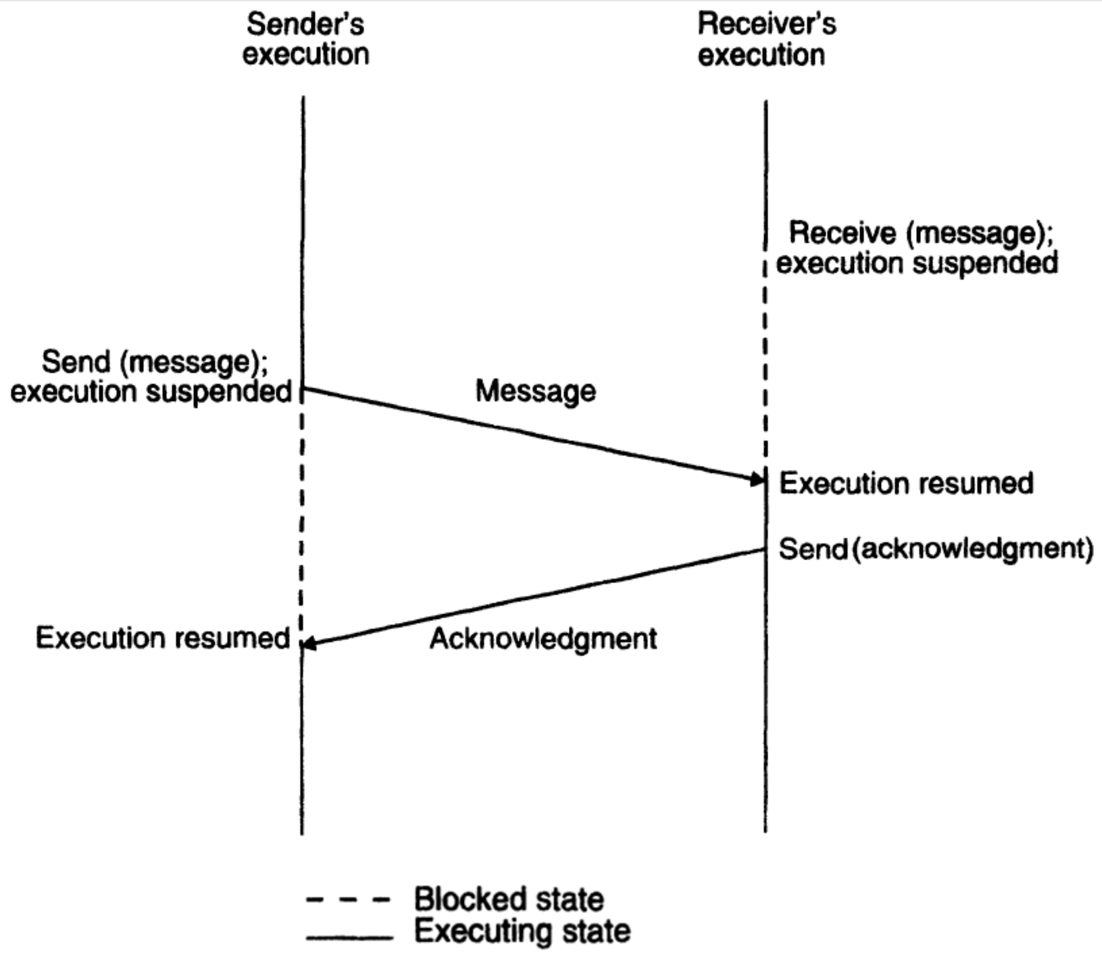


Fig. 3.3 Synchronous mode of communication with both *send* and *receive* primitives having blocking-type semantics.

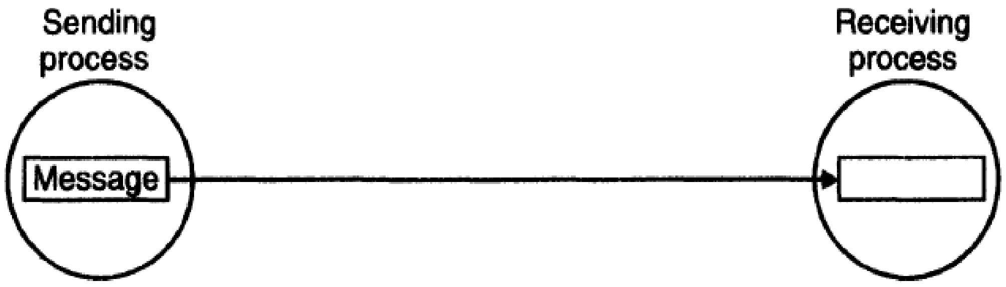
BUFFERING

Messages can be transmitted from one process to another by copying the body of the message from the address space of the sending process to the address space of the receiving process. a buffer in which messages can be stored prior to the receiving process executing specific code to receive the message.

four types of buffering strategies :

1. a *null buffer* (or *no buffering) :*
2. *single-message*
3. *unbounded capacity.*
4. *finite-bound (multiple-message)* buffers.

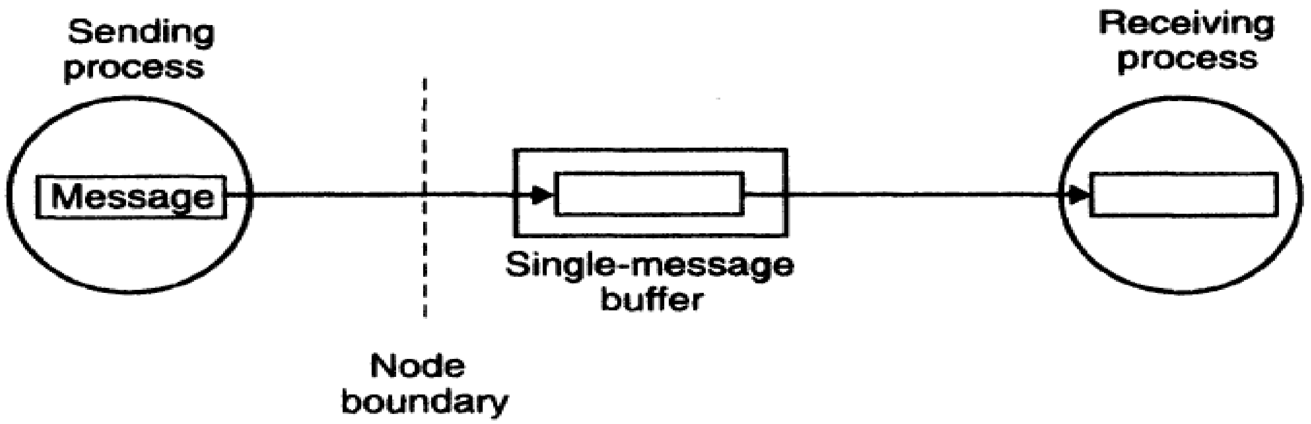
1. a *null buffer (or no buffering) :* In case of no buffering, there is no place to temporarily store the message.



(*a*)

single-message Buffer :

The main idea behind the single-message buffer strategy is to keep the message ready for use at the location of the receiver. The message buffer may either be located in the kernel's address space or in the receiver process's address space.



(*b*)

Unbounded – Caacity Buffer :

In the asynchronous mode of communication, since a sender does not wait for the receiver to be ready, there may be several pending messages that have not yet been accepted by the receiver. Therefore, an unbounded-capacity message buffer that can store all unreceived messages is needed to support asynchronous communication with the assurance that all the messages sent to the receiver will be delivered.

4. finite-bound (multiple-message) buffers :

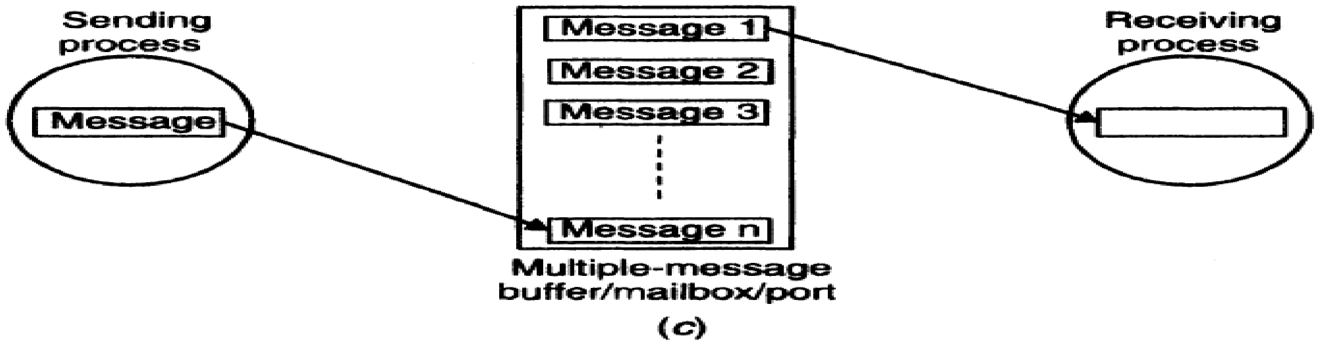
Unbounded capacity of a buffer is practically impossible. Therefore, in practice, systems using asynchronous mode of communication use finite-bound buffers, also known as multiple-message buffers. When the buffer has finite bounds, a strategy is also needed for handling the problem of a possible buffer overflow. The buffer overflow problem can be dealt with in one of the following two ways:

1. Unsuccessful communication. In this method, message transfers simply fail

whenever there is no more buffer space. The *send* normally returns an error message to the sending process, indicating that the message could not be delivered to the receiver because the buffer is full. Unfortunately, the use of this method makes message passing less reliable.

2. Flow-controlled communication. The second method is to use flow control, which

means that the sender is blocked until the receiver accepts some messages, thus creating space in the buffer for new messages. This method introduces a synchronization between the sender and the receiver and may result in unexpected deadlocks. Moreover, due to the synchronization imposed, the asynchronous send does not operate in the truly asynchronous mode for all *send* commands.



MUlTI DATAGRAM MESSAGES

Almost all networks have an upper bound on the size of data that can be transmitted at a time. This size is known as the *maximum transfer unit (MTU)* of a network.

A message whose size is greater than the MTU has to be fragmented into multiples of the MTU, and then each fragment has to be sent separately.

Each fragment is sent in a packet that has some control information in addition to the message data. Each packet is known as a *datagram.* Messages smaller than the MTU of the network can be sent in a single packet and are known as *single-datagram messages.*

On the other hand, messages larger than the MTU of the network have to be fragmented and sent in multiple packets. Such messages are known as *multi datagram messages*.

ENCODING AND DECODING OF MESSAGE DATA

A message data should be meaningful to the eceiving process. This implies that, ideally, the structure of program objects should be preserved while they are being transmitted from the address space of the sending process to the address space of the receiving process.

it is very difficult to achieve this goal mainly because of two reasons:

1. An absolute pointer value loses its meaning when transferred from one process address space to another. Therefore, such program objects that use absolute pointer values cannot be transferred in their original form, and some other form of epresentation must be used to transfer them.

2. Different program objects occupy varying amount of storage space. To be meaningful, a message must normally contain several types of program objects, such as long integers, short integers, variable-length character strings, and so on. In this .case, to make the message meaningful to the receiver, there must be some way for the receiver to identify which program object is stored where in the message buffer and how much space each program object occupies.

two representations may be used for the Encoding and Decoding of a message data:

1. *In tagged representation* the type of each program object along with its value is encoded in the message. In this method, it is a simple matter for the receiving process to check the type of each program object in the message because of the self-describing nature of the coded data format.

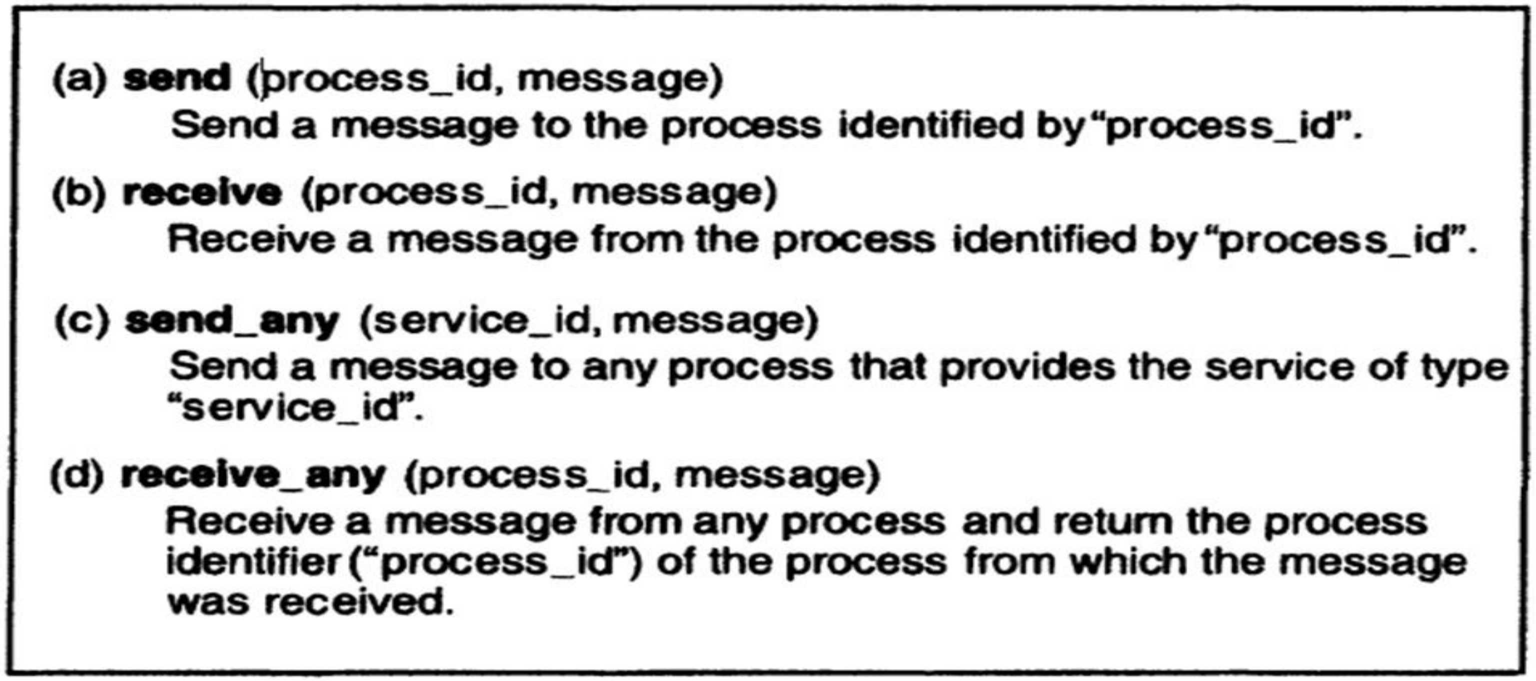
2. *In untagged representation* the message data only contains program objects. No information is included in the message data to specify the type of each program object. In this method, the receiving process must have a prior knowledge of how to decode the received data because the coded data format is not self-describing.

PROCESS ADDRESSING

a message-passing system usually supports two types of process addressing:

1. Explicit addressing. The process with which communication is desired is explicitly named as a parameter in the communication primitive used. Primitives *(a)* and *(b)* of Figure 3.5 require explicit process addressing.

2. Implicit addressing. A process willing to communicate does not explicitly name a process for communication. Primitives *(c)* and *(d)* of Figure 3.5 support implicit process addressing.



(a) **send** (process_id, message)
Send a message to the process identified by "process_id".

(b) **receive** (process_id, message)
Receive a message from the process identified by "process_id".

(c) **send_any** (service_id, message)
Send a message to any process that provides the service of type "service_id".

(d) **receive_any** (process_id, message)
Receive a message from any process and return the process identifier ("process_id") of the process from which the message was received.

Fig. 3.5 Primitives for explicit and implicit addressing of processes.

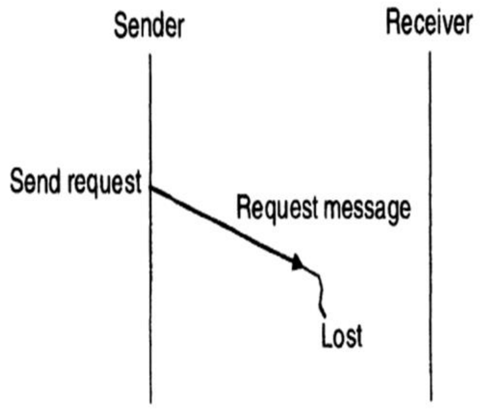
FAilURE HANDLING

While a distributed system may offer potential for parallelism, it is also prone to partial failures such as a node crash or a communication link failure. As shown in Figure 3.6,

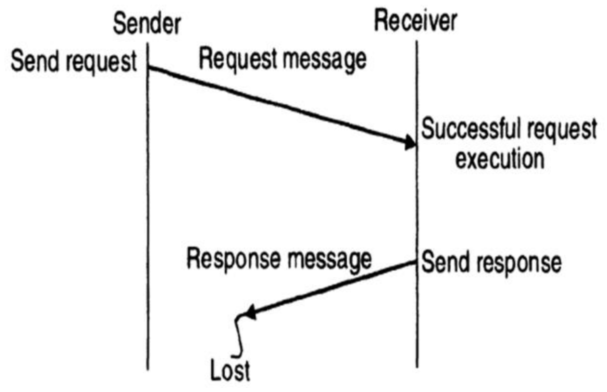
1. Loss of request message. This may happen either due to the failure of communication link between the sender and receiver or because the receiver's node is down at the time the request message reaches there.

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

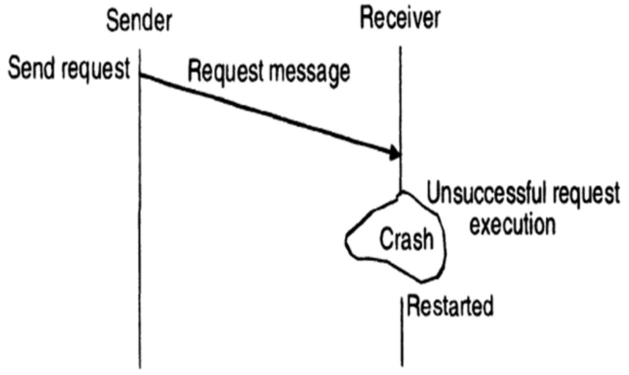
3. Unsuccessful execution of the request. This happens due to the receiver's node crashing while the request is being processed.



(a)



(b)



(c)

(a) Request Message is lost (b) Response message is lost. (c) Receiver's computer crashed.

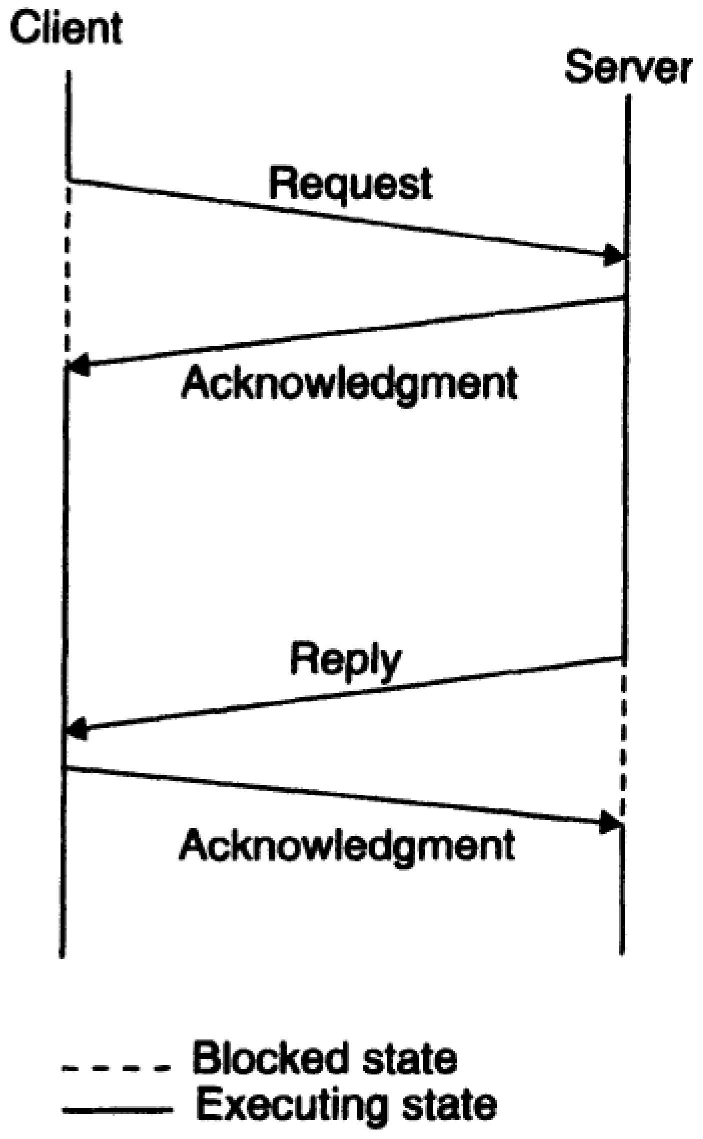


Fig. 3.7 The four-message reliable IPC protocol for client-server communication between two processes.

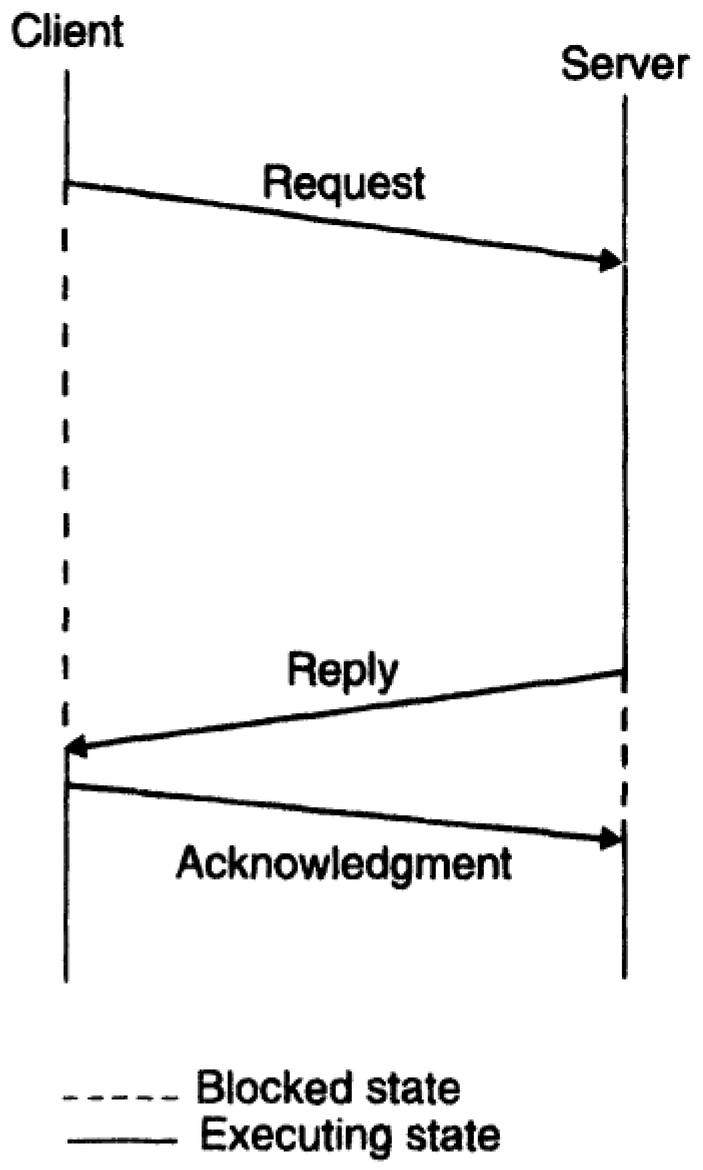


Fig. 3.8 The three-message reliable IPC protocol for client-server communication between two processes.

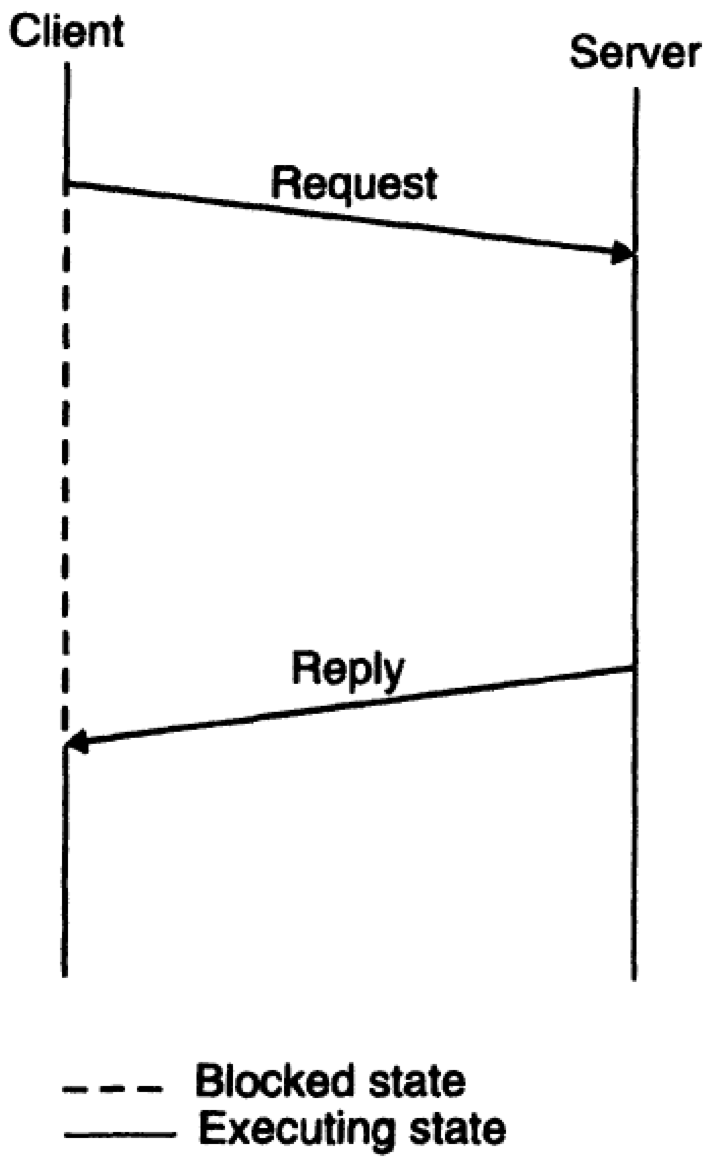


Fig. 3.9 The two-message IPC protocol used in many systems for client-server communication between two processes.

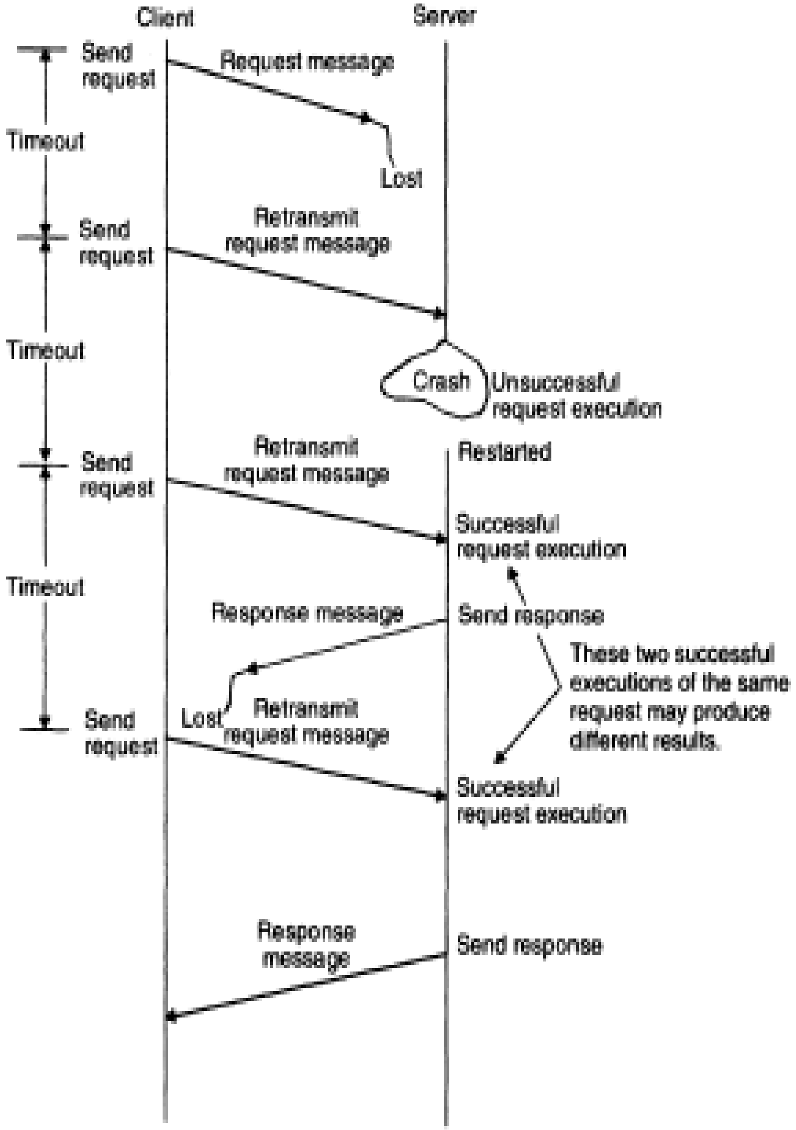

Client
Server
Send request
Request message
Timeout
Lost
Send request
Retransmit request message
Timeout
Crash
Unsuccessful request execution
Restarted
Send request
Retransmit request message
Successful request execution
Timeout
Response message
Send response
These two successful executions of the same request may produce different results.
Lost
Send request
Retransmit request message
Successful request execution
Response message
Send response

Fig. 3.10 An example of fault-tolerant communication between a client and a server.

GROUP COMMUNICATION

three types of group communication are possible:

1. One to many (single sender and multiple receivers)
2. Many to one (multiple senders and single receiver)
3. Many to many (multiple senders and multiple receivers)

In this scheme, there are multiple receivers for a message sent by a single sender. One-tomany scheme is also known as *multicast communication*. A special case of multicast communication is *broadcast communication*, in which the message is sent to all processors connected to a network.

Group Management

In case of one-to-many communication, receiver processes of a message form a group. Such groups are of two types--closed and open.

A *closed group* is one in which only the members of the group can send a message to the group. an *open group* is one in which any process in the system can send a message to the group as a whole.

A simple mechanism for this is to use a centralized *group server* process. All requests to create a group, to delete a group, to add a member to a group, or to remove a member from a group are sent to this process. Therefore, it is easy for the group server to maintain up-to-date information of all existing groups and their exact membership.

Group Addressing

A two-level naming scheme is normally used for group addressing.

The high-level group : is an ASCII string that is independent of the location information of the processes

The low-level group : name depends to a large extent on the underlying hardware.

Such a network address is called a *multicast address*. A packet sent to a multicast address is automatically delivered to all machines listening to the address.

Networks with broadcasting facility declare a certain address, such as zero, as a *broadcast address.*

Message Delivery to Receiver Processes

When the packet reaches a machine, the kernel of that machine extracts the list of process identifiers from the packet and forwards the message in the packet to those processes in the list that belong to its own machine.

Buffered and Unbuffered Multicast

Unbuffered multicast, the message is not buffered for the receiving process and is lost if the receiving process is not in a state ready to receive it.

Buffered multicast, the message is buffered for the receiving processes, so each process of the multicast group will eventually receive the message.

Many-to-One Communication

In this scheme, multiple senders send messages to a single receiver. The single receiver may be selective or nonselective. A *selective receiver* specifies a unique sender; a message exchange takes place only if that sender sends a message. On the other hand, a *nonselective receiver* specifies a set of senders, and if anyone sender in the set sends a message to this receiver, a message exchange takes place.

Thus we see that an important issue related to the many-to-one communication scheme is non determinism. The receiver may want to wait for information from any of a group of senders, rather than from one specific sender. As it is not known in advance which member (or members) of the group will have its information available first, such behavior is nondeterministic. In some cases it is useful to dynamically control the group of senders from whom to accept message.

Many-to-Many Communication

In this scheme, multiple senders send messages to multiple receivers. The one-to-many and many-to-one schemes are implicit in this scheme. an important issue related to many-to-many communication scheme is that of *ordered message delivery.* Ordered message delivery ensures that all messages are delivered to all receivers in an order acceptable to the application. This property is needed by many applications for their correct functioning.

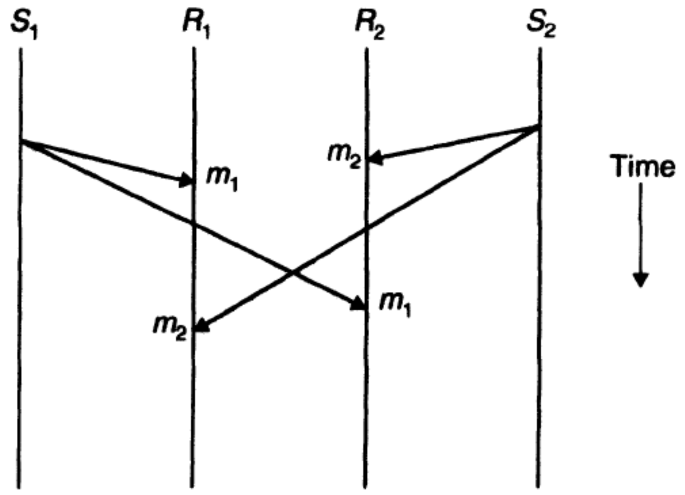


Fig. 3.14 No ordering constraint for message delivery.

The commonly used semantics for ordered delivery of multicast messages are :

absolute ordering

consistent ordering

causal ordering

Absolute ordering

This semantics ensures that all messages are delivered to all receiver processes in the exact order in which they were sent

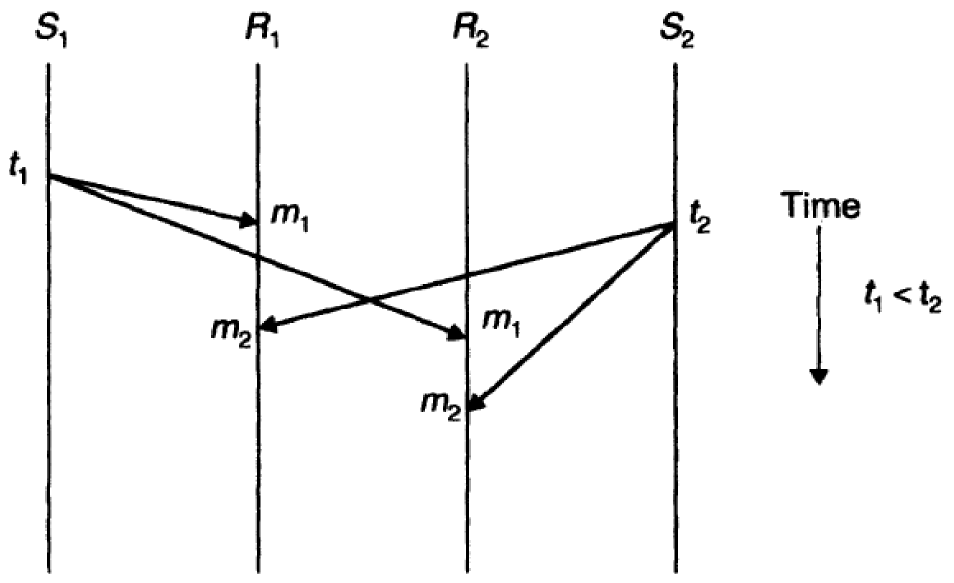


Fig. 3.15 Absolute ordering of messages.

consistent ordering

This semantics ensures that all messages are delivered to all receiver processes in the same order. However, this order may be different from the order in which messages were sent (see Fig. 3.16).

One method to implement consistent-ordering semantics is to make the many-to-many scheme appear as a combination of many-to-one and one-to-many schemes. That is, the kernels of the sending machines send messages to a single receiver (known as a *sequencer)* that assigns a sequence number to each message and then multicasts it.

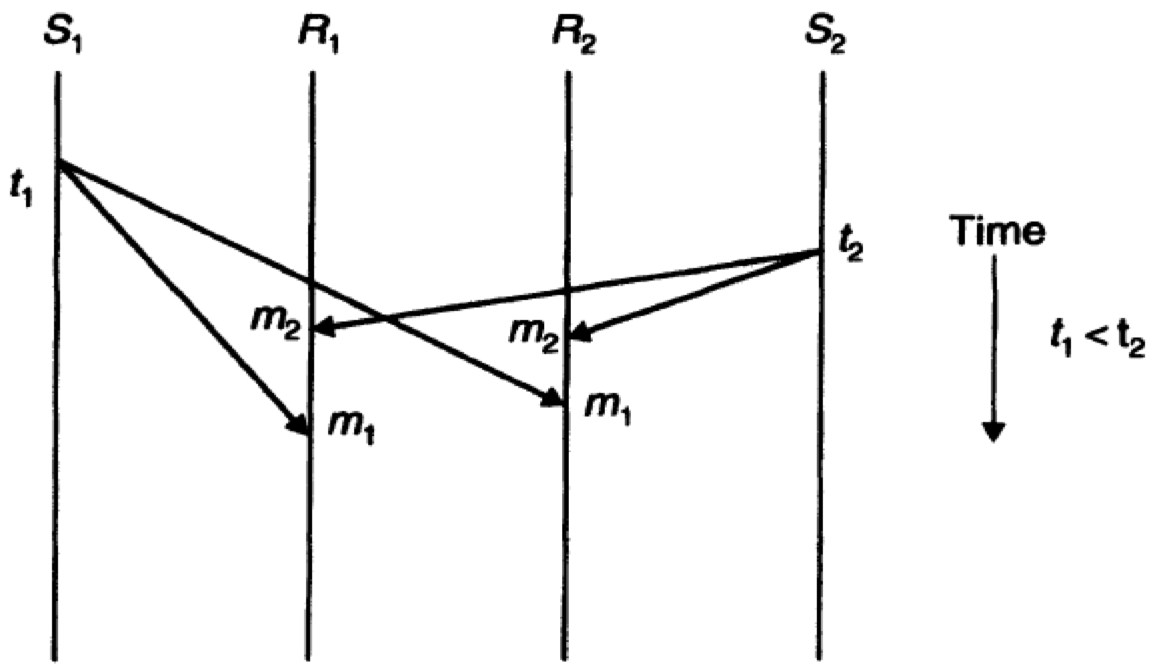


Fig. 3.16 Consistent ordering of messages.

Causal Ordering

An example of causal ordering of messages is given in Figure 3.17. In this example, sender SI sends message m1 to receivers *R1, R2 ,* and *R3* and sender *S2* sends message *m2* to receivers *R2 ,* and *R3 .* On receiving *nu ,* receiver *R1* inspects it, creates a new message *ms,* and sends *m-;* to *R2* and *R3 .* Note that the event of sending *m3* is causally related to the event of sending *m,* because the contents of *m-;* might have been derived in part from ml; hence the two messages must be delivered to both *R2* and *R3* in the proper order, *m,* before *m3'* Also note that since *m2* is not causally related to either *m,* or *m3' m2* can be delivered at any time to *R2* and *R3* irrespective of *m,* or *m-,* This is exactly what the example of Figure 3.17 shows.

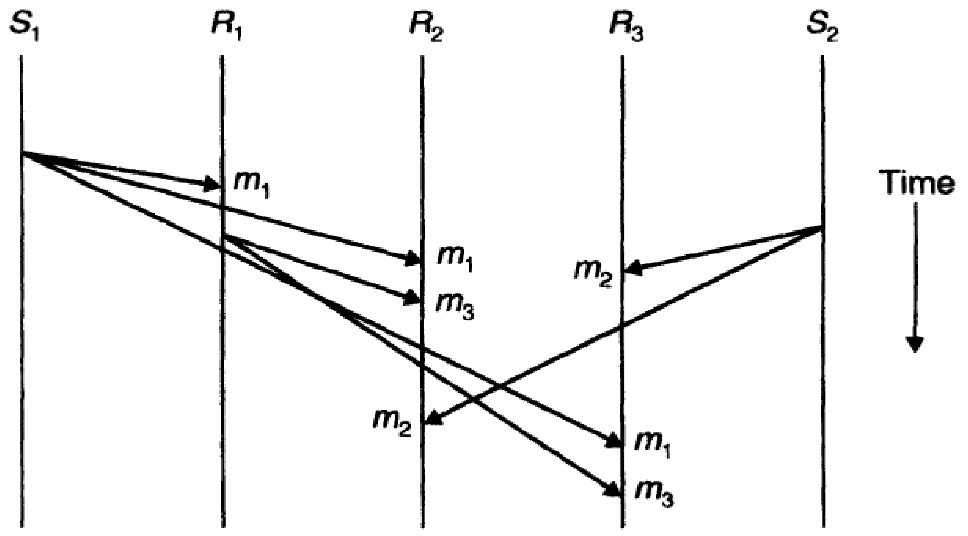
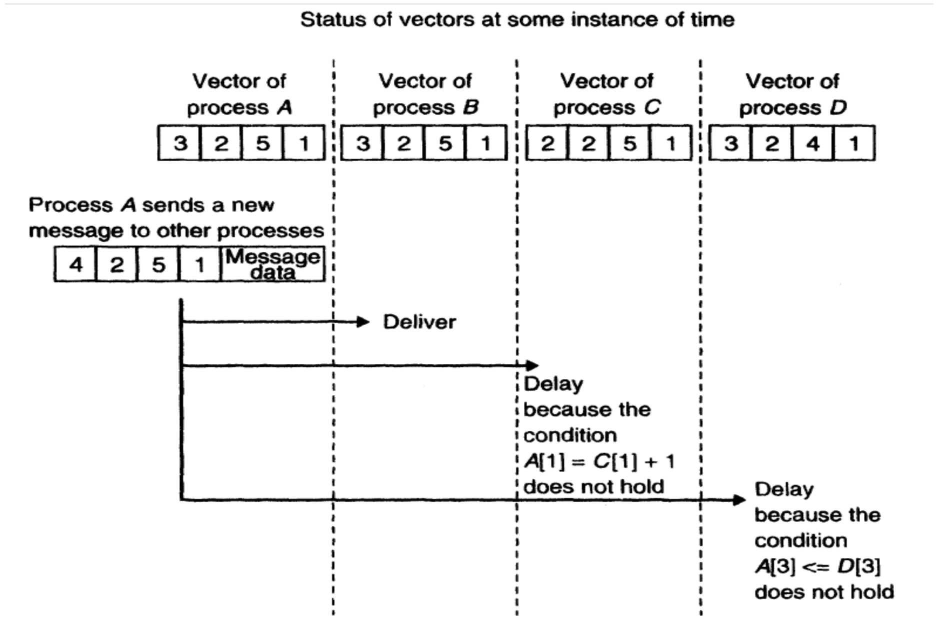


Fig. 3.17 Causal ordering of messages.

One method for implementing causal-ordering semantics is the CBCAST protocol of the ISIS system.


Status of vectors at some instance of time
Vector of process A
3 2 5 1
Vector of process B
3 2 5 1
Vector of process C
2 2 5 1
Vector of process D
3 2 4 1
Process A sends a new message to other processes
4 2 5 1 Message data
Deliver
Delay because the condition A[1] = C[1] + 1 does not hold
Delay because the condition A[3] <= D[3] does not hold

A simple example to illustrate the algorithm is given in Figure 3.18. In this example, there' are four processes *A, B,* C, and *D.* The status of their vectors at some instance of time is (3, 2, 5, I), (3, 2, 5, 1), (2, 2, 5, 1), and (3, 2, 4, 1), respectively. This means that, until now, *A* has sent three messages, *B* has sent two messages, C has sent five messages, and *D* has sent one message to other processes. Now *A* sends a new message to other processes. Therefore, the vector attached to the message will be (4, 2, 5, 1). The message can be delivered to *B* because it passes both tests. However, the message has to be delayed by the runtime systems of sites of processes C and *D* because the first test fails at t he site of process C and the second test fails at the site of process *D.*

A good message-passing system should support at least consistent- and causal ordering semantics and should provide the flexibility to the users to choose one of these in their applications.

Distributed Shared Memory

Two basic paradigms

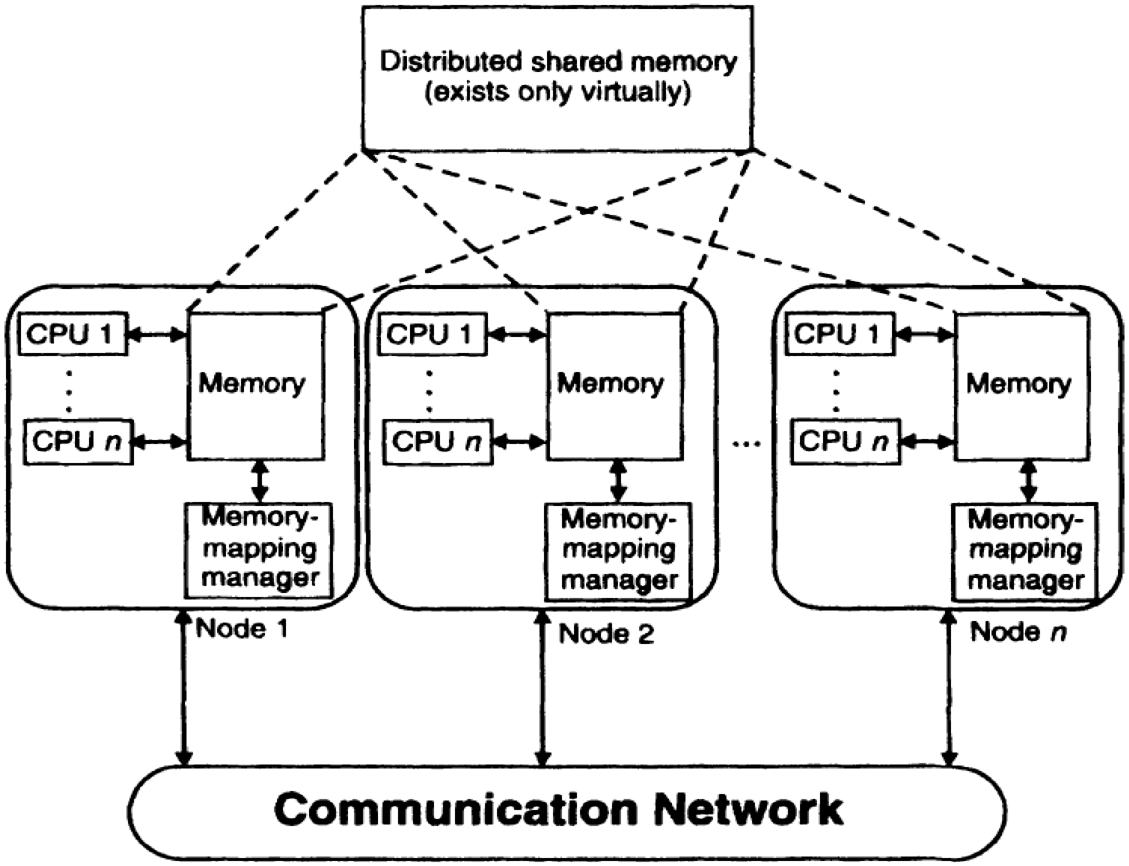
Shared-memory paradigm :

Send *(recipient, data)*

Receive *(data)*

Message-passing paradigm : *data* = Read *(address)*

Write *(address, data)*



DESIGN AND IMPlEMENTAION ISSUES OF DSM

1.Granularity.

2. Structure of shared-memory space.

3. Memory coherence and access synchronization.

4. Data location and access

5. Replacemet strategy

6. Thrashing

7. Heterogeneity

Granularity :

Factors Influencing Block Size Selection

1. Pageing overhead
2. Directory Size
3. Thrashing
4. False Sharing

Structure of Shared-memory space :

1. No Structuring
2. Structuring by data type
3. Structuring as a database

A *consistency model* basically refers to the degree of consistency that has to be maintained for the shared-memory data for the memory to work correctly for a certain set of applications. It is defined as a set of rules that applications must obey if they want the DSM system to.provide the degree of consistency guaranteed by the consistency model.

- Strict Consistency Model
- Sequential Consistency Model
- Causal Consistency Model
- Pipelined Random – Access Memory Consistency Model
- Processor Consistency Model
- Weak Consistency Model
- Release Consistency Model
- Discussion of Consistency Models

Implementing Sequential Consistency Model

We saw above that the most commonly used consistency model in DSM systems is the sequential consistency model. Hence, a description of the commonly used protocols for implementing sequentially consistent D5M systems is presented below. A protocol for implementing a release-consistent DSM system will be presented in the next section. Protocols for implementing the sequential consistency model in a DSM system depend to a great extent on whether the DSM system allows replication and/or migration of shared-memory data blocks.

1. Nonreplicated, nonmigrating blocks (NRNMBs)
2. Nonreplicated, migrating blocks (NRMBs)
3. Replicated, migrating blocks (RMBs)
4. Replicated, nonmigrating blocks (RNMBs)

Nonreplicated, Nonmigrating Blocks

This is the simplest strategy for implementing a sequentially consistent DSM system. In this strategy, each block of the shared memory has a single copy whose location is always fixed. All access requests to a block from any node are sent to the owner node of the block, which has the only copy of the block. On receiving a request from a client node, the memory management unit (MMU) and operating system software of the owner node perform the access request on the block and return a response to the client (Fig. 5.3).

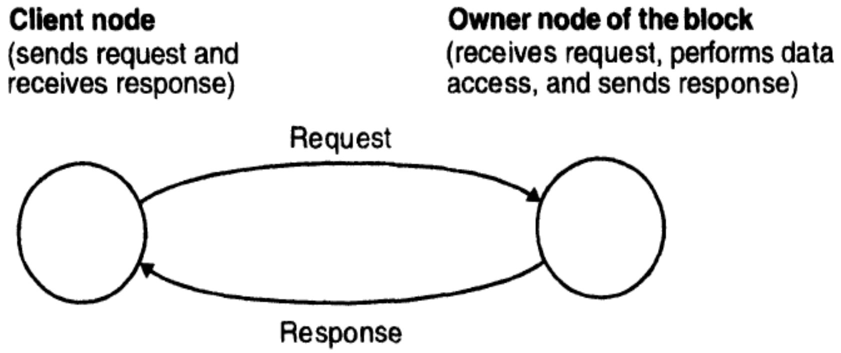

Client node
(sends request and receives response)
Owner node of the block
(receives request, performs data access, and sends response)
Request
Response

Fig. 5.3 Nonreplicated, nonmigrating blocks (NRNMB) strategy.

Although the method is simple and easy to implement, it suffers from the following drawbacks:

- Serializing data access creates a bottleneck.
- Parallelism, which is a major advantage of DSM, is not possible with this method.

Nonreplicated, Migrating Blocks

In this strategy each block of the shared memory has a single copy in the entire system. However, each access to a block causes the block to migrate from its current node to the node from where it is accessed. Therefore, unlike the previous strategy in which the owner node of a block always remains fixed, in this strategy the owner node of a block changes as soon as the block is migrated to a new node (Fig. 5.4).

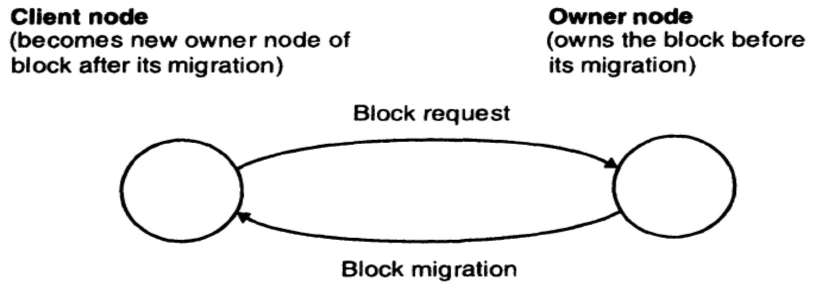

Client node
(becomes new owner node of block after its migration)
Owner node
(owns the block before its migration)
Block request
Block migration

Fig. 5.4 Nonreplicated, migrating blocks (NRMB) strategy.

The method has the following advantages :

1. No communication costs are incurred when a process accesses data currently held locally.
2. It allows the applications to take advantage of data access locality. If an application exhibits high locality of reference, the cost of data migration is amortized over multiple accesses.

However, the method suffers from the following drawbacks:

1. It is prone to thrashing problem. That is, a block may keep migrating frequently from one node to another, resulting in few memory accesses between migrations and thereby poor performance.
2. The advantage of parallelism cannot be availed in this method also.

REPLACEMENT STRATEGY

following issues must be addressed when the available space for caching shared data fills up at a node:

1. Which block should be replaced to make space for a newly required block?
2. Where should the replaced block be placed?

Which Block to Replace

usual classification of replacement algorithms group them into the following categories

1. Usage based versus non-usage based (2algorithms : Least recently used (LRU) / (FIFO))
2. Fixed space versus variable space (5 types)

   * unused - A free memory block that is not currently being used.
   * nil - A block that has been invalidated.
   * Read – only : A block for which the node has only read access right.
   * Read – owned : A block for which the node has only read access right but is also the owner of the block.
   * Writable - A block for which the node has write access permission.

Where to Place a Replaced Block

two commonly used approaches :

I. *Using secondary store.* In this method, the block is simply transferred on to a local disk.

2. Using the memory space of other nodes. Sometimes it may be faster to transfer a block over the network than to transfer it to a local disk. Therefore, another method for storing a useful block is to keep track of free memory space at all nodes in the system and to simply transfer the replaced block to the memory of a node with available space.

Thrashing is said to occur when the system spends a large amount of time transferring shared data blocks from one node to another, compared to the time spent doing the useful work of executing application processes.

Thrashing may occur in the following situations:

1. When interleaved data accesses made by processes on two or more nodes causes a data block to move back and forth from one node to another in quick succession (a ping-pong effect)

2. When blocks with read-only permissions are repeatedly invalidated soon after they are replicated

Such situations indicate poor (node) locality in references. If not properly handled, thrashing degrades system performance considerably. Therefore, steps must be taken to solve this problem. The following methods may be used to solve the thrashing problem in DSM systems:

I. *Providing application-controlled locks.* Locking data to prevent other nodes from accessing that data for a short period of time can reduce thrashing. An applicationcontrolled lock can be associated with each data block to implement this method.

2. Nailing a block to a node for a minimum amount of time. Another method to reduce thrashing is to disallow a block to be taken away from a node until a minimum amount of time *t* elapses after its allocation to that node. The time *t* can either be fixed statically or be tuned dynamically on the basis of access patterns

3. Tailoring the coherence algorithm to the shared-data usage patterns. Thrashing can also be minimized by using different coherence protocols for shared data having different characteristics.

ADVANTAGES OF DSM

Simpler Abstraction it provides to the application programmers of loosely coupled distributed-memory machines.

Better Portability of Distributed Application Programs This allows for a more natural transition from sequential to distributed applications. In principle, distributed application programs written for a shared-memory multiprocessor system can be executed on a distributed shared-memory system without change.

Better Performances of Some Applications

3 reasons

- locality of Data
- On-demand data movement
- Larger memory space

Flexible Communication Environment in *which* the sender process need not specify the identity of the receiver processes of the data. It simply places the data in the shared memory and the receivers access it directly from the shared memory. Therefore, the coexistence of the sender and receiver processes is also not necessary in the shared-memory paradigm.

Ease of Process Migration process control block (PCB) of the migrant process from the processor of the old node and attaching it to the ready queue of the new node's processor. A PCB is a data block or a record associated with each process that contains useful information such as process state, CPU registers, scheduling information, memory management information, I/O status information, and so on.

Synchronization

In systems with multiple concurrent processes , it is economical to share the system resources (hardware or software) among the concurrently executing processes. In such a situation, sharing may be cooperative or competitive.

Both cooperative and competitive sharing require adherence to certain rules of behavior that guarantee that correct interaction occurs . The rules for enforcing correct interaction are implemented in the form of synchronization mechanisms .

synchronization mechanisms that are suitable for distributed systems. In particular, the following synchronization-related issues are described:

- Clock synchronization
- Event ordering
- Mutual exclusion
- Deadlock
- Election algorithms

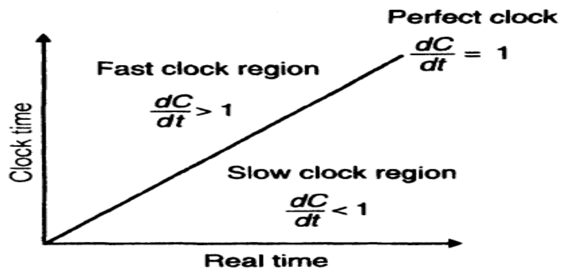
Clock SYNCHRONIZATION

Every computer needs a timer mechanism (called a computer clock) to keep track of current time and also for various accounting purposes such as calculating the time spent by a process in CPU utilization, disk *I/O*, and so on, so that the corresponding user can be charged properly. In a distributed system, an application may have processes that concurrently run on multiple nodes of the system. For correct results, several such distributed applications require that the clocks of the nodes are synchronized with each other.

distributed operating system designer to devise and use suitable algorithms for properly synchronizing the clocks of a distributed system.

How Computer Clocks Are Implemented : A computer clock usually consists of three components-a quartz crystal that oscillates at a well-defined frequency, a *counter* register, and a *constant* register.

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



Types of clock synchronization:

1. *Synchronization of the computer clocks with real-time (or external) clocks.*
2. *Mutual (or internal) synchronization of the clocks of different nodes of the system.*

Clock Synchronization Algorithms

Clock synchronization algorithms

1. centralized Algorithms , one node has a real-time receiver. This node is usually called the *time server node,* and the clock time of this node is regarded as correct and used as the reference time. The goal of the algorithm is to keep the clocks of all other nodes synchronized with the clock time of the time server node.

- Passive Time Server Centralized Algorithm
- Active Time Server Centralized Algorithm

2. Distributed Algorithms , a simple method for clock synchronization may be to equip each node of the system with a real-time receiver so that each node's clock can be independently synchronized with real time.

- Global Averaging Distributed Algorithms
- Localized Averaging Distributed Algorithms

EVENT ORDERING

Keeping the clocks in a distributed system synchronized to within 5 or 10msec is an expensive and nontrivial task.

a new relation called *happened before* and introduced the concept of logical clocks for ordering of events based on the happened-before relation.

two events are concurrent if neither can causally affect the other. Due to this reason, the happened-before relation is sometimes also known as the relation of *causal ordering.*

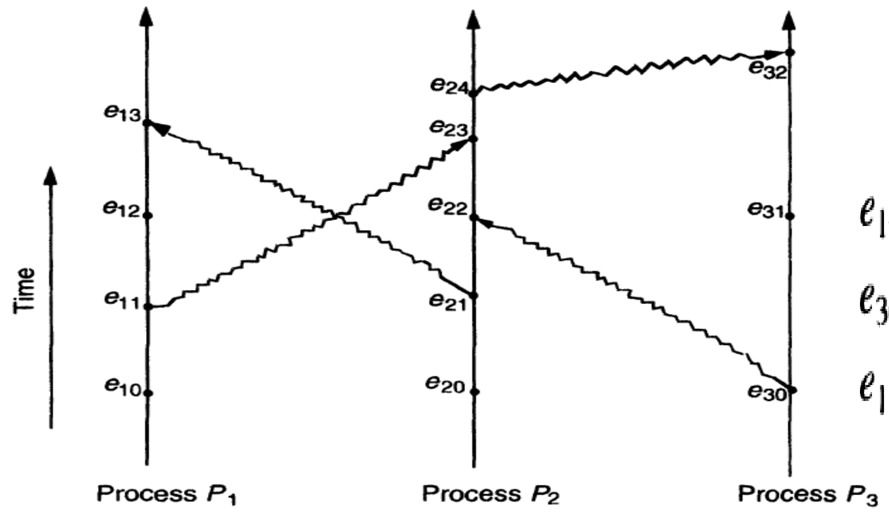


Fig. 6.3 Space-time diagram for three processes.

Some of the events are below :

$e_{10} \rightarrow e_{11}$ $e_{20} \rightarrow e_{24}$ $e_{11} \rightarrow e_{23}$ $e_{21} \rightarrow e_{13}$

$e_{30} \rightarrow e_{24}$ (since $e_{30} \rightarrow e_{22}$ and $e_{22} \rightarrow e_{24}$)

$e_{11} \rightarrow e_{32}$ (since $e_{11} \rightarrow e_{23}$, $e_{23} \rightarrow e_{24}$, and $e_{24} \rightarrow e_{32}$)

1. Logical Clocks
2. Physical Clocks

Total Ordering of Events

MUTUAL EXCLUSION

exclusiveness of access is called *mutual exclusion* between processes, means are introduced to prevent processes from executing concurrently within their associated critical sections.

An algorithm for implementing mutual exclusion must satisfy the following requirements:

1. *Mutual exclusion.*
2. *No starvation.*

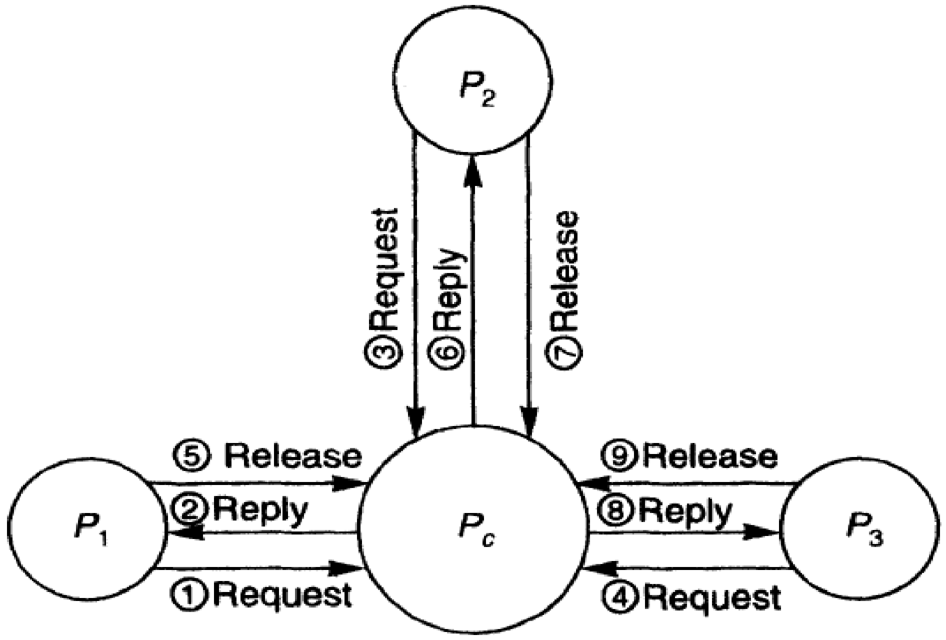
Three basic approaches used by different algorithms for implementing mutual exclusion in distributed systems are described below.

Centralized Approach

Distributed Approach

Token Passing Approach

A *token* is a special type of message that entitles its holder to enter a critical section.



DEADLOCK

Since a system consists of a finite number of units of each resource type (for example, three printers, six tape drives, four disk drives, two CPUs, etc.), multiple concurrent processes normally have to compete to use a resource.

deadlock, that is, a situation in which competing processes prevent their mutual

progress even though no single one requests more resources than are available. It may

happen that some of the processes that entered the waiting state (because the requested resources were not available at the time of request) will never again change state, because the resources they have requested are held by other waiting processes. This situation is called *deadlock*, and the processes involved are said to be *deadlocked.*

Necessary Conditions for Deadlock

Deadlock Modeling

Necessary and Sufficient Conditions for Deadlock

Handling Deadlocks In Distributed Systems

Deadlock Avoidance

Deadlock Prevention

Deadlock Detection

Unit – IV Distributed File Systems

A *file system* is a subsystem of an operating system that performs file management activities such as organization, storing, retrieval, naming, sharing, and protection of files.

A *distributed file system* provides similar abstraction to the users of a distributed system and makes it convenient for them to use files in a distributed environment.

It's support the following :

1. Remote information sharing
2. User mobility
3. Availability
4. Diskless workstations

DFS following 3 types of Services

1. Storage Service
2. True File Service
3. Name Service

Desirable FEATURES OF A GOOD DISTRIBUTED FILE SYSTEM

1. Transparency – 4 types :Structure | Access Transparency | Naming | Replication

2. User Mobility - the flexibility to work on different nodes at different times, to automatically bring a user's environment at the time of login to the node where the user logs in.

3. Performance - is measured as the average amount of time needed to satisfy client requests.

4. Simplicity and ease of use - semantics of the distributed file system understand.

5. Scalability – a scalable design should withstand high service load, accommodate growth of the user community, and enable simple integration of added resources.

6. High Availabilty - continue to function even when partial failures occur due to the failure of one or more components, such as a communication link failure, a machine failure, or a storage device crash.

7. High reliability - to make backup copies of their files of the unreliability of the system.

8. Data Integrity - A file is often shared by multiple users.

9. Security - Necessary security mechanisms must be implemented to protect information stored in a file system against unauthorized access.

10. Heterogeneity - As a consequence of large scale, heterogeneity becomes inevitable in distributed systems. Heterogeneous distributed systems provide the flexibility to their users to use different computer platforms for different applications.

File-ACCESSING MODELS

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

Accessing Remote Files :

1.Remote Service Model - That is, the client's request for file access is delivered to the server, the server machine performs the access request, and finally the result is forwarded back to the client. The access requests from the client and the server replies for the client are transferred across the network as messages.

2. Data – caching model - The client's request is processed on the client's node itself by using the cached data. A replacement policy, such as the least recently used (LRU), is used to keep the cache size bounded.

Unit of Data Transfer : The four commonly used data transfer models based on this factor are as follows:

1. File-level transfer model. In this model, when an operation requires file data to be transferred across the network in either direction between a client and a server, the whole file is moved.

2. Block-level transfer model. In this model, file data transfers across the network between a client and a server take place in units of file blocks.

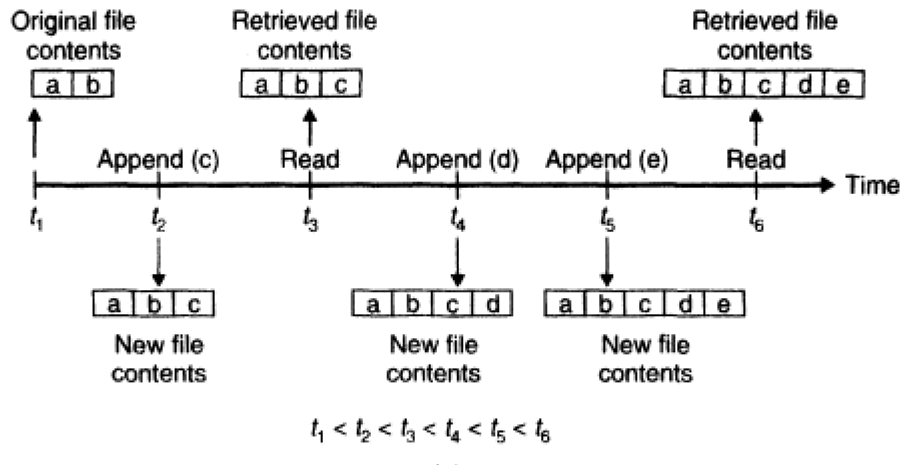
3. Byte-level transfer model. In this model, file data transfers across the network between a client and a server take place in units of bytes.

4. Record-level transfer model. The three file data transfer models described above are commonly used with unstructured file models. The record-level transfer model is suitable for use with those file models in which file contents are structured in the form of records.

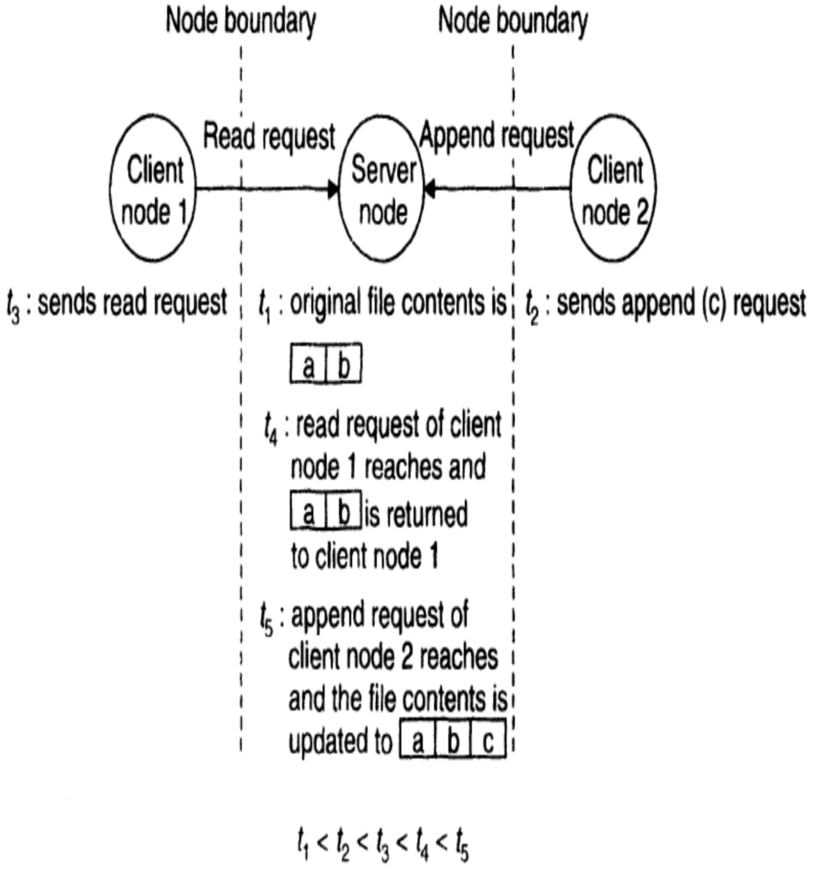
FILE-SHARING SEMANTICS

A shared file may be simultaneously accessed by multiple users. when modifications of file data made by a user are observable by other users. This is defined by the type of file sharing semantics adopted by a file system, defined the following types of file-sharing semantics:

1. UNIX semantics : This semantics enforces an absolute time ordering on all operations and ensures that every read operation on a file sees the effects of all previous write operations performed on that file [Fig. *9.1(a)].*



(a)



Node boundary
Node boundary
Client node 1
Read request
Server node
Append request
Client node 2
t_3 : sends read request
t_1 : original file contents is a b
t_2 : sends append (c) request
t_4 : read request of client node 1 reaches and a b is returned to client node 1
t_5 : append request of client node 2 reaches and the file contents is updated to a b c
$t_1 < t_2 < t_3 < t_4 < t_5$

(b)

2. Session semantics.

3. Immutable shared-files semantics.

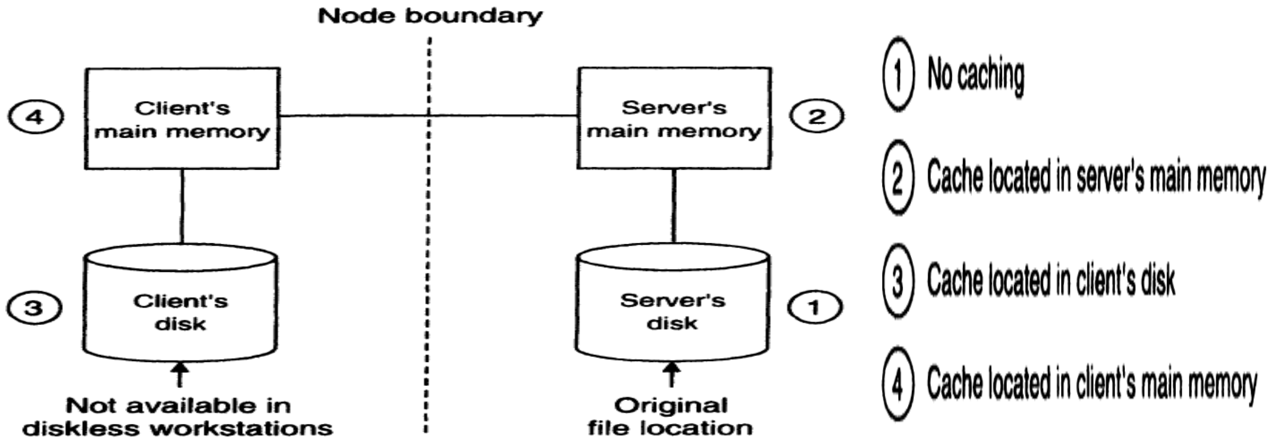
4. Transaction-like semantics.

FILE-CACHING SCHEMES

The idea in file caching in these systems is to retain recently accessed file data in main memory, so that repeated accesses to the same information can be handled without additional disk transfers . In implementing a file-caching scheme for a centralized file system, one has to make several key decisions, such as the granularity of cached data (large versus small), cache size (large versus small, fixed versus dynamically changing), and the replacement policy.

a file-caching scheme for a distributed file system should also address the following key decisions:

1. Cache location : Cache location refers to the place where the cached data is stored. Assuming that the original location of a file is on its server's disk, there are three possible cache locations in a distributed file system (Fig. 9.2).



Node boundary
4
Client's main memory
Server's main memory
2
3
Client's disk
Server's disk
1
Not available in diskless workstations
Original file location
1 No caching
2 Cache located in server's main memory
3 Cache located in client's disk
4 Cache located in client's main memory

Fig. 9.2 Possible cache locations in a file-caching scheme for a distributed file system.

2. Modification propagation : a file's data may simultaneously be cached on multiple nodes. In such a situation, when the caches of all these nodes contain exactly the same copies of the file data, we say that the caches are *consistent.* Write-through Scheme | Delayed – write Scheme : *1. Write on ejection from cache, 2. Periodic write, 3. Write on close.*

3. Cache validation : A file data may simultaneously reside in the cache of multiple nodes. The modification propagation policy only specifies when the master copy of a file at the server node is updated upon modification of a cache entry. Therefore, it becomes necessary to verify if the data cached at a client node is consistent with the master copy. There are basically two approaches to verify the validity of cached data-the client initiated approach and the server-initiated approach

FILE REPLICATION

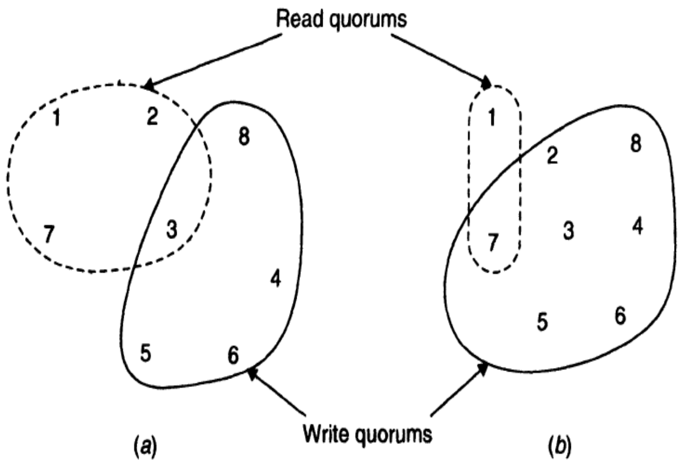
A *replicated file* is a file that has multiple copies, with each copy located on a separate file server. Each copy of the set of copies that comprises a replicated file is referred to as a *replica* of the replicated file.

Difference between Replication and Caching

two concepts have the following basic differences:

1. A replica is associated with a server, whereas a cached copy is normally associated with a client.
2. The existence of a cached copy is primarily dependent on the locality in file access patterns, whereas the existence of a replica normally depends on availability and performance requirements.
3. As compared to a cached copy, a replica is more persistent, widely known, secure, available, complete, and accurate.
4. A cached copy is contingent upon a replica. Only by periodic revalidation with respect to a replica can a cached copy be useful.

a replicated copy from a cached copy by calling them *first-class re* respectively.



ig. 9.4 Examples of quorum consensus algorithm: (*a*) $n = 8$, $r = 4$, $w = 5$; (*b*) $n = 8$, $r = 2$, $w = 7$.

Advantages of Replication

1. I creased availability

2. Increased reliability.

3 Improved response time.

4. Reduced network traffic.

5. Improved system throughput.

6. Better scalability.

7. Autonomous operation.

RaplicatIon Transparency :- Naming of Replicas , Replication Control [Explicit replication, Implicit replica]

- Multicopy Update Problem :- In fact, maintaining consistency among copies when a replicated file is updated is the major design issue of a file system that supports replication of files. Read-Only Replication , Read-Any-Write-All Protocol , Available-Copies Protocol , Primary-Copy Protocol ,**Quorum-Based Protocols(fig. 9.4)**

Fault Tolerance

Fault tolerance is an important issue in the design of a distributed file system. Various types of faults could harm the integrity of the data stored by such a system. For instance, a processor loses the contents of its main memory in the event of a crash.

Distributed file system to tolerate faults are as follows :

1. *Availability.*

2. *Robustness :-* Robustness of a file refers to its power to survive crashes of the storage device and decays of the storage medium on which it is stored.

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

stable-storage technique and the effect of a service paradigm on the fault tolerance of distributed file systems are : *I. Volatile storage, 2. Nonvolatile storage, 3. Stable storage .*

Effect of Service Paradigm on Fault Tolerance : 1.Stateful , 2. Stateless

5 operations : Opern, Read, Write, Seek , Close. **2 operations : Read , Write .**

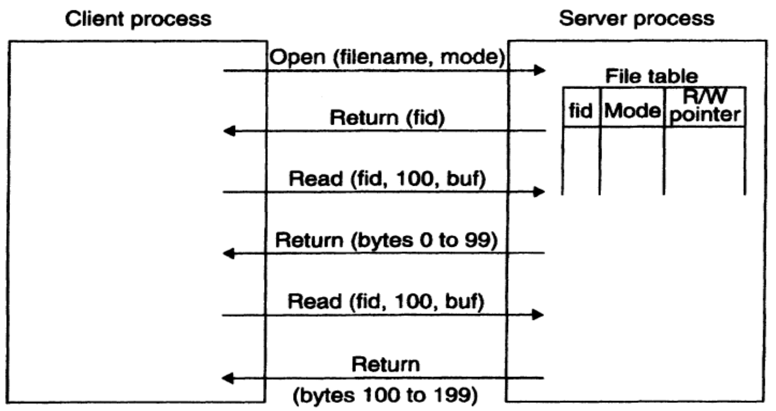


Fig. 9.5 An example of a stateful file server.

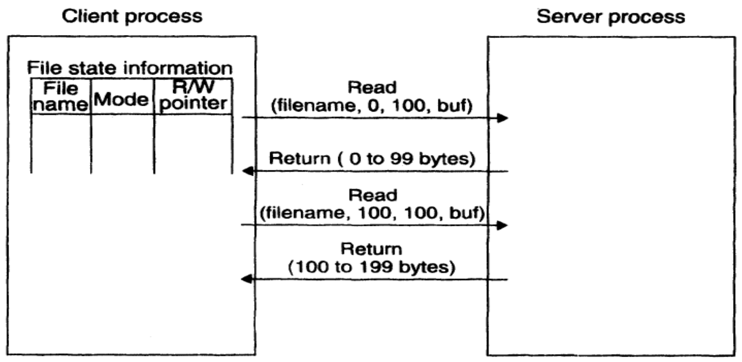


Fig. 9.6 An example of a stateless file server.

An **Atomic transaction** (or just *transaction* for short) is a computation consisting of a collection of operations that take place indivisibly in the presence of failures and concurrent computations

Transactions have the following essential properties:

1. *Atomicity.* This property ensures that to the outside world all the operations of a transaction appear to have been performed indivisibly.
2. *Serializability.* This property (also known as *isolation property)* ensures that concurrently executing transactions do not interfere with each other.

3. *Permanence.* This property (also known as *durability property)* ensures that once a transaction completes successfully, which it is running crashes.

The three essential operations for transaction service are as follows:

1.Begin_transaction : Begins a new transaction and returns a unique transaction identifier (TID). This identifier is used in other operations of this transaction.

2. End_transaction : This operation indicates that, from the

viewpoint of the client, the transaction completed successfully.

3. Abort_transaction : Aborts the transaction, restores any changes made so far within the transaction to the original values, and changes its status to inactive. A transaction is normally aborted in the event of some system failure.

DESIGN PRINCIPLES

The following general principles for designing distributed file systems:

1.Clients have cycles to burn This principle aims at enhancing the scalability of the design, since it lessens the need to increase centralized (commonly used) resources and allows graceful degradation of system performance as the system grows in size.

2. Cache whenever possible Better performance, scalability, user mobility, and site autonomy motivate this principle. Caching of data at clients' sites frequently improves overall system performance because it makes data available wherever it is being currently used, thus saving a large amount of computing time and network bandwidth.

3. Exploit usage properties files should be grouped into a small number of easily identifiable classes, and then class-specific properties should be exploited for independent optimization for improved performance.

4. Minimize system wide knowledge and change. This principle is aimed at enhancing the scalability of design. The larger is a distributed system, t.he more difficult it is to be aware at all times of the entire state of the system and to update distributed or replicated data structures in a consistent manner.

5. Trust the fewest possible entities. This principle is aimed at enhancing the security of the system. For example, it is much simpler to ensure security based on theintegrity of the much smaller number of servers rather than trusting thousands of clients.

6. Batch if possible. Batching often helps in improving performance greatly. For example, grouping operations together can improve throughput, although it is often at the cost of latency. Similarly, transfer of data across the network in large chunks rather than as individual pages is much more efficient.

Unit –V Security

The security goals of a computer system are decided by its security policies, and the methods used to achieve these goals are called security mechanisms.

common goals of computer security are as follows :

1. Secrecy. Information within the system must be accessible only to authorized users.

2. Privacy. Misuse of information must be prevented. That is, a piece of information given to a user should be used only for the purpose for which *it* was given.

3. Authenticity. When a user receives some data, the user must be able to verify its authenticity. That is, the data arrived indeed from its expected sender and not from any other source.

4. Integrity. Information within the system must be protected against accidental destruction or intentional corruption by an unauthorized user.

A total approach to computer security involves both external and internal security.

External security deals with securing the computer system against external factors such as fires, floods, earthquakes, stolen disks/tapes, leaking out of stored information by a person who has access to the information, and so on.

Internal security, on the other hand, mainly deals with the following two aspects:

1. User authentication. Once a user is allowed physical access to the computer facility, the user's identification must be checked by the system before the user can actually use the facility.

2. Access control. A computer system contains many resources and several types of information.

3. Communication security. In a distributed system, the communication channels that are used to connect the computers are normally exposed to attackers who may try to breach the security of the system by observing, modifying, or disrupting the communications.

POTENTIAL ATTACKS TO COMPUTER SYSTEMS

computer security is to identify the potential threats/attacks to computer systems. The term *intruder* or *attacker* is commonly used to refer to a person or program trying to obtain unauthorized access to data or a resource of a computer system. An intruder may be a threat to computer security in many ways that are broadly classified into two categories-passive attacks and active attacks.

Some commonly used methods of **passive attack** are described below:

1. *Browsing.* In this method, intruders attempt to read stored files, message packets passing by on the network, other processes' memory, and so on, without modifying any data.

2. Leaking. Prevention of leaking is a difficult problem to solve and requires preventing all types of communication between the accomplice and the intruder. The problem of ensuring that it is impossible for a potential accomplice to leak any information to the outside world is called the *confinement problem*

3. Inferencing, In this method, an intruder tries to draw some inference by closely observing and analyzing the system's data or the activities carried out by the system.

4. Masquerading. In this method, an intruder masquerades as an authorized user or program in order to gain access to unauthorized data or resources. These programs can improperly use the access rights of an executing user and leak information. This editor program is then compiled and read into the *bin* directory of a user, whose files the intruder is interested in. Penetrating computer security in this manner is known as the *Trojan horse attack.*

An intruder can also masquerade as a trusted server to a client requesting a service from the system. This action is known as ***spoofing.***

Active Attacks

Active intruders are more malicious than passive intruders. active attacks cause are corrupting files, destroying data, imitating hardware errors, slowing down the system, filling up memory or disk space with garbage, causing the system to crash, confounding a receiver into accepting fabricated messages, and denial/delay of message delivery.

Some commonly used forms of active attacks are described below.

Viruses : A *computer virus* is a piece of code attached to a legitimate program that, when executed, infects other programs in the system by replicating and attaching itself to them. In addition to this replicating effect, a virus normally does some other damage to the system, such as corrupting/erasing files.

Worms : Worms are programs that spread from one computer to another in a network of computers. They spread by taking advantage of the way in which resources are shared on a computer network and, in some cases, by exploiting flaws in the standard software installed on network systems. A worm program may perform destructive activities after arrival at a network node.

Logic Bombs : A logic bomb is a program that lies dormant until some trigger condition causes it to explode. On explosion, it destroys data and spoils system software of the host computer. A trigger condition may be an event such as accessing a particular data file, a program being run a certain number of times, the passage of a given amount of time, or the system clock reaching some specific date.

CRYPTOGRAPHY

Cryptography is a means of protecting private information against unauthorized access in those situations where it is difficult to provide physical security.

Basic Concepts and Terminologies :

Two primitive operations employed by cryptography are *encryption* and *decryption.*

Encryption (also called *enciphering)* is the process of transforming an intelligible information (called *plaintext* or *clear text)* into an unintelligible form (called *cipher text).*

Decryption (also called *deciphering)* is the process of transforming the information back from cipher text to plaintext.

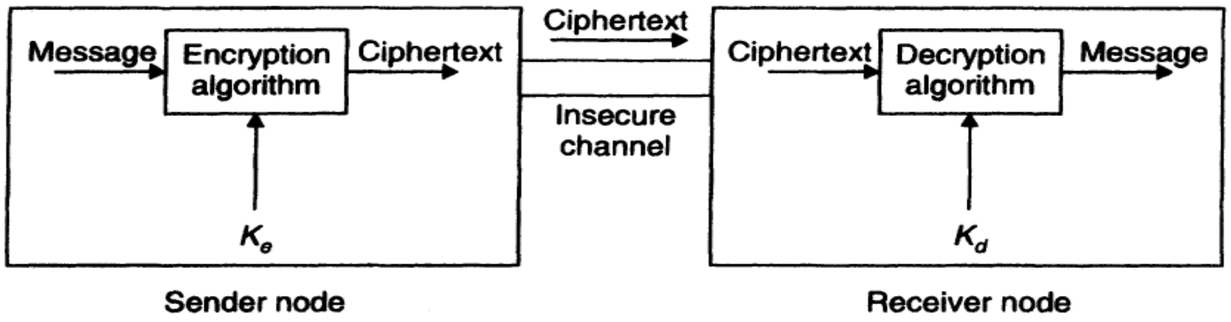


Fig. 11.1 General structure of a cryptosystem.

Encryption is basically a mathematical function (encryption algorithm) having the following form: $\boxed{C = E(P, Ke)}$

where *P* is the plaintext to be encrypted, *K,* is an encryption key, and C is the resulting Cipher text.

Decryption of C is performed by a matching function (decryption algorithm) that has the following form: $\boxed{P = D(C, Kd)}$

where *Kd* is the decryption key. Note that the decryption function *D* is the inverse of the encryption function *E.*

AUTHENTICATION

Authentication deals with the problem of verifying the identity of a user before permitting access to the requested resource. Authentication basically involves identification and verification. *Identification* is the process of claiming a certain identity by a user, while *verification* is the process of verifying the user's claimed identity.

The main types of authentication in a distributed system are as follows:

1. User logins authentication. It verifying the identity of a user by the system at the time of login.

2. One-way authentication of communicating entities. It deals with verifying the identity of one of the two communicating entities by the other entity.

3. Two-way authentication of communicating entities. It deals with mutual authentication, whereby both communicating entities verify each other's identity.

Approaches to Authentication :

1. *Proof by knowledge.* In this approach, authentication involves verifying something that can only be known by an authorized principal.

2. Proof by possession. In this approach, a user proves his or her identity by producing some item that can only be possessed by an authorized principal. The system is designed to verify the produced item to confirm the claimed identity.

3. Proof by property. In this approach, the system is designed to verify the identity of a user by measuring Some physical characteristics of the user that are hard to forge. The measured property must be distinguishing, that is, unique among all possible users.

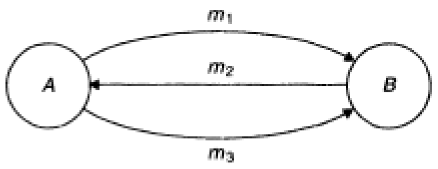
Types of Authentication

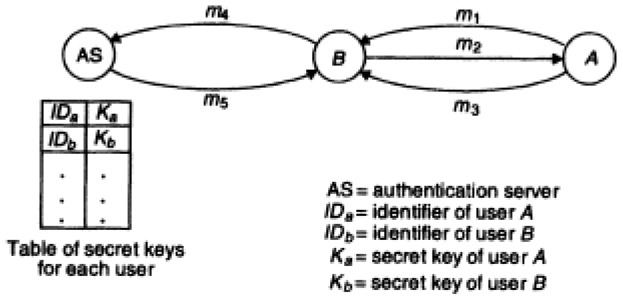
User Login Authentication : based on passwords, the system maintains a table of authorized users' login names and their corresponding passwords. a password-based authentication system must have mechanisms for the following:

1. Keeping passwords secret
2. Making passwords difficult to guess
3. Limiting damages done by a compromised password
4. Identifying and discouraging unauthorized user logins
5. Single sign-on for using all resources in the system

One – Way Authentication of Communicating Entities : protocols can be broadly classified into two categories-those based on symmetric cryptosystems and those based on asymmetric cryptosystems.

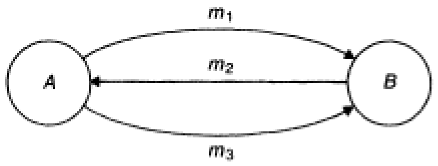
Protocols Based on Symmetric Cryptosystems

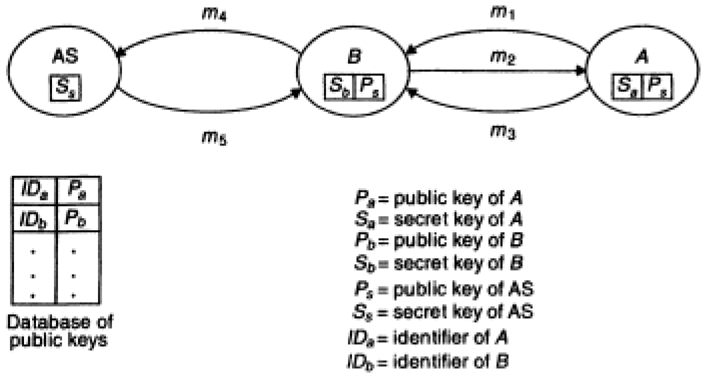




m_4
m_1
m_2
AS
B
A
m_5
m_3
ID_a | K_a
ID_b | K_b
Table of secret keys for each user
AS = authentication server
ID_a = identifier of user A
ID_b = identifier of user B
K_a = secret key of user A
K_b = secret key of user B

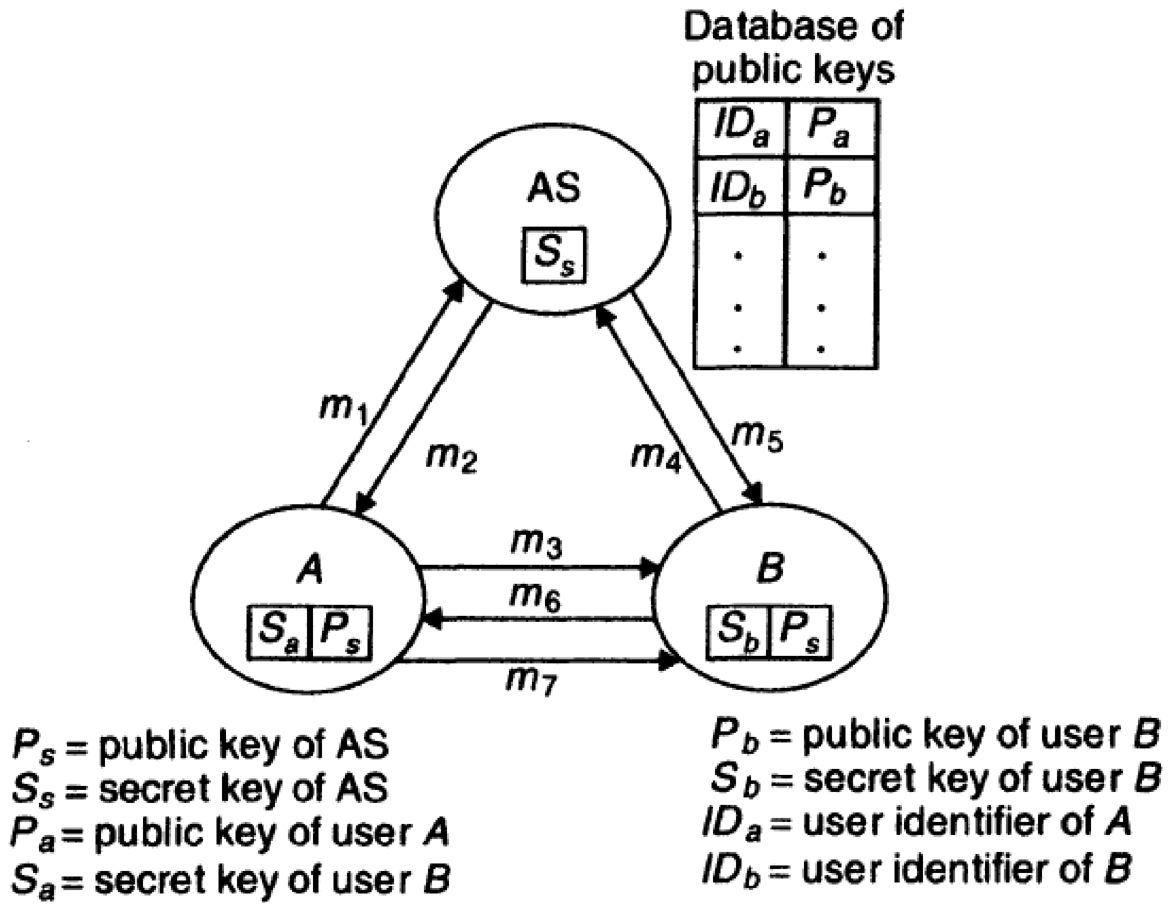
Protocols Based on Asymmetric Cryptosystems





Two-Way Authentication of Communicating Entities

Two-way authentication protocols allow both communicating entities to verify each other's identity before establishing a secure logical communication channel between them.



P_s = public key of AS
S_s = secret key of AS
P_a = public key of user A
S_a = secret key of user B

P_b = public key of user B
S_b = secret key of user B
ID_a = user identifier of A
ID_b = user identifier of B

DIGITAL SIGNATURES

A *digital signature* is basically a code, or a large number, that is unique for each message and to each message originator. It is obtained by first processing the message with a hash function (called a *digest function)* to obtain a small digest dependent on each bit of information in the message and then encrypting the digest by using the originator's secret key.

A protocol based on a digital signature for ensuring message integrity works as follows:

1. A sender *(A)* computes the digest *(D)* of a message *(M).* It then encrypts the digest D by using its secret key *(Sa)* to obtain a ciphertext Cl = *E(D, Sa).* A signed message is then created that consists of the sender's identifier, the message *M* in its plaintext form, and the ciphertext C,. The signed message, which has the form *(IDa'* CJ , *M),* is then sent to a receiver.

2. On receiving the signed message, the receiver decrypts Cl by using the public key of the sender to recover the digest *D.* It then calculates a digest for *M* (by using the same digest function) and compares the calculated digest· with the digest recovered by decrypting Cl' If the two are equal, message M is considered to be correct; otherwise it is considered incorrect

Privacy Enhanced Mail (PEM) scheme, designed for adding privacy to Internet mail applications, is a good example of use of cryptography and digital signature techniques. PEM offers confidentiality, authentication, and message integrity.

The PEM program maintains a database of the secret keys of its local users and the public keys of remote users. Currently, the Rivest-Shamir-Adleman (RSA) algorithm is used to generate the public/secret key pairs for users. PEM provides the following types of facilities:

1. *Confidentiality.* Sending a message in encrypted form so that sensitive information within it cannot be read by an int.ruder.

2. *Message integrity*. Sending a signed message so that the receiver can be ensured that the contents of the message were not changed.

DESIGN PRINCIPlES

1.Least privilege. The principle of least privilege (also known as the need-to-know principle) states that any process should be given only those access rights that enable it to access, at any time, what it needs to accomplish its function and nothing more and nothing less.

2. Fail-safe defaults. Access rights should be acquired by explicit permission only and the default should be no access. This principle requires that access control decisions should be based on why an object should be accessible to a process rather than on why it should not be accessible.

3. Open design. This principle requires that the design of the security mechanisms should not be secret but should be public. It is a mistake on the part of a designer to assume that the intruders will not know how the security mechanism of the system works.

4. Built in to the system. This principle requires that security be designed into the systems at their inception and be built in to the lowest layers of the systems. That is, security should not be treated as an add-on feature because security problems cannot be resolved very effectively by patching the penetration holes detected in an existing system.

5. Check for current authority. This principle requires that every access to every object must be checked using an access control database for authority. This is necessary to have immediate effect of revocation of previously given access rights.

6. Easy granting and revocation of access rights. For greater flexibility, a security system must allow access rights for an object to be granted or revoked dynamically. It should be possible to restrict some of the rights and to grant to a user only those rights that are sufficient to accomplish its functions.

7. Never trust other parties. For producing a secure distributed system, the system components must be designed with the assumption that other parties (people or programs) are not trustworthy until they are demonstrated to be trustworthy. For example, clients and servers must always be designed to view each other with mutual suspicion.
8. Always ensure freshness of messages. To avoid security violations through the replay of messages, the security of a distributed system must be designed to always ensure freshness of messages exchanged between two communicating entities.
9. Build firewalls. To limit the damage in case a system's security is compromised, the system must have firewalls built into it. One way to meet this requirement is to allow only short-lived passwords and keys in the system*10. Efficient,* The security mechanisms used must execute efficiently and be simple
to implement.
11. Convenient to use. To be psychologically acceptable, the security mechanisms must be convenient to use. Otherwise, they are likely to be bypassed or incorrectly used by the users.
12. Cost effective. It is often the case that security needs to be traded off with other goals of the system, such as performance or ease of use. Therefore, in designing the security of a system, it is important to come up with the right set of trade-offs that take into account the likelihood that the system will be compromised with the cost of providing the security, both in terms of money and personnel experience.