ALAGAPPA UNIVERSITY

[Accredited with 'A+' Grade by NAAC (CGPA:3.64) in the Third Cycle
and Graded as Category–I University by MHRD-UGC]

(A State University Established by the Government of Tamil Nadu)

KARAIKUDI – 630 003

Directorate of Distance Education

M.Sc. [Computer Science]

II - Semester

341 22

DISTRIBUTED OPERATING SYSTEMS

Authors

Dr Syed Mohsin Saif Andrabi, *Assistant Professor, Islamic University of Science & Technology, Awantipora, Jammu and Kashmir*

Dr Mudasir M Kirmani, *Assistant Professor-cum-Junior Scientist, Sher-e-Kashmir University of Sciences and Technology of Kashmir*
Units (1, 4, 5, 6.2-6.3, 7, 8, 10.0-10.1, 10.4-10.10)

Rohit Khurana, *CEO, ITL Education Solutions Ltd.*
Units (3.0-3.2, 14.0-14.2, 14.4-14.9)

V.K. Govindan, *Professor, Computer Engineering, Department of Computer Science and Engineering, NIT, Calicut*
Unit (13)

Vikas® Publishing House, Units (2, 3.3-3.8, 6.0-6.1, 6.4-6.9, 9, 10.2-10.3, 11, 12, 14.3)

VIKAS® PUBLISHING HOUSE PVT. LTD.
E-28, Sector-8, Noida - 201301 (UP)
Phone: 0120-4078900 • Fax: 0120-4078999
Regd. Office: 7361, Ravindra Mansion, Ram Nagar, New Delhi 110 055
• Website: www.vikaspublishing.com • Email: helpline@vikaspublishing.com

Work Order No. AU/DDE/DE1-291/Preparation and Printing of Course Materials/2018 Dated 19.11.2018 Copies - 500

SYLLABI-BOOK MAPPING TABLE

Distributed Operating Systems

Syllabi	Mapping in Book

BLOCK 1: FUNDAMENTALS
Unit - 1: Introduction: What is Distributed Operating System, Evolutions, Models
Unit - 2: Issues in Designing Distributed Computing System
Unit - 3: Introduction to Computer Networks: Network Types, LAN, WAN, Communication Protocols, Internetworking, ATM Technology
	Unit 1: Introduction to Distributed Operating System **(Pages 1-8);**
	Unit 2: Issues in Designing Distributed Computing System **(Pages 9-16);**
	Unit 3: Introduction to Computer Networks **(Pages 17-34)**

BLOCK 2 : MESSAGE PASSING
Unit - 4: Introduction: Features, Issued in PC Message Passing, Synchronization
Unit - 5: Buffering, Multidatagram Messages, Encoding and Decoding
Unit - 6: Process Addressing, Failure Handling, Group Communication
	Unit 4: Introduction to Message Passing **(Pages 35-46);**
	Unit 5: Buffering and Multidatagram Messages **(Pages 47-52);**
	Unit 6: Process Addressing and Failure Handling **(Pages 53-70)**

BLOCK 3 : DISTRIBUTED SHARED MEMORY
Unit - 7: Introduction: General Architecture of DSM System, Design and Implementation Issues of DSM, Granularity, Structure of Shared Memory Consistency Models, Replacement Strategy, Thrasing
Unit - 8: Other Approaches to DSM, Heterogeneous DSM, Advantages
Unit - 9: Synchronization: Introduction, Clock Synchronization, Event Ordering, Mutual Exclusion, Deadlock, Election Algorithm
	Unit 7: Introduction to DSM **(Pages 71-83);**
	Unit 8: Approaches to DSM **(Pages 84-89);**
	Unit 9: Synchronization **(Pages 90-113)**

BLOCK 4 : DISTRIBUTED FILE SYSTEM
Unit - 10: Introduction: Desirable Features, File Modes, File Accessing Models
Unit - 11: File Sharing Semantics, File Caching Schemes, File Replication
Unit - 12: Fault Tolerence, Atomic Transaction, Design Principles
	Unit 10: Introduction to DFS **(Pages 114-123);**
	Unit 11: File Sharing and Replication **(Pages 124-143);**
	Unit 12: Fault Tolerance and Transaction in DFS **(Pages 144-159)**

BLOCK 5 : SECURITY **Unit - 13:** Introduction: Potential Attacks to Computer System, Cryptography, Authentication **Unit - 14:** Access Control, Digital Signatures, Design Principles	**Unit 13:** Introduction to Security **(Pages 160-174)** **Unit 14:** Access Control and Design Principles **(Pages 175-184)**

CONTENTS

INTRODUCTION

NOTES

Distributed Operating System (DOS) is an operating system which manages a set of independent computer systems, and makes these computer systems appear as a single unit to the user of a system. In DOS, two types of hardware architectures—parallel and distributed—can be used. Parallel architecture is a hardware architecture in which there are multiple processors that are tightly coupled and have a shared memory. Distributed hardware architecture is a hardware architecture in which there are multiple loosely coupled computer systems, each system with its own memory. Software architectures which can be used for DOS, include multiprocessor Operating System (OS), network OS and distributed OS. Multiprocessor OS is an operating system in which communication between processors takes place through shared memory and a single-run queue. Network OS is an operating system in which communication takes place using shared files and n-run queues. Distributed OS is an operating system in which communication takes place using messages and N-run queues.

DOS has a number of advantages over other network operating systems. It provides an easy interface required to obtain services from different systems of the network. This interface is implemented below the application program that enables a user to use the existing systems easily. Another key advantage of DOS is its communication techniques which are used to transfer messages from one system to another. In DOS, every computer system has an operating system. DOS provides an interface that a program can use to obtain services, such as input and output services. DOS allows a computer in the distributed system to execute a program and access data from another computer system.

This book, *Distributed Operating Systems*, follows the self-instruction mode or the SIM format wherein each unit begins with an 'Introduction' to the topic followed by an outline of the 'Objectives'. The content is presented in a simple and structured form interspersed with 'Check Your Progress' questions for better understanding. At the end of the each unit a list of 'Key Words' is provided along with a 'Summary' and a set of 'Self Assessment Questions and Exercises' for effective recapitulation.

BLOCK - I
FUNDAMENTALS

NOTES

UNIT 1 INTRODUCTION TO DISTRIBUTED OPERATING SYSTEM

1.0 INTRODUCTION

In the last few decades, you have seen a rapid development in technology which has resulted in facilitation of different tasks performed by a common man in their daily lives. The cost of technology has become affordable to a common man which has results in acceptance of technology by masses. The connectivity of computers to internet has enabled to explore the whole world virtually. However, the connectivity to internet involves different technologies working in tandem in order to make the communication between any two devices possible. The computers are connected with local networks which are further connected to network of networks in order to make the communication between any two computers a reality. The computers also known as nodes in which distance doesn't matter but the connectivity between the two nodes is very important in order to establish a connection.

A computer is an electronic device which accepts inputs and processes as per the requirement and at the end gives the output to the end-user. A computer does not understand the language of a human therefore operating system works as a interface between a computer and a user in order to use the computer for improving the effectiveness and efficiency of different tasks performed. An operating system is a program that acts as an intermediary between the user of a computer and the computer hardware. The purpose of an operating system is

NOTES

to provide an environment in which a user can execute programs in a convenient and efficient manner. The operating system must ensure the correct operation of the computer system and should prevent user programs from interfering with the proper operation of the system by providing appropriate mechanisms. The operating system provides certain services to programs and to the users of those programs in order to make their tasks easier. The services differ from one operating system to another. An operating system is a program that manages the computer hardware.

Operating Systems are of different types and some of them are:

Single User OS: An operating system designed for use by a single individual, for example MS-DOS (Microsoft Disk Operating System) is a single user operating system.

Multi User OS: An operating system that can be used by more than one person. Multi User operating system can be accessed simultaneously by several people through communications facilities or via network terminals. Windows operating system is one of the examples of multi user operating system.

Distributed OS: A form of information processing in which work is performed by separate computers linked through a communications network. Distributed operating system is usually categorized as either plain distributed processing or true distributed processing. The distributed operating system has been discussed in detail, later in this unit.

1.1 OBJECTIVES

After going through this unit, you will be able to:

- Discuss the concept of distributed operating system
- Understand the evolution of distributed OS
- Explain the various models of distributed OS

1.2 DISTRIBUTED OPERATING SYSTEM

The distributed operating system consists of different computers which are connected with a network within a virtual shell. A user interacts with the virtual shell in order to perform different tasks as and when needs arises but the distributed operating system architecture delegates the responsibility of performing a task to one or more computers within the virtual shell.

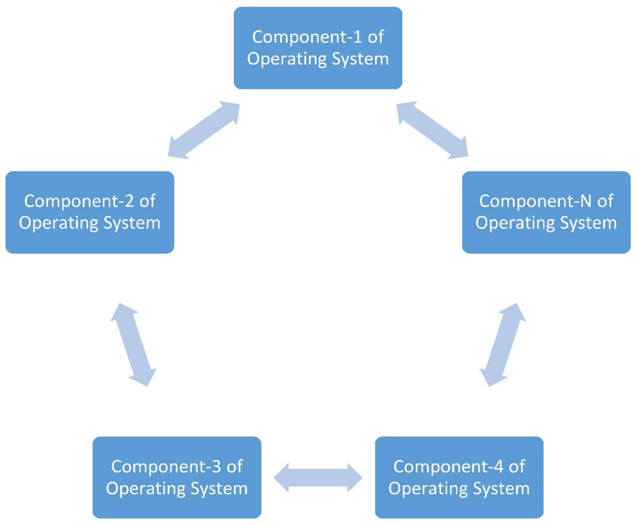

Component-1 of Operating System
Component-2 of Operating System
Component-N of Operating System
Component-3 of Operating System
Component-4 of Operating System

Fig. 1.1 *View of Distributed Operating System*

The dotted circle (boundary) in the Figure 1.1 gives an idea about the user's view of distributed operating system and the internal details are not visible to a user. However, the developer's view of the distributed operating system will be component available within the dotted circle. The Figure 1.1 shows the availability of components from 1 up to N which can vary from one scenario to other.

The components within a distributed operating system can be computers of different operating system which are located at different sites for execution. All the components are connected using a strong local area network as the same plays a vital role in execution of a task within a distributed architecture.

The operating system generally provides different functions like process management, input/ output management, memory management, file organization, security & protection and network management. The distributed operating system has different components which are responsible from different functions of an operating system and all these components are connected with a network. Whenever a user submits a command to a distributed operating system, the instruction may require the services of one or more than one components of an operating system. The user gets a feel as if the whole system is a single unit but internally the whole system consists of sub systems which work in tandem in order to achieve the centralized objective of a distributed operating system.

Plain distributed processing shares the workload among computers that can communicate with one another. True distributed processing has separate computers to perform different tasks in such a way that their combined work can contribute to a goal. The latter type of processing requires a highly structured environment that allows hardware and software to communicate, share resources, and exchange information freely.

1.3 EVOLUTION OF DISTRIBUTED OPERATING SYSTEM

The computers were having very less computing power in the early days of computing when the computers were introduced. The pace of execution along with processing power was increasing many-fold with the advances in technology. Nowadays, minicomputers were used to process different volumes of data. With the advancement in technology and increase in computer speed along with the processing power of a processor, more complex processing along with huge volumes of data was performed on personal computers. However, data analysis where the volume of data was mammoth was performed on main-frame computers. The main-frame computers were not appropriate options for processing of different task in some case which led researchers in exploring option to find different alternatives and one of the alternatives provided by the research community was distributed operating system architecture. The distributed architecture was about delegating the responsibility among one or more computers connected together within a network. This feature of distributed architecture allowed an end user to get higher speed and better processing power. The architecture helped researchers to increase the processing power to manifold by integrating the resources of different independent computers as a single virtual computer. The distributed architecture is not only about hardware rather it is amalgamation of hardware utilization along with appropriate software which help the system to perform with efficiency and effectiveness.

The distributed architecture was further enhanced with the advent of RPC remote procedure calls. The RPCs gained prominence due to the availability of TCP/IP protocol which is known as Transmission Control Protocol/ Internet Protocol. The RPCs helps an individual to execute different commands and instructions on any computer remotely provided the computer is part of a network which a user has permissions to access. The feature has led to more prominence and relevance to the need of distributed operating system architectures in providing higher processing power to an end-user.

Initially the architectural framework was designed to perform batch processing which was later on improved with the inception of minicomputers. The mini computers enabled the researchers to incorporate the concept of master/ slave where a computer was nominated as master and other connected nodes were treated as slaves. The complete execution of a process from inception till maturity was monitored by master and all the slaves will execute the orders of the master as per the requirement of a process or task. As the size of computers was reduced dramatically and with the increase in the processing power of microcomputers the processing within a distributed operating system become more easy, effective and efficient. Nowadays the networks that allow user to connect to internet has reached newer heights in terms of performance, scalability and security which has resulted in leveraging from the advancements in networks by incorporating the high end servers to help in distributed computing with the help of strong, reliable and secure networks.

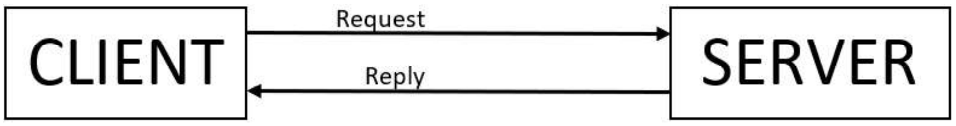
1.4 MODELS OF DISTRIBUTED OPERATING SYSTEM

The models in distributed operating system can also be termed as architectural models. These models give a brief idea about the connection of different components within a distributing operating system of distributed computing. The information about the architectures helps the researchers in devising policies and procedures to make these systems more scalable, robust, effective and efficient. The different architectural models of distributed operating system are listed below:

1. Interaction Model
2. Failure Model
3. Security Model

Interaction model

In the interaction model, the issues related to the interaction of process are included like timing of events. An example of an interaction model can be a client-server architecture where a client submits a request to the server and the server responds with a reply.



In a distributed operating system environment, all the clients need to interact with the server in order to process a request. The server in turn manages and monitors the processing within all the components of a distributed operating system connected with a network. However, the communication between the client and the server result in different issues which need to be addressed by the developers appropriately.

The interaction model can be categorized into two main categories which are given below:

(i) Synchronous Model

(ii) Asynchronous Model

The Failure Model

Failure Model uses the specifications of different faults that occur during the processing of a process and while communicating between the different components of a process. The failure model handles different situation where failures do occur provided the system is able to identify and detect a failure like Omission failure, Byzantine (Arbitrary) Failure and timing failures etc.

NOTES

The Security Model

The Security Model uses the specifications of different threats that occur during the processing of a process and while using the communication channels which play a pivotal part in execution of any process in a distributed operating system. Security threats in a distributed operating system are generally attacks that are intended to break the communication channel or to make a communication process unsuccessful which will result in failure of a distributed operating system.

Some of the commonly occurred threats are list below:

- Unauthorized connection to network
- Identity Threat
- Denial of Service
- Modification/ Deletion of message in a communication channel
- Trojan Horse
- Virus
- Worm
- Server overloading

In order to make any system secure the recommendation is always to use pro-active approach which is similar to the approach of "Prevention is better than cure" where the impact of threats can be mitigated by properly monitoring the processing of tasks and by taking preventive measuring for mitigating the impact of threats.

Check Your Progress

1. What do you understand by distributed OS?
2. What are the two categories of interaction model?

1.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Distributed OS is a form of information processing in which work is performed by separate computers linked through a communications network.
2. Interaction model is categorized into synchronous and asynchronous model.

NOTES

1.6 SUMMARY

- The distributed operating system consists of different computers which are connected within a network within a virtual shell. A user interacts with the virtual shell in order to perform different tasks as and when needs arises.
- The components within a distributed operating system can be computers of different operating system which are located at different sites for execution.
- The models in distributed operating system can also be termed as architectural models. These models give a brief idea about the connection of different components within a distributing operating system of distributed computing.
- In the interaction model, the issues related to the interaction of process are included like timing of events.
- Failure Model uses the specifications of different faults that occur during the processing of a process and while communicating between the different components of a process.
- The Security Model uses the specifications of different threats that occur during the processing of a process and while using the communication channels which play a pivotal part in execution of any process in a distributed operating system.

1.7 KEY WORDS

- **Distributed system:** A collection of hardware and software components which are connected through a network and distributed system layer (middleware) which allows these components to communicate and coordinate with each other and share the resources in such a way that it appears to its user as a single computing facility.
- **Process:** A program under execution or we can say an executing set of machine instructions.

1.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. Write a short note on distributed operating system.
2. What are the various functions of OS?
3. Discuss the evolution of distributed OS.

Long Answer Questions

1. Explain the concept of distributed OS with the help of a diagram.
2. What are the various models of distributed operating system? Explain.

1.9 FURTHER READINGS

Tanenbaum, Andrew S. and Maarten Van Steen. 2006. *Distributed Systems: Principles and Paradigms*. New Jersey: Prentice Hall.

Garg, Vijay K. 2002. *Elements of Distributed Computing*. New Jersey: Wiley-IEEE Press.

Sinha, Pradeep K. 1996. *Distributed Operating Systems: Concepts and Design*. New Delhi: Prentice-Hall of India.

UNIT 2 ISSUES IN DESIGNING DISTRIBUTED COMPUTING SYSTEM

2.0 INTRODUCTION

In the previous unit, you have learnt that a distributed system consists of a large number of CPUs connected to a high-speed network. An important feature of distributed systems is the linkage between multiple processors for processing data. The processors can be of different types such as bus-based multiprocessor and switched multiprocessor. In this unit, you will learn about the various design issues such as transparency, flexibility and performance, of distributed systems.

2.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain the various design issues of distributed systems
- Discuss the various types of transparencies
- Understand the bottlenecks in developing a large system

2.2 DESIGN ISSUES

The design issues in a distributed system help engineers in developing the distributed system effectively. The various design issues in the development of distributed systems are stated as follows:

- Transparency
- Flexibility
- Reliability
- Performance
- Scalability

NOTES

Transparency

A distributed system is said to be transparent if the users of the system feel that the collection of machines is a timesharing system and belongs entirely to him. Transparency can be achieved at two different levels. In the first level, the distribution of the system is hidden from the users: for example, in UNIX, when a user compiles his program using the make command, compilation takes place in parallel on different machines, which use different file systems. The whole system can be made to look like a single-processor system.

In the other level, the system is made to look transparent to the programs. The system call interface can be designed in such a way that the existence of multiple processors can be hidden. This process is more difficult than the first process. There are various types of transparency in the distributed system, which are stated as follows:

- **Location transparency**: It implies that in a true distributed system, the user cannot tell the location of hardware and software resources, such as CPU, printer, database and files: for example, if a user wants to change his address in the database, then he can update the database with his new address. But while doing so, he does not need to know where the database is stored.
- **Migration transparency**: It implies that the resources should move from one location to another without changing their names: for example, consider that there is a file with the hierarchy fun/games. An user decides that playing games is fun and changes the file from fun/games to games/fun. Then if some other client will boot the system, he will not see the file as fun/games but as games/fun.
- **Replication transparency**: It means that the OS in a distributed system has the authorization to create additional copies of files and resources without involving the user. It means that the user does not know about the number of the copies of the files that exist: for example, consider a collection of servers, which are logically connected to each other to form a ring. Each server maintains the directory tree structure of files. If a client sends a request to read a file to any of the servers, it will reply only if it contains the file; otherwise, it forwards the request to the next server. The next server repeats the same process. In replication transparency, servers can make multiple copies of files, which are heavily used.
- **Concurrency transparency**: It suggests that multiple users can use the resources automatically. The problem arises when two or more users try to access the same resource or update the same file concurrently. In such a situation, the system locks the resource or the file if someone else starts accessing it. The lock is released only after the user has finished accessing the resource or the file.

- **Parallelism transparency**: It refers to the parallel execution of activities without the user's knowledge: for example, if the user wants to evaluate the boards of the chess program, multiple situations have to be evaluated in parallel. After evaluating the results, they all send the result to the system and the user can see the result on the screen.

Flexibility

Flexibility in distributed systems is important because this system is new for engineers, and thus there may be false starts and it might be required to backtrack the system. The design issues might prove wrong in the later stages of development. There are two different schemes for building distributed systems. The first one, called monolithic kernel, states that each machine should run a traditional kernel, which provides most of the services itself. Figure 2.1 shows a monolithic kernel.

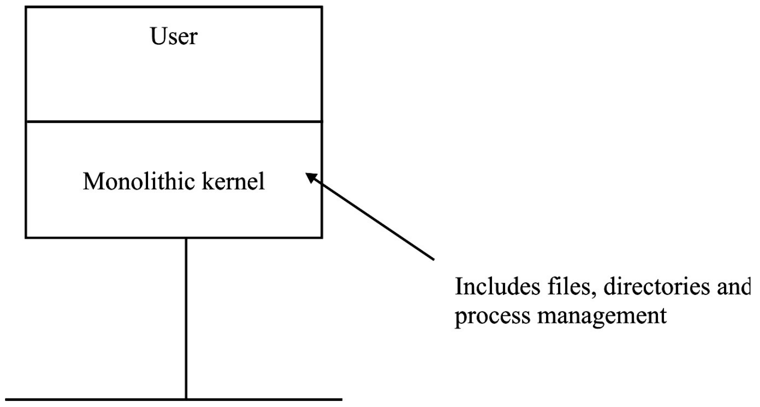


Fig. 2.1 *Monolithic Kernel*

Monolithic kernel is the centralized OS, which has networking ability and remote services. The system calls are made by locking the kernel, then the desired task is performed and the kernel is released after returning the result to the user. In this approach, the machines have their own disks and maintain their own local file system.

The other one, called microkernel, states that the kernel should provide very little services and most of the OS services should be provided from the user-level servers. Figure 2.2 shows a microkernel.

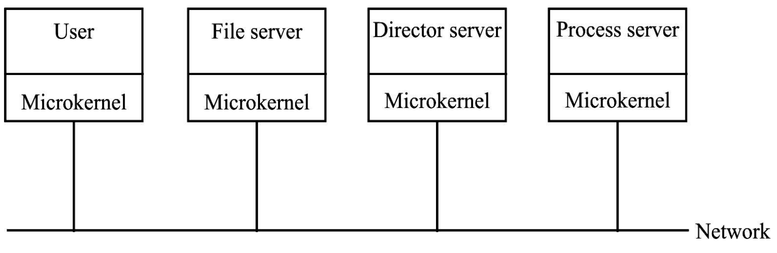


Fig. 2.2 *Microkernel*

Most of the distributed systems are designed to use microkernel because it performs very few tasks. As a result, these systems are more flexible. The services provided by the microkernel are stated as follows:

- It provides interprocess communication.
- It manages the memory.
- It performs low-level process management and scheduling.
- It also performs low-level I/O.

These services are provided by the microkernel because it is expensive for the user-level servers to provide these services. However, it does not provide the file system, directory system, process management and system calls. If the user wants to search for a name, he sends the request to the server, which returns the result after searching. The advantages of this system are stated as follows:

- It is easy to install, implement and debug.
- There is a well-defined interface for each service.
- Each service is equally accessible to all the clients.
- It is more flexible as users have the ability to add, delete and modify the services because it does not involve stopping or booting the kernel.
- The users can write their own services.

The microkernel is more powerful as compared to the monolithic kernel because there can be multiple file servers in the distributed system in which one supports MS-DOS and the other one uses the UNIX file system. Individual users can decide to use either of them or both in the microkernel, whereas in the monolithic kernel, the users do not have any choice. The disadvantage of using the microkernel is its poor performance. The monolithic kernel is faster as it locks the kernel to perform the task rather than sending the request to the server to perform the task as in the case of the microkernel.

Reliability

Distributed systems are more reliable than single-processor systems because if one system in a distributed system stops functioning, other systems can take over. Some other machine can take up the job of the machine that is not working. There are various issues related to reliability, which are stated as follows:

- **Availability**: It refers to the fraction of time during which the system is usable. The availability of a system can be ensured with the design, which does not require simultaneous use of key components. In other words, the components, which are required very often, should not be used concurrently. The resources or files, which are used frequently, can be replicated. If any one of them is occupied by one process or fails, the other processes do not have to wait indefinitely. The data, which is required frequently, can be stored on multiple servers for quick access. But it must be ensured that all the copies are consistent with each other. If one file is updated, all the other copies should also be updated.

- **Security**: It implies that anyone can access the data stored on a distributed system. As a result, it should be protected from unauthorized access. This problem also persists in single-processor systems. But in single-processor systems, the users are required to log in, and thus they are authenticated and the system can check the permission of the user. But in a distributed system, the system has no provision for determining the user and his permission. Anyone can send the request for any file or data in the distributed system. As a result, the issue of security has to be kept in mind while designing the distributed systems.
- **Fault tolerance**: It refers to masking of failure of one of the servers from the users. Suppose while processing some task, one of the servers crashes and quickly reboots. Then, the user will not be able to know what has happened and the data will be lost. Distributed systems should be designed in such a way that even if one of the servers crashes, another server takes up the job and continues evaluating it. If there is cooperation between the servers, the user will not be able to know that the server has crashed and his work will also be completed, though with a little degradation in the performance.

Performance

Building a system, which is flexible and reliable, is of no use if the system is slower than a single-processor system. To measure the performance of the system, various performance metrics are used, such as number of jobs per hour, system utilization and amount of network capacity consumed. The result of any standard used to measure the performance of the system is dependent on the nature of the standard. A standard, which involves a large number of CPU-related computations, will give results different from that of which involves scanning and searching a file for some pattern. In a distributed system, performance is measured by the communication between two users. Sending a message from one system to another and getting a reply does not take much time. Time is wasted in waiting for the protocols, which are used to handle the messages. To increase the performance of the system, it is required to run as many activities as possible at a time. But this requires sending a large number of messages. To solve this problem, it is important to analyse the grain size of all the computations. First of all, small computations, such as addition of two numbers, has to be performed and then complex computations should be performed. This is because large complex computations require more CPU cycles as compared to simple computations. There are some processes in distributed systems, which require simple computations but involve high interaction with one another. These processes are said to have fine-grained parallelism. On the other hand, processes, which involve large computations and require low interaction with one another, are said to exhibit coarse-grained parallelism. Processes, which have coarse-grained parallelism, are preferred over processes having fine-grained parallelism. Better reliability can be achieved when the servers cooperate on a single request: for example, when the request arrives at the server, it can send a copy of the message to the other server for reliability. If the first server crashes before completing the task, the other server can continue the task. When the task is completed, the server has to inform the first server that the task has been completed. This provides more reliability to the system but involves extra messages across the network, which does not produce any output.

Scalability

Distributed systems are designed to work with a few hundred CPUs. There may be situations in future when the systems are much bigger than the systems, which are presently used. As a result, the solution, which works well, may not work well for the system containing very large number of CPUs: for example, consider that the postal, telephone and telegraph administration decides to install a terminal in every house and business in its area. The terminal will allow the people to access the online database containing all the telephone numbers in the area. Thus, there is no need for printing and distributing telephone directory. When all the terminals are placed, they can also be used for e-mails. Using this system, the users can access all the databases and services, such as electronic banking and ticket reservation. There are certain bottlenecks in developing such a large system, which are stated as follows:

- **Centralized components**: There should not be any centralized components in the systems: for example, if there is a single centralized mail server for all the users of the system, the traffic over the network will increase. The system will not be able to tolerate faults and also if any one of the systems fails, the whole system will crash.
- **Centralized tables**: If the data of the users is stored in the centralized tables, the communication lines will be blocked. Thus, the system will become prone to faults and failures.
- **Centralized algorithms**: If the messages in such a large system are sent using a single algorithm, it will take much time to reach the destination due to the large number of users and traffic.

In such a large system, only decentralized algorithms should be used. The characteristics of decentralized systems are stated as follows:

- No machine has the complete information about the system.
- The decisions made by the machines are based on local information.
- Failure of any one system does not affect the overall system.

Check Your Progress

1. Define the term transparency in distributed system.
2. What is the significance of flexibility in distributed systems?

2.3 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A distributed system is said to be transparent if the users of the system feel that the collection of machines is a timesharing system and belongs entirely to him.
2. Flexibility in distributed systems is important because the system is new for engineers, and thus there may be false starts and it might be required to backtrack the system.

2.4 SUMMARY

- The design issues in a distributed system help engineers in developing the distributed system effectively.
- A distributed system is said to be transparent if the users of the system feel that the collection of machines is a timesharing system and belongs entirely to him.
- Transparency can be achieved at two different levels. In the first level, the distribution of the system is hidden from the users. In the other level, the system is made to look transparent to the programs.
- Flexibility in distributed systems is important because this system is new for engineers, and thus there may be false starts and it might be required to backtrack the system.
- Distributed systems are more reliable than single-processor systems because if one system in a distributed system stops functioning, other systems can take over.
- Building a system, which is flexible and reliable, is of no use if the system is slower than a single-processor system. To measure the performance of the system, various performance metrics are used, such as number of jobs per hour, system utilization and amount of network capacity consumed.

2.5 KEY WORDS

- **Monolithic Kernel:** It is an operating system architecture where the entire operating system is working in kernel space.

- **Microkernel:** It states that the kernel should provide very little services and most of the OS services should be provided from the user-level servers.

2.6 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. List the various types of design issues in distributed systems.
2. Discuss the advantages of microkernel system.
3. How the performance of a distributed system is important?

Long Answer Questions

1. What are the various types of transparencies? Explain.
2. What are the two types of schemes for building distributed system?
3. What are the various issues related to reliability of distributed system?

2.7 FURTHER READINGS

Tanenbaum, Andrew S. and Maarten Van Steen. 2006. *Distributed Systems: Principles and Paradigms*. New Jersey: Prentice Hall.

Garg, Vijay K. 2002. *Elements of Distributed Computing*. New Jersey: Wiley-IEEE Press.

Sinha, Pradeep K. 1996. *Distributed Operating Systems: Concepts and Design*. New Delhi: Prentice-Hall of India.

UNIT 3 INTRODUCTION TO COMPUTER NETWORKS

3.0 INTRODUCTION

In this unit, you will learn about the basic concept of networks. Computers are connected by many different technologies. A network is a system of two or more computers that can interconnect in a peer-to-peer or client-to-server fashion, most often over a virtual and shared connection. In this unit, you will also learn about the different network classifications, such as LAN, WAN and ATM technology.

3.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain the different types of networks
- Discuss the different types of network topologies
- Describe the various types of communication protocols
- Explain the Asynchronous Transfer Mode reference model

3.2 COMPUTER NETWORKS

Computer networks form the basis for the distributed systems. A computer network includes both networking hardware and software. Networking hardware deals basically with networking technology and design of computer networks, and software deals with implementing communication between a pair of processes. The reliability and throughput of a distributed system is determined by the underlying hardware and software. In this section we will discuss some of the hardware and software aspects of networking.

3.2.1 Types of Networks

NOTES

A computer network can be as small as several personal computers on a small network or as large as the Internet. Depending on the geographical area they span, computer networks can be classified into two main categories, namely, local area networks and wide area networks.

Local Area Networks

A Local Area Network (LAN) is the network restricted to a small area such as an office or a factory or a building. It is a privately owned network that is confined to an area of few kilometers. In a LAN, the computers connected have a network operating system installed in them. One computer is designated as the **file server** which stores all the software that controls the network and the software that can be shared by the computers attached to the network. The other computers connected to the file server are called workstations. The workstations can be less powerful than the file server and may have additional software on their hard drives. On most LANs, cables are used to connect the computers. Generally, a LAN offers a bandwidth of 10 to 10 Gbps. LANs are distinguished from other networks by three main characteristics including their size, topology, and transmission technology.

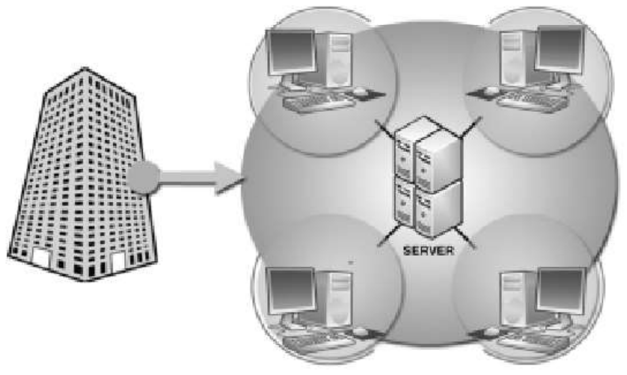


Fig. 3.1 *Local Area Network*

Wide Area Network (WAN)

A Wide Area Network (WAN) spreads over a large geographical area like a country or a continent. It is much bigger than a LAN and interconnects various LANs. This interconnection helps in a faster and more efficient exchange of information at a higher speed and low cost. These networks use telephone lines, satellite transmission and other long-range communication technologies to connect the various networks. For example, a company with offices in New Delhi, Chennai and Mumbai may connect their individual LANs together through a WAN. The largest WAN in existence is the Internet.

NOTES

Fig. 3.2 *Wide Area Network*

3.2.2 Network Topology

A network topology refers to the way a network is laid out either physically or logically. The selection of a particular topology is important and depends upon the number of factors like cost, reliability and flexibility. The various network topologies include bus, ring, star, tree, mesh, and graph.

Bus/Linear Topology

The bus topology uses a common single cable to connect all the workstations. Each computer performs its task of sending messages without the help of the central server. Whenever a message is to be transmitted on the network, it is passed back and forth along the cable from one end of the network to the other. However, only one workstation can transmit a message at a particular time in the bus topology.

As the message passes through each workstation, the workstations check the message's destination address. If the destination address in the message does not match with the workstation's address, the bus carries the message to the next station until the message reaches its desired workstation. Note that the bus comprises terminators at both ends. The terminator absorbs the message that reaches the end of the medium. This type of topology is popular because many computers can be connected to a single central cable.

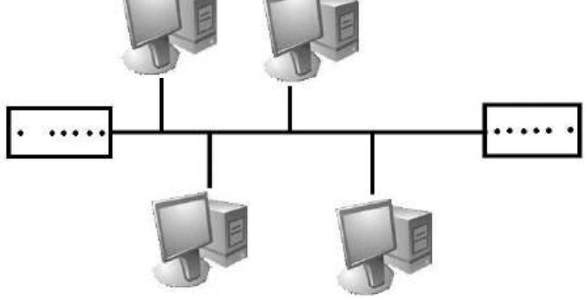


Fig. 3.3 *Bus Topology*

Advantages

- It is easy to connect and install.
- It involves a low cost of installation.
- It can be easily extended.

Disadvantages

- The entire network shuts down if there is a failure in the central cable.
- Only a single message can travel at a particular time.
- It is difficult to troubleshoot an error.

Ring/Circular Topology

In the ring topology, the computers are connected in the form of a ring without any terminated ends. Every workstation in the ring topology has exactly two neighbours. The data is accepted from one workstation and is transmitted to the destination through a ring in the same direction (clockwise or counter clockwise) until it reaches its destination.

Each node in a ring topology incorporates a repeater. That is, each workstation re-transmits data or message received from a neighbouring workstation, no signal is lost and hence, repeaters are not required. In addition, since the ring topology does not have a terminator that terminates the message received, the source computer needs to remove the message from the network.

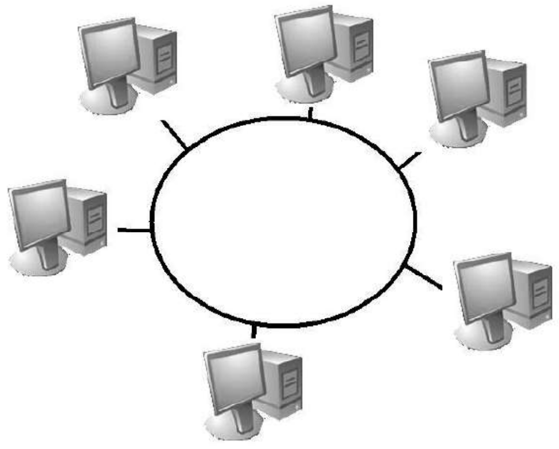


Fig. 3.4 *Ring Topology*

Advantages

- It is easy to install.
- It uses lesser cable length for the installation.
- Every computer is given equal access to the ring.

Disadvantages

- The maximum ring length and the number of nodes are limited.
- A failure in any cable or node breaks the loop and can take down the entire network.

Star Topology

In the star topology, the devices are not directly linked to each other but are connected through a centralized network component known as the **hub** or the **concentrator**. Computers connected to the hub by cable segments send their traffic to the hub that resends the message either to all the computers or only to the destination computer. The hub acts as a central controller and if a node wants to send the data to another node, it boosts up the message and sends the message to the intended node. This topology commonly uses twisted pair cable, however, coaxial cable or optical fibre can also be used.

It is easy to modify and add new computers to a star network without disturbing the rest of the network. Simply a new line can be added from the computer to the central location and plugged into the hub. However, the number of systems that can be added depends upon the capacity of the hub.

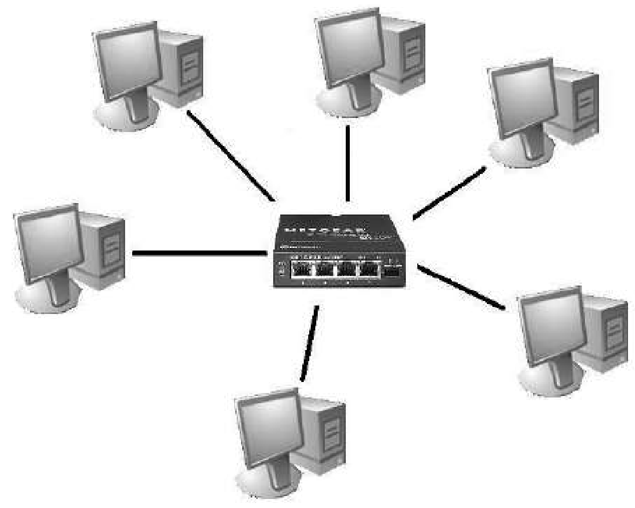


Fig. 3.5 *Star Topology*

Advantages

- It is easy to troubleshoot.
- A single node failure does not affect the entire network.
- The fault detection and removal of faulty parts is easier.
- In case a workstation fails, the network is not affected.

Disadvantages

- It is difficult to expand.
- The cost of the hub and the longer cables makes it expensive over others.
- In case hub fails, the entire network fails.

Tree Topology

The tree topology combines the characteristics of the bus and star topologies. It consists of groups of star-configured workstations connected to a bus backbone cable. Every node is not directly plugged to the central hub. The majority of nodes connect to a secondary hub which in turn is connected to the central hub. Each secondary hub in this topology functions as the originating point of a branch to which other nodes connect. This topology is commonly used where a hierarchical flow of data takes place.

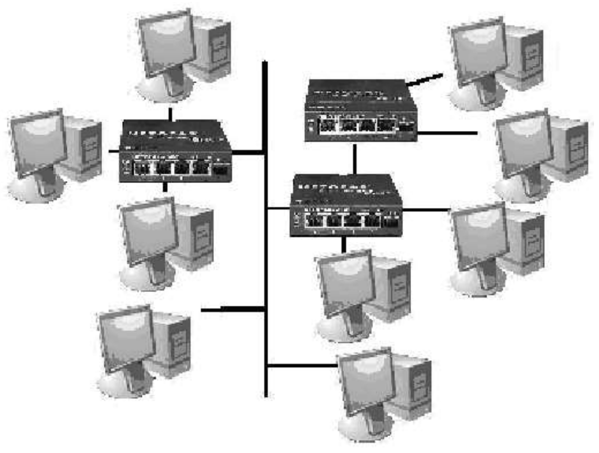


Fig. 3.6 *Tree Topology*

Advantages

- It eliminates network congestion.
- The network can be easily extended.
- The faulty nodes can easily be isolated from the rest of the network.

Disadvantages

- It uses large cable length.
- It requires a large amount of hardware components and hence, is expensive.
- The installation and reconfiguration of the network is very difficult.

Mesh Topology

In the mesh topology, each workstation is linked to every workstation in the network. That is, every node has a dedicated point-to-point link to every other node. The messages sent on a mesh network can take any of the several possible paths from the source to the destination. A fully connected mesh network with n devices has n(n-1)/2 physical links. For example, if an organization implementing the topology has 8 nodes, 8(8-1)/2, that is, 28 links are required. In addition, routers are used to dynamically select the best path to be used for transmitting the data.

The mesh topology is commonly used in large Internet-working environment because it provides extensive back up and outing capabilities. This topology is ideal for distributed computers.

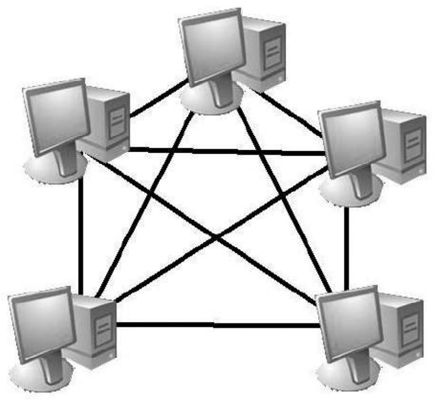


Fig. 3.*7 Mesh Topology*

Advantages

- The availability of large number of routes eliminates congestions.
- It is fault tolerant, that is, failure of any route or node does not result in network failure.

Disadvantages

- It is expensive as more cabling is required.
- It difficult to install.

Graph Topology

In a graph topology, the nodes are connected randomly in an arbitrary fashion. There can be multiple links and all the links may or may not be connected to all the nodes in the network. However, if all the nodes are linked through one or more links, the layout is known as a **connected graph**.

3.2.3 Switching Techniques

The main aim of networking is transfer of the data or messages between different computers. The data is transferred using *switches* that are connected to communication devices directly or indirectly. On a network, switching means routing traffic by setting up temporary connections between two or more network points. This is done by the devices located at different locations on the network called switches (or exchanges). A **switch** is a device that selects an appropriate path or circuit to send the data from the source to the destination. In a switched network, some switches are directly connected to the communicating devices while others are used for routing or forwarding information.

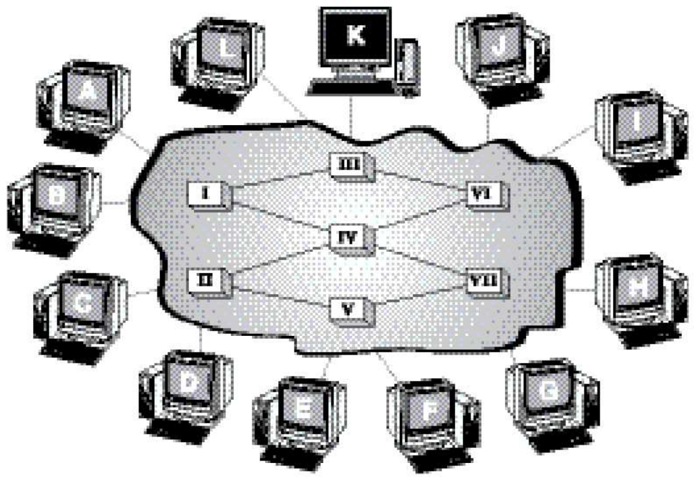


Fig. 3.8 *Switched Network*

Figure 3.8 depicts a switched network in which the communicating computers are labelled 1, 2, 3, etc., and the switches are labelled I, II, III, IV, etc. Each switch is connected either to a communicating device or to any other switch for forwarding information. The technique of using the switches to route the data is called a **switching technique** (also known as **connection strategy**). A switching technique basically determines *when* a connection should be set up between a pair of processes, and *for how long* it should be maintained. There are three types of switching techniques, namely, *circuit switching*, *message switching* and *packet switching*.

Circuit Switching

In the circuit switching technique, first, the complete end-to-end transmission path between the source and the destination computers is established and then the message is transmitted through the path. The main advantage of this technique is that the dedicated transmission path provides a guaranteed delivery of the message. It is mostly used for voice communication such as in the Public Switched Telephone Network (PSTN) in which when a telephone call is placed, the switching equipment within the telephone system seeks out a physical path all the way from the computer to the receiver’s telephone.

In circuit switching, the data is transmitted with no delay (except for negligible propagation delay). In addition, this technique is simple and requires no special facilities. Hence, it is well suited for low speed data transmission.

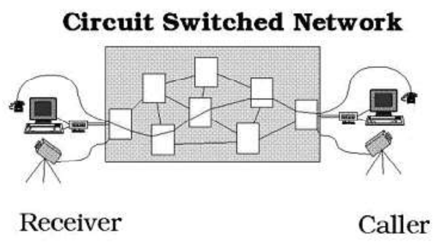


Fig. 3.9 *Circuit Switching*

Message Switching

In the message switching technique, no physical path is established between sender and receiver in advance. This technique follows the *store and forward* mechanism. In this mechanism, a special device (usually a computer system with large memory storage) in the network receives the message from the source computer and stores it in its memory. It then finds a free route and sends the stored information to the intended receiver. In this kind of switching, a message is always delivered to one device where it is stored and then rerouted to its destination.

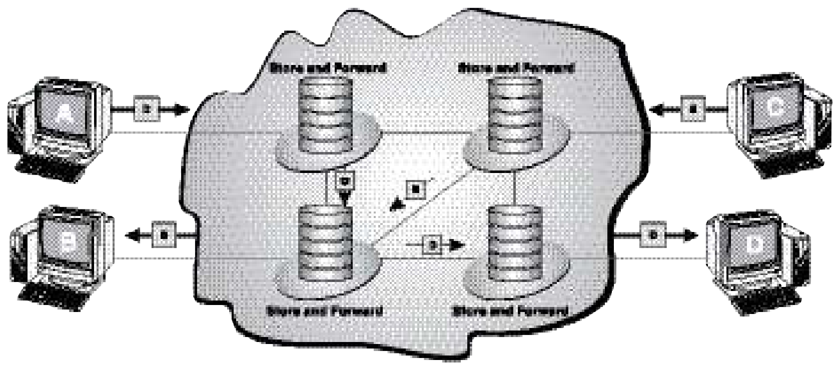


Fig. 3.10 *Message Switching*

Message switching is one of the earliest types of switching techniques, which was common in the 1960s and 1970s. As delays in such switching are inherent (time delay in storing and forwarding the message) and a large capacity of data storage is required, this technique has virtually become obsolete.

Packet Switching

In the packet switching technique, the message is first broken down into fixed size discreet units known as **packets**. The packets are discrete units of variable length block of data. Apart from data, the packets also contain a header with the control information such as the destination address, priority of the message, etc. The packets are transmitted from the source to its local **Packet Switching Exchange (PSE)**. The PSE receives the packet, examines the packet header information and then passes the packet through a free link over the network. If the link is not free, the packet is placed in a queue until it becomes free. The packets travel in different routes to reach the destination. At the destination, the **Packet Assembler and Disassembler (PAD)** puts each packet in order and assembles the packet to retrieve the information.

The benefit of packet switching is that since packets are short, they are easily transferred over a communication link. Longer messages require a series of packets to be sent, but do not require the link to be dedicated between the transmission of each packet. This also allows packets belonging to other messages to be sent between the packets of the original message. Hence, packet switching provides a much fairer and efficient sharing of the resources. Due to these characteristics, packet switching is widely used in data networks like the Internet.

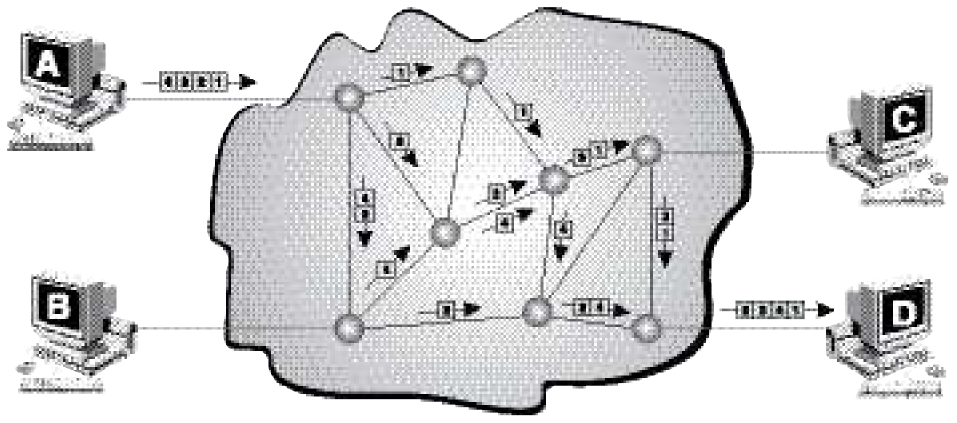


Fig. 3.11 *Packet Switching*

The comparison of the three switching techniques is listed in Table 3.1

Table 3.1 *Comparison between the Various Switching Techniques*

Criteria	Circuit	Message	Packet
Path established in advance	Yes	No	No
Store and forward technique	No	Yes	Yes
Message follows multiple routes	No	Yes	Yes

3.2.4 Communication Protocols

A communication protocol (also known as a network protocol) is a set of rules that coordinates the exchange of information. If one computer is sending information to another and they both follow the same protocol, the message gets through; regardless of what types of machines they are and on what operating systems they are running. As long as machines have software that can manage the protocol, communication is possible. The two most popular types of communication protocols are the ISO protocol and TCP/IP protocol.

ISO Protocol

NOTES

The International Standards Organization (ISO) provided an Open Systems Interconnection (OSI) reference model for communication between two end users in a network. In 1983, ISO published a document called 'The Basic Reference Model for Open Systems Interconnection' which visualizes network protocols as a seven-layered model. The model lays a framework for the design of network systems that allow for communication across all types of computer systems. It consists of seven separate but related layers, namely, Physical, Data Link, Network, Transport, Session, Presentation and Application.

A layer in the OSI model communicates with two other OSI layers, the layer directly above it and the layer directly below it. For example, the data link layer in System X communicates with the network layer and the physical layer. When a message is sent from one machine to another, it travels down the layers on one machine and then up the layers on the other machine. This route is illustrated in Figure 3.12.

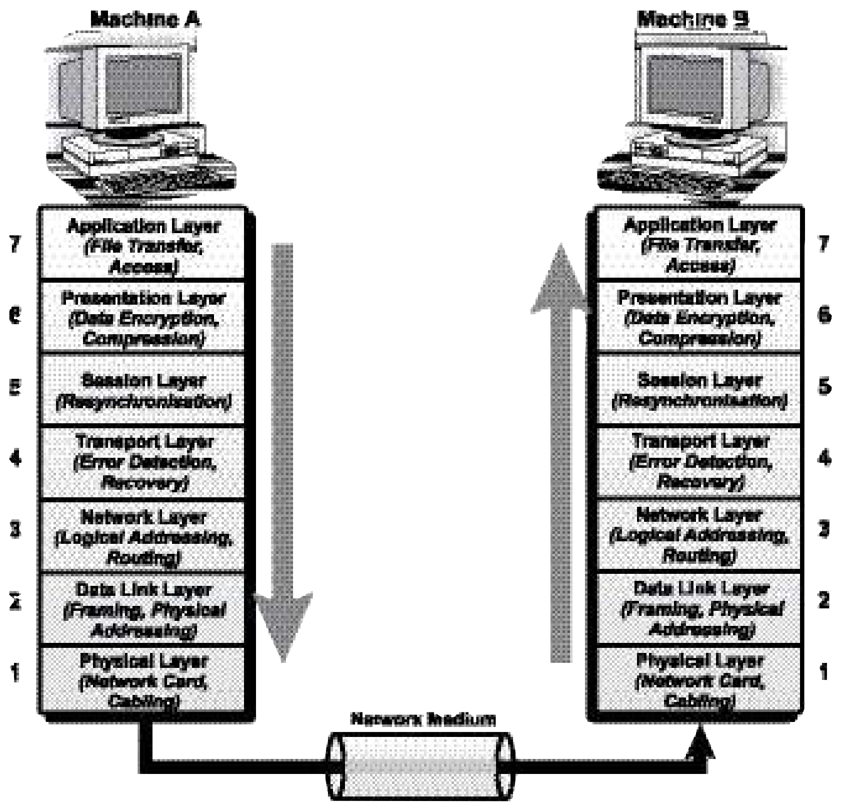

Machine A
Machine B
7 Application Layer (File Transfer, Access)
6 Presentation Layer (Data Encryption, Compression)
5 Session Layer (Resynchronisation)
4 Transport Layer (Error Detection, Recovery)
3 Network Layer (Logical Addressing, Routing)
2 Data Link Layer (Framing, Physical Addressing)
1 Physical Layer (Network Card, Cabling)
Network Medium

Fig. 3.12 *Communication between two machines using OSI Model*

As the message travels down the first stack, each layer (except the physical layer) adds header information to it. These headers contain control information that are read and processed by the corresponding layer on the receiving stack. At the receiving stack, the process happens in reverse. As the message travels up the other machine, each layer strips off the header added by its peer layer.

The seven layers of the OSI model are listed here.

- **Physical Layer:** It is the lowest layer of the OSI model that defines the physical characteristics of the network. This layer communicates with data link layer and regulates transmission of stream of bits (0s and 1s) over a physical medium such as cables, optical fibers, etc. In this layer, bits are converted into electromagnetic signal before traveling across physical medium.
- **Data Link Layer:** It takes the streams of bits from the network layer to form frames. These frames are then transmitted sequentially to the receiver. The data link layer at the receiver's end detects and corrects any errors in the transmitted data, which travels from the physical layer.
- **Network Layer:** This layer is responsible for transferring data between the devices that are not locally attached. Network layers manage the network traffic problems such as routing data packets. The network layer device such as router facilitates routing services in a network. The router checks the destination address of the packet received and compares with the routing table (comprises network addresses). The router directs the packet to an appropriate router to reach the destination.
- **Transport Layer:** The transport layer establishes, maintains and terminates communication between the sender and the receiver. This layer manages the end-to-end message delivery in the network
- **Session Layer:** The fifth layer of the OSI model organizes and synchronizes the exchange of data between the sending and the receiving application. This layer keeps each application at one end and confirms know the status of other applications at other end.
- **Presentation Layer:** The sixth layer in the OSI model is responsible for format and code conversion like encoding and decoding data, encrypting and decrypting data, compresses and decompresses data. The layer ensures that information or data sent from the application layer of a computer system is readable by application layer of another computer system. The layer packs and unpacks the data.
- **Application Layer:** This layer is the entrance point that programs use to access the OSI model. It is the "topmost" or the seventh layer of the OSI model. It provides standardized services such as virtual terminal file and job transfer operations.

TCP/IP Protocol

The Transmission Control Protocol/Internet Protocol (TCP/IP) is the most widely adopted protocol over the Internet. It has fewer layers than that of the ISO protocol, which makes it more efficient. However, it combines several functions in each layer, which makes it more difficult and complex to implement. The various layers in the TCP/IP protocol are listed here.

- **The Link Layer:** It corresponds to the hardware, including the device driver and interface card. The link layer has data packets associated with it depending on the type of network being used, such as Token ring or Ethernet.
- **The Network Layer:** It manages the movement of packets around the network. It is responsible for making sure that packets reach their destinations correctly.
- **The Transport Layer:** It is the mechanism used for two computers to exchange data with regards to software. The two types of protocols that are the transport mechanisms are TCP and UDP.
- **The Application Layer:** It refers to networking protocols that are used to support various services, such as FTP, Telnet, etc.

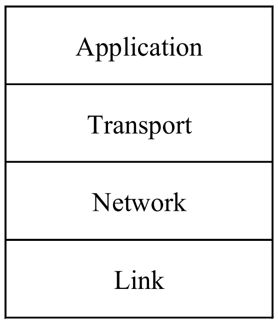


Fig. 3.13 *TCP/IP Protocol*

3.3 ATM TECHNOLOGY/MODEL

A network in which ATM reference model is used is known as ATM network. In the ATM network, a sender first establishes a connection with the receiver. During the establishment of connection, information related to the route, which has been selected to transfer the information, is sent. This information is stored in the switches presented in the selected routes.

After establishing the connection and selecting the route, the sender transfers the information. The information is sent in the form of packets. These packets are chopped into small units called cells, which are of fixed-size. These cells are passed through that route, whose information is stored in the switches. If the connection established by the sender and the receiver is not required for a long time, then the information related to the route stored in switches is flushed out from the switches.

The ATM network scheme has many advantages over traditional packet and circuit switching technique. The most important advantages of this scheme are that a number of items, such as voice, data, audio, images and videotapes can be transferred through a single medium. Therefore, this network is suitable for video conferencing.

In this network, the cells are the most important entity. This means, the network is only concerned about transmission of cells without considering the content stored in these cells. The technique used to transmit cells is known as cell switching, which is a suitable technique for transmitting data into thousands of houses and offices connected to a common transmission media. The main advantage of the cell switching technique is that it handles point-to-point networking and multicasting networking. Elimination of delay occurred due to transmission of a large packet is another advantage of the cell switching technique. Delay is eliminated because of the fixed size of cells.

The ATM network follows its own hierarchy of protocols, which represents the different layers of the ATM reference model. The ATM reference model consists of three layers. The bottom layer of the ATM reference model is known as physical layer, the middle layer is known as ATM layer and the top layer is known as adaptation layer. Figure 3.14 shows the ATM reference model.

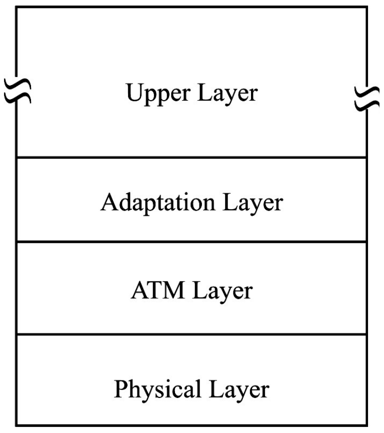


Fig. 3.14 *The ATM Reference Model*

Physical Layer of ATM Reference Model

The functions of the physical layer of the ATM reference model are similar to the physical layer of the OSI reference model. It is responsible for maintaining the physical connection of the network. In the physical layer, an ATM adaptor board is used to transfer the bit stream of data over the transmission medium of the network. The transmission of data should be synchronous. To maintain synchronization of transmission, empty cells are transmitted over the transmission medium. The adaptor board uses the Synchronous Optical NETwork (SONET) in the physical layer. It places different cells into the payload portion of the SONET frame.

The SONET frame is an array having nine rows and 90 columns of bytes. In other words, a frame is a group of 810 bytes. 36 bytes of this frame are overhead and 774 bytes of this frame are payload. A frame takes 125 microseconds to transmit via a transmission medium.

The physical layer is also responsible for generating the essential form, i.e. analog or digital-wave form of data that is required to be transmitted from one end to the other end of the network. This layer defines the bit timings and the other characteristics required to encode and decode data for transmission over the network.

ATM Layer

The ATM layer deals with cells and their transportation. For this purpose, it selects the most suitable route that is similar to the function of the second layer of the OSI reference model. However, it does not recover lost and damaged cells.

In the ATM layer, a 53-bytes cell is used to transmit data. This byte cell includes a 5-bytes header and 48-bytes data field used to contain data stream. This byte cell is known as ATM cell, which is not suitable for SONET payload. Therefore, the ATM cell is spanned with SONET frames and requires two separate levels for synchronization. One level of synchronization is used to detect the starting of the SONET frame and the other one is used to detect the starting of the ATM cell spanned with the SONET frame.

The layout of the cell header of the byte cell depends on the type of devices connected in the network: for example, if a cell is transmitted from a computer to the first ATM switch, then the first four bytes of the cell header are known as General Flow Control (GFC); whereas, if the cell is transmitted between two ATM switches, then four bytes of GFC are involved in the Virtual Path Identifier (VPI) field of the cell header. The fields of the cell header, which are common to all the cell headers, are stated as follows:

- **Virtual Channel Identifier (VCI)**: VCI is used to identify the network and the path related to a particular cell. To identify the path and the network, it is combined with the VPI field. The VPI and VCI fields are modified at each hop along the path.
- **Payload Type**: Payload type is used to distinguish the data cells from the control cells.
- **Cell Loss Priority (CLP)**: CLP is used to mark the less important cells that help in maintaining traffic during occurrence of congestion.
- **Cyclic Redundancy Checksum (CRC)**: CRC is used to maintain accuracy of data.

Figure 3.15 shows the layout of the cell header including GFC field.

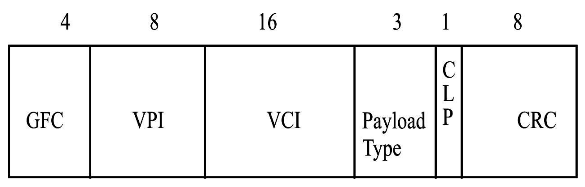


Fig. 3.15 *The Cell Header Including GFC Field*

Adaptation Layer

The Adaptation layer of an ATM network is responsible for managing transmission of cells and packets over the network. Generally this layer is known as AAL, which stands for ATM Adaptation Layer. In the ATM network, a cell can arrive in 3 microseconds and the average speed of network is 155 Mbps. If a computer is able to handle the interrupts with the rate of 300,000 interrupts/sec, then a special mechanism is required by the network. This special mechanism is capable of transmitting packets in the form of fixed size cells and of re-assembling all the cells at the destination end. It is also responsible for generating one interrupt per packet. It is assumed that the main adaptor board is responsible for running the adaptation layer on the board and generating an interrupt after receiving one packet.

The adaptation layer handles the following classes of traffic:

- Constant bit rate traffic
- Variable bit rate traffic with bounded delay
- Connection-oriented data traffic
- Connection less data traffic

Note: *The Upper layer refers to a set of layers included in the reference model that are used to connect the different systems of the network.*

Check Your Progress

1. What are the two main categories of networks?
2. Define network topology.
3. What are the three types of switching techniques?
4. What is a communication protocol?

3.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Computer networks can be classified into two main categories, namely, local area network and wide area networks.
2. A network topology refers to the way a network is laid out either physically or logically.
3. There are three types of switching techniques, namely, circuit switching, message switching and packet switching.
4. A communication protocol (also known as a network protocol) is a set of rules that coordinates the exchange of information.

3.5 SUMMARY

NOTES

- A computer network can be as small as several personal computers on a small network or as large as the Internet. Depending on the geographical area they span, computer networks can be classified into two main categories, namely, local area networks and wide area networks.
- A Local Area Network (LAN) is the network restricted to a small area such as an office or a factory or a building.
- A Wide Area Network (WAN) spreads over a large geographical area like a country or a continent. It is much bigger than a LAN and interconnects various LANs.
- A network topology refers to the way a network is laid out either physically or logically. The various network topologies include bus, ring, star, tree, mesh, and graph.
- The main aim of networking is transfer of the data or messages between different computers. The data is transferred using switches that are connected to communication devices directly or indirectly. A switch is a device that selects an appropriate path or circuit to send the data from the source to the destination.
- The technique of using the switches to route the data is called a switching technique (also known as connection strategy). There are three types of switching techniques, namely, circuit switching, message switching and packet switching.
- A communication protocol (also known as a network protocol) is a set of rules that coordinates the exchange of information. The two most popular types of communication protocols are the ISO protocol and TCP/ IP protocol.
- The International Standards Organization (ISO) provided an Open Systems Interconnection (OSI) reference model for communication between two end users in a network. An OSI model consists of seven separate but related layers, namely, Physical, Data Link, Network, Transport, Session, Presentation and Application.
- The Transmission Control Protocol/Internet Protocol (TCP/IP) is the most widely adopted protocol over the Internet. It has fewer layers than that of the ISO protocol. The various layers in the TCP/IP protocol are Link, Network, Transport and Application.
- A network in which ATM reference model is used is known as ATM network. In the ATM network, a sender first establishes a connection with the receiver. During the establishment of connection, information related to the route, which has been selected to transfer the information, is sent.
- The ATM network follows its own hierarchy of protocols, which represents the different layers of the ATM reference model. The ATM reference model

consists of three layers. The bottom layer of the ATM reference model is known as physical layer, the middle layer is known as ATM layer and the top layer is known as adaptation layer

3.6 KEY WORDS

- **Network Topology:** The way a network is laid out either physically or logically.
- **Switch:** A device that selects an appropriate path or circuit to send the data from the source to the destination.
- **Communication Protocol:** A set of rules that coordinates the exchange of information.

3.7 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. Discuss the different types of computer networks.
2. Write the advantages and disadvantages of ring topology.
3. Differentiate between the following:
    (i) LAN and WAN
    (ii) Star and tree topology
    (iii) Circuit switching and packet switching

Long Answer Questions

1. Explain the different types of networks.
2. What are the different types of network topology? Explain with advantages and disadvantages.
3. Describe the various types of switching techniques.
4. Explain the OSI model in detail. How is TCP/IP model different from OSI model?
5. Explain the ATM reference model.

3.8 FURTHER READINGS

Tanenbaum, Andrew S. and Maarten Van Steen. 2006. *Distributed Systems: Principles and Paradigms*. New Jersey: Prentice Hall.

Garg, Vijay K. 2002. *Elements of Distributed Computing*. New Jersey: Wiley-IEEE Press.

Sinha, Pradeep K. 1996. *Distributed Operating Systems: Concepts and Design*. New Delhi: Prentice-Hall of India.

BLOCK - II

MESSAGE PASSING

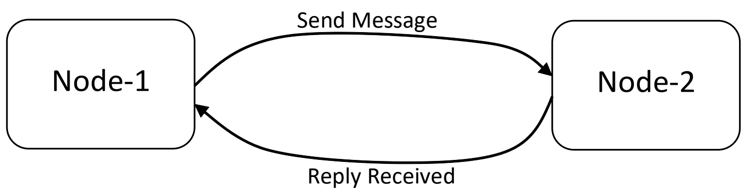
NOTES

UNIT 4 INTRODUCTION TO MESSAGE PASSING

4.0 INTRODUCTION

You have already learnt that the distributed operating system works on the concept of having a collection of independent computers which are capable of communicating with one another in order to complete any task submitted by a user or a routine that needs to be executed by a distributed operating system.

The communication between the computers in a distributed operating system works as a backbone to the whole system. The communication between the computers in a distributed operating system is also known as IPS (Inter Process Communication).



When a user initiates a command starts to execute an application, it will result in initiation of multiple process which need to work in tandem in order to process the request of a user. The process created to complete the request need to interact with one another either by sharing the resources or by passing the messages.

NOTES

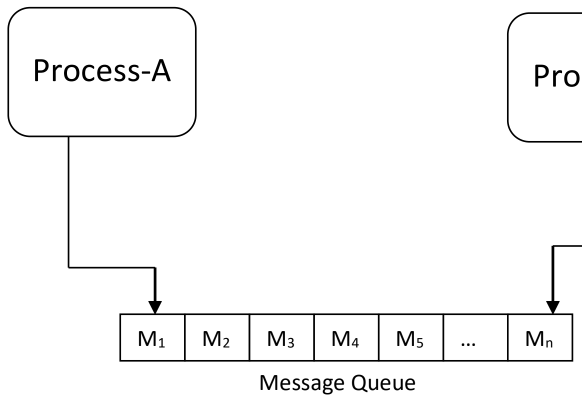


Fig. 4.1 *An Example of Message Passing Communication between the Two Processes*

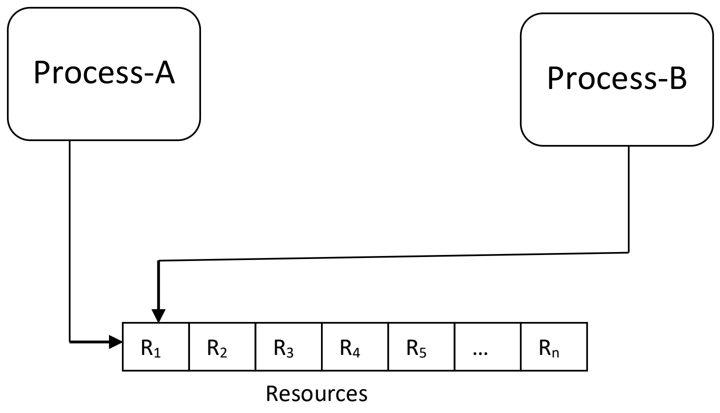


Fig. 4.2 *An Example of Resource Sharing between Two Processes*

One needs to understand the process of communication and inter-process communication within a distributed operating system. The inter-process communication is either implemented using shared-data method or message-passing method. However, in a computer network message-passing method is a preferred choice. The communication in a distributed operating system can be carried out using four different approaches which are listed below:

(i) Message Passing Communication

(ii) Request–Reply Communication

(iii) Transaction Communication

(iv) Group Communication

The second important component in the communication system in a distributed operating system is the network architecture that is used while passing the messages from one computer to the other.

4.1 OBJECTIVES

NOTES

After going through this unit, you will be able to:

- Discuss the features of message passing in distributed system
- Explain the issues encountered in messaging
- Explain the need of synchronization

4.2 FEATURES

The characteristics of any communication in a distributed system should ensure the effectiveness, efficiency and correctness along with optimal usage of resources required for execution of a process from inception till maturity. Keeping in mind the importance of inter-process communication in distributed operation system, following are the basic characteristics that a good inter-process communication system should have.

(i) Ease of Use: The approach used for inter-process communication should be simple and easy to use for different routines or processes of a distributed operating system. A programmer or developer should be able to write the programmes in order to communicate between two or more process in a distributed operating system and these interfaces written should be easy to understand in order to provide ease-of-use feature to a distributed operating system. The ease-of-use feature should not become an obstacle in the performance and efficiency of a distributed operating system. The simplicity of the inter-process communication should allow developers to develop programs for sharing of messages and the internal communication should be handled by the inter-process communication routines.

(ii) Effective Communication: The inter-process communication system should be effective, efficient and correct while passing messages among two or more computers in a distributing operating system. The message to be communicated should not get modified or lost while communicating from one computer system to the other. It should ensure that the message communicated among any two computer system is complete & correct along with the optimal usage of resources. Any communication between two or more processes is the backbone for a distributed computing environment therefore any delay that occurs in inter-process communication will result in decrease in efficiency of a distributed operating system. Therefore, the effective communication is a pivotal feature which a distributed operating system should have in order to complete any process requested by a user.

The inter-process communication should be designed with the objective to reduce the over-heads which are required to be managed while establishing a connection between any two nodes within a network of computers.

(iii) Reliable Communication: The inter-process communication system should use a reliable mode of communication between any two nodes within a distributed system. For example, if a computer system or a node is lost due to some technical error should not result in making the complete system non-interactive because this will result in failure of the whole distributed operating system. Therefore, the inter-process communication system should handle the situation in such a fashion that the operations of a distributed operating system do not get affected.

(iv) Uniformity in Communication: The inter-process communication system should have uniformity in different message passing mechanisms within a distributed operating system. This feature will help developers and users of the operating system to develop routines to interact within the sub-nodes or computers available in a distributed operating system. This characteristic will facilitate the ease-of-use feature in any inter-process communication system. The communication within a distributed operating system can be either local or remote. In local the communication between any two process occurs at the node locally however, in case of remote communication any two process can communicated remotely provided the processes are connected to a network which helps the process in getting connected remotely.

(v) Flexible & Portable Communication: The inter-process communication should be able to accommodate new changes in order to provide hassle free connectivity within the nodes of a distributed operating system. The network of nodes available should have the flexibility feature imbibed in order to adapt to the new developments or changes. Any change or adaption of new technological developments should not hamper the process of communication between two or more nodes. The feature of flexibility will facilitate the process of making the inter-process communication portable as well. The inter-process communication system will be portable when the system has been designed to ensure the adherence of ease-of-use, flexibility and uniformity characteristic within a distributed operating system. The portable feature in inter-process communication will allow a message passing mechanism to be used in any other environment where the working of the message passing mechanism will not stop rather it will continue to work effectively and efficiently. This feature will help the developers as well as users to re-use the message passing mechanisms in different distributed computing environments.

(vi) Secure Communication: The message communicated between any two nodes within a network of computers of a distributed operating system can be modified. These types of threats may lead to breakdown of the communication between any two or more nodes. Therefore, a secure communication system must ensure that the message from a sender to a received should not be viewed or modified by any one during the process of communication within a distributed operating system.

(vii) Complete and Correct data: The message transferred from sender to receiver node may get lost or manipulated while transferring the message. It works as a motivation for people to develop mechanisms which will ensure that the message or data transferred from the sender to the receiver is correct and complete. In other words, it can be said that the data transferred from the sender to receiver must adhere to the basic properties and the same are listed below:

a. *Atomicity:* When a message is transferred from a sender to a receiver either it should reach the receiver as a complete message or it should not deliver any message to the receiver which explains the concept of atomicity in inter-process communication. Therefore, the distributed operating system needs to have the feature of ensuring the atomicity in message transfer which will result in correct and complete transfer of message.

b. *Consistency*: The message transferred from a sender to a receiver is complete and accurate. In case any changes are made at the sender node accordingly all the parameters dependent on the modification are appropriately modified in order to accommodate the change. The modifications made at the sender node should also be delivered at the received node with all the changes that were applied at the sender node.

c. *Isolation*: In case more than one message is passed from a sender to different receiver nodes, the distributed computing message passing procedures should ensure that all the appropriate messages destined for the nodes are delivered correctly at the destined node only. The message or data transferred from a sender should not be delivered at a wrong node which will lead to reduction in performance and reliability of a message passing mechanism in a distributed operating system. While transferring more than one message or data packet from a sender to a receiver the sequence of the messages or data packets is also very important while reframing the messages or data packets transferred from a sender to a receiver. Therefore, a message passing mechanism should ensure that the messages are delivered at the receiver node in the same sequence as was marked by the sender node while sending the message or a data packet. This feature will ensure the completeness of the message at the receiver node as per the format and sequence that has been prepared by the sender node.

d. *Durability*: While transferring message from a sender to a receiver the acknowledgement and reply to message is also very important in order to complete the process of communication between any two nodes. In case the process of message transfer is not complete between any two nodes that resources occupied by these nodes will not be released which will increase the probability of reaching a deadlock state. Therefore, the inter-process communication in a distributed operating system should have a feature to ensure that the process of communication between any two nodes should be monitored for any errors. If the process of communication has not generated any errors then the message transfer process should be completed and the resources acquired by the processes should be released accordingly. However, in case the message passing process has not been completed then appropriate measures need to be taken to ensure that the resources should not remain in acquired state which will result in reduction of performance of whole system.

4.3 ISSUES IN PC MESSAGING

The design of an inter-process communication needs to address some basic issues while designing communication process procedures used for communication between any two or more nodes. The messages are generally transferred from sender to receiver in the form of data packets. The sender node will send the data packet which will include of two basic components fixed length header and variable length block. The fixed length header includes different information related to sender process address, receiving process address, message unique identification number, type of data, number of bytes/ element. The information available in the fixed length header helps a data packet to reach the correct destination and given correct information to the receiver about the originating process as the same is used for sending the acknowledgement from the receiver. Once the process of receiving a data packet is complete then an acknowledgement is sent to the originating process in order to complete the transfer of message between a sender and a receiver.

Variable length Block	Fixed length block

Fig. 4.3 *Data Packet which is Transferred from a Sender Node to a Receiver Node*

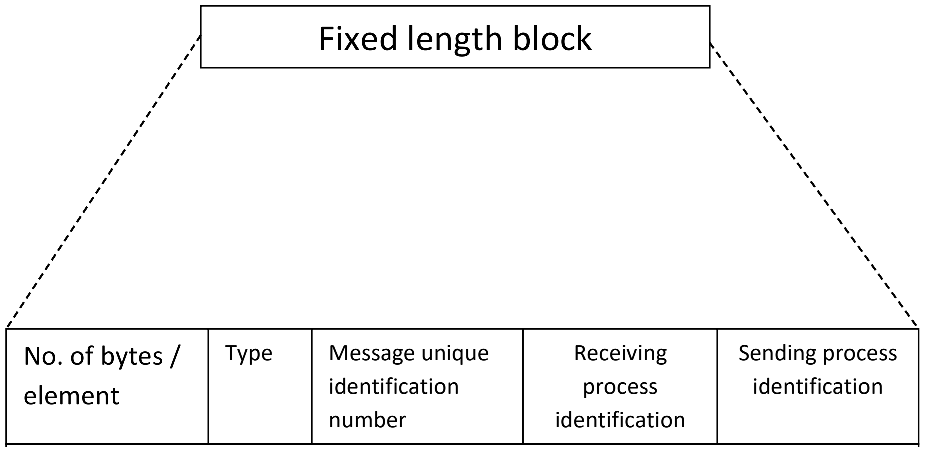

Fixed length block
No. of bytes / element
Type
Message unique identification number
Receiving process identification
Sending process identification

Fig. 4.4 *Parameters Available in "Fixed Length Block" of a Data Packet*

The list of the basic challenges that are encountered by a distributed operating system during inter process communication are given below:

(i) Naming and Name Resolution: Every process in a communication system is assigned a unique identification number know as Process-ID (process identification number). The computer network system should have a naming system which allows a process to names in order to resolve any conflicts or in order to manage the process execution in a distributed operating system or inter-process communication. The implementation of a naming system can be implemented either using distributed or non-distributed approach. The method of selection have a direct impact on the effectiveness and efficiency of a distributed operating system.

(ii) Routing Strategies: The main activity in an inter-process communication is how a data packet will be sent from sender to receiver. Which path the data packet should select in order to reach from source to destination. The data packet may be required to pass through different nodes or computers in order to reach the destination node. Needless to mention the message will be decrypted or viewed only at the destination node not at any of the nodes it passes through in order to reach the destination. The path a data packet selects from the source node to the destination node is known as route. The methods and mechanisms used in an inter-process communication for identifying a route from source to destination is known as routing strategy. The prime concern of any inter-process communication must be to select a routing strategy which will be effective, efficient, secure and should use the resources optimally. The commonly used routing strategies are given below:

a. Fixed routing
b. Virtual Circuit
c. Dynamic Routing

(iii) Connection Strategies: The back bone of any communication between a sender and receiver is the physical link or a physical connection. The method or technique used to establish a physical connection between a sender node and a receiver node is known as connection strategy. If a connection strategy is not selected appropriately will lead to different issues like delay in communication, loss of message in communication or modification of a message during the process of communication. The different connection strategies used in distributed operating systems that need to be selected based on the requirements of an operating system are given below:

a. Circuit Switching;

b. Message Switching;

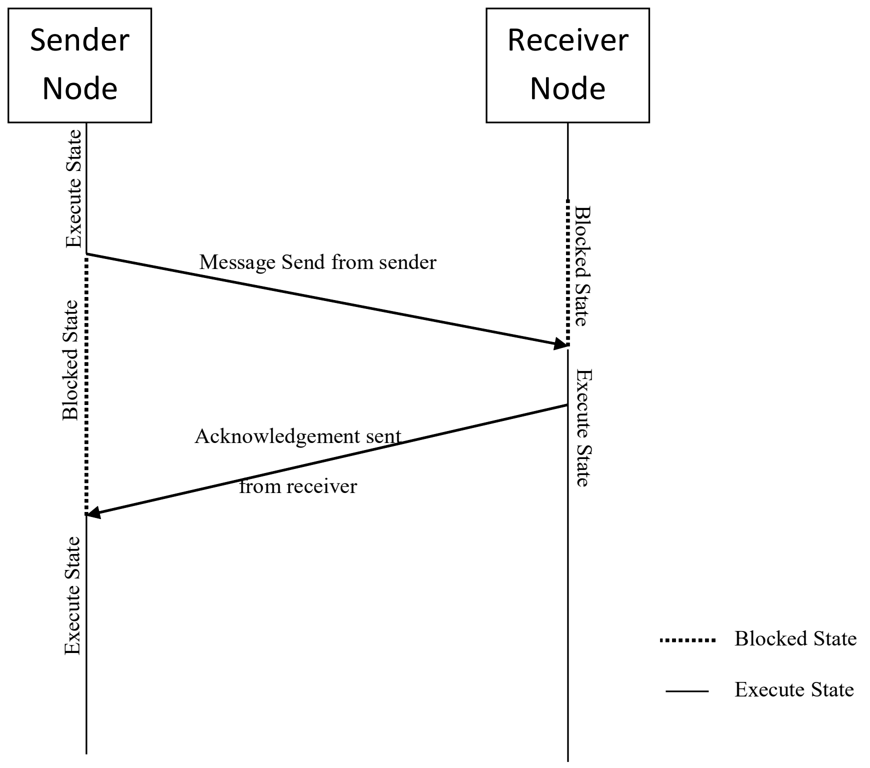
c. Packet Switching

The message within an inter-process communication system is send from the source node to the destination node in a format which includes information about the different attributes like Address, Sequence number, structural information and data within a block. The address parameter corresponds of sender address and received address and the structural information corresponds of information about the type and size of information. The actual data will be available at the end of the block and in some cases the last element of the block will carry pointer to the actual data.

4.4 SYNCHRONIZATION

The basic building blocks of a distributed operating system are cooperation, exchange of data between different nodes of a distributed operating system. In order to exchange data between any two nodes, the computer system at the destination should accept the data that has been sent by a sender which is possible only when synchronization between the sender and receiver is implemented. The process of synchronization gives the details of the timing between sender and receiver which helps the sender and receiver to communicate. The distributed operating system is fully dependent on the communication between different independent computers in a distributed system network which increases the priority and importance of synchronization in distributed operating systems. The process of synchronization can be categorized in two basic categories like blocking and non-blocking synchronization. A communication process is said to be complete only when the message is transferred to the receiver in its original form. The message which is transferred in the form of packets can be send using any one of the two methods of synchronization where if the sender node is blocked after sending the message to the receiver and remains in the block state till the receiver acknowledges the receipt of the message is known as blocking type of synchronization. However, if the sender is not blocked after sending the data packet from sender node rather the control is immediately transferred to the sender node is known as non-blocking type of synchronization. Both blocking and non-blocking type of synchronization are discussed in detail in the following section.

(a) Blocking: In every communication process a message is passed from a sender to a receiver. In the blocking synchronization method the sender after sending the message will wait for the acknowledgement from the receiver and during the wait period the sender will be blocked till acknowledgement is received. Similarly, the receiver after sending the acknowledgement will wait for a message from the sender in order to proceed further. The process of message communication from a sender to a receiver is shown graphically in the figure given below where the dotted lines represent the block state and the solid line represents the execute state. The sender node state changes from the solid line to a dotted line when a message is send from a sender node and the sender node is blocked. The receiver node needs to send the acknowledgement on receipt of the message which results in unblocking of the sender node.



The sender or the receiver node is blocked while the message is being transmitted from sender to receiver and in some cases may result in permanent blocking of either sender or receiver. In order to avoid any situation of permanent blocking of sender or a receiver a timeout is fixed either at the system level or at the process execution level. The timeout is used to fix a time limit for keeping a sender in a blocking stage while waiting for the acknowledgement. In case a sender does not receive any acknowledgement the sender will be clocked till it will reach the timeout limit. Once the timeout limit is exceeded the sender will be unblocked and the system can proceed with further execution of other process which will result in mitigation of reaching a deadlock state. If both the sender and receiver are using the blocking method of synchronization the method of communication is called as synchronous communication. The synchronous method of communication is better the asynchronous communication method as it is simple to implement and the method ensures that the message has reached the receiver from sender. However, the synchronous communication method may lead to different state which can result in a deadlock state where the whole system will not be able to respond.

(b) Non-Blocking: In the non-blocking synchronization method the sender after sending the message is not blocked and the sender will not wait for the acknowledgement from the receiver in order to proceed further with execution. The sender is allowed to go ahead once the message from the sender is moved to buffer. In an inter-process communication process a message is sent from a sender to a receiver via a buffer. A receiver will proceed with other process executions after executing the receive statement. A receiver in non-blocking will not wait for the message from the sender in order to proceed further with other process executions. If both the sender and receiver are using the nonblocking method of synchronization the method of communication is called as asynchronous communication.

In the non blocking synchronization how and when will a receiver node know that the message is available in the buffer? A message may be lost in the buffer and receiver will not receive the message or a receiver has to continuously check the buffer for any messages which are destined for the receiver node. A method of continuously checking the buffer status for any messages by the receiver is followed in non-blocking synchronization method and is known as ***polling***. Another method to intimate the receiver about the availability of a message in a buffer is known as ***Interrupt method***.

In the interrupt method an interrupt is generated using software once the message is available in the buffer and the same is sent to the receiver. The receiver will receive the interrupt and will start with the process of retrieving the message from the buffer. However, the interrupt method requires for efforts and resources in order to implement the synchronization and at times it becomes tough to implement the same using software in distributed operating systems.

In both the blocking and non-blocking methods of synchronization some challenges are to be addressed in order to make the system more robust, effective and efficient along with utilizing the resources optimally. However it is always recommended to use a perfect blend of both blocking and non-blocking methods to complete the communication process between two nodes in an inter-process communication system.

NOTES

Check Your Progress

1. What are the approaches used for message passing in a distributed system?
2. What are the two types of block that a data packet holds?

4.5 ANSWERS TO CHECK YOUR PROGRESS

1. The communication within a distributed operating system can be carried out using four different approaches:
   (i) Message Passing Communication
   (ii) Request–Reply Communication
   (iii) Transaction Communication
   (iv) Group Communication
2. The sender node will send the data packet which will include of two basic components fixed length header and variable length block.

4.6 SUMMARY

- The communication between the computers within a distributed operating system is also known as IPS (Inter Process Communication).
- The inter-process communication is either implemented using shared-data method or message-passing method.
- The messages are generally transferred from sender to receiver in the form of data packets. The sender node will send the data packet which will include of two basic components fixed length header and Variable length block. The fixed length header includes different information related to sender process address, receiving process address, message unique identification number, type of data, number of bytes/ element.
- Every process in a communication system is assigned a unique identification number know as Process-ID (process identification number).
- The path a data packet selects from the source node to the destination node is known as route. The methods and mechanisms used in an inter-process communication for identifying a route from source to destination is known as routing strategy.
- The method or technique used to establish a physical connection between a sender node and a receiver node is known as connection strategy.
- The process of synchronization can be categorized in two basic categories like blocking and non-blocking synchronization.

- If the sender node is blocked after sending the message to the receiver and remains in the block state till the receiver acknowledges the receipt of the message is known as blocking type of synchronization. However, if the sender is not blocked after sending the data packet from sender node rather the control is immediately transferred to the sender node is known as non-blocking type of synchronization.

4.7 KEY WORDS

- **Inter-process communication:** It refers to the mechanisms an operating system provides to allow the processes to manage shared data.
- **Routing Strategy:** It refers to the methods and mechanisms used in an inter-process communication for identifying a route from source to destination.
- **Connection Strategy:** The method or technique used to establish a physical connection between a sender node and a receiver node.

4.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. What is inter-process communication?
2. What information a fixed block of data provides?
3. What are the types of routing and connection strategies used by a distributed system?

Long Answer Questions

1. Explain the features of inter-process communication system.
2. What are the issues of message passing in a system?
3. Explain the blocking and non-blocking process of synchronization.

4.9 FURTHER READINGS

Tanenbaum, Andrew S. and Maarten Van Steen. 2006. *Distributed Systems: Principles and Paradigms*. New Jersey: Prentice Hall.

Garg, Vijay K. 2002. *Elements of Distributed Computing*. New Jersey: Wiley-IEEE Press.

Sinha, Pradeep K. 1996. *Distributed Operating Systems: Concepts and Design*. New Delhi: Prentice-Hall of India.

NOTES

UNIT 5 BUFFERING AND MULTIDATAGRAM MESSAGES

5.0 INTRODUCTION

In this unit, you will learn about the buffering, multi-datagram messages, encoding and decoding. Buffering is the process which uses a memory area for message storage till the receiver node receives it. If the message size is large then it is divided into small datagrams and these datagrams are known as multi-datagram.

5.1 OBJECTIVES

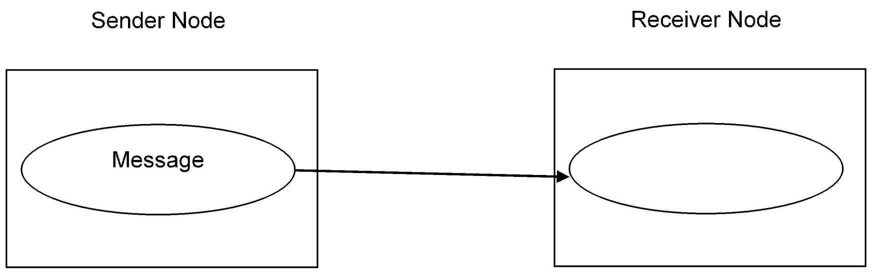
After going through this unit, you will be able to:

- Understand the concept of buffering
- Explain the multi-datagram messages
- Understand the term encoding and decoding

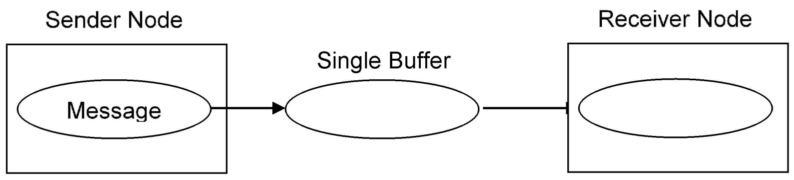
5.2 BUFFERING

The message that is sent from a sender to receiver will either use synchronous or asynchronous communication method. However, when a message is sent from a sender it needs to be temporarily stored in a memory area until the receiver node receives the message. The message can be stored in a memory area that is available at the sender node or at the memory area which is managed by the operating system. This memory area which is used to store the message till the receiver node receives it is known as buffer and the process is known as buffering. Different types of buffering are used based on the requirement of a process within a distributed operating system and some of them are given below:

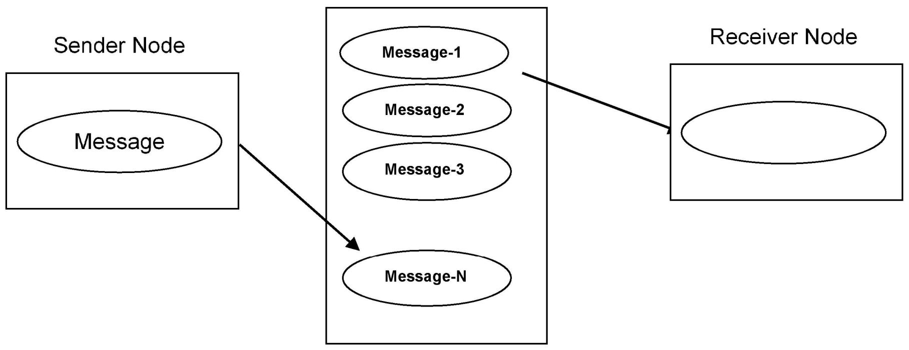
1. **Null Buffering:** This type of buffering doesn't use any buffer rather the send process remains in suspended mode till the receiver node in a position to receive the message. Once the process of send message starts the receiver starts the receiving the message and accordingly an acknowledgement is sent once the message is delivered. The sender node on receipt of acknowledgement sends a message to the received in order to unblock the receiver node for further processing.



2. **Single Message Buffering:** This type of buffering uses a single buffer either at the receiver node address space in order to ensure that the message is readily available to the receiver as and when the receiver node is ready to accept the same. The single message buffer performs better in some situations as the message is available in the buffer which helps the while system in reducing the blocking duration at different nodes. The single message buffer method reduces the delays in communication in comparison with the Null buffering method of communication.



3. **Multiple Message Buffering**: The multiple message buffering communication mechanism is generally used in asynchronous type of communication in inter process communication within distributed operating system. The multiple message buffer as shown in the figure below works as a mail box which is either stored at the receiver's address space or operating system address space. A sender executes the send process in order to send a message and the same is received by the receiver from the mail box as and when the receiver processes the receive message process.



The multiple message buffering is a good mechanism for message passing however, the method is prone to buffer overflow problem. The buffer overflow problem is handled in two different mechanisms where a message send request will generate an error message in case the buffer is full and similarly, the receive message request will generate an error message when the buffer is empty. The second method to handle this problem is to use the multiple message buffer in an controlled fashion where the send message request is processed and the message is stored in the multiple message buffer. The multiple message buffer is blocked till the acknowledgement from the receiver is received to acknowledge the receipt of the message at the receiver node. This mechanism is also treated as forced synchronous mode of communication between different nodes and this mechanism may lead to unexpected occurrence of deadlocks in the communication process within a distributed operating system.

5.3 MULTI-DATAGRAM MESSAGES

The inter-process communication between different nodes within a distributed operating system is an essential part of a network based operating system. The messages transferred from a sender to a receiver in a network is in the form of packets where the data packets correspond of different information attributes like process identifier, address, sequence number, structural information and actual data. A datagram is a self-sufficient and independent packet of data associated with a packet switched network. It carries adequate information to be routed from the source to the destination and the network. Datagrams provide a connectionless communication service across a packet-switched network. Every network allows a maximum allowable size of a datagram that can be transmitted from one node to the other and the same is known as MTU maximum transfer unit. In case the size of the message is smaller than the maximum transfer unit then the datagram is known as “single datagram” message. However, in case the size of the datagram is more than the maximum transfer unit then the datagram is divided into smaller datagrams in order to communicate these multiple datagrams from sender to receiver. The multiple datagrams communicated from sender to receiver are known as multi-datagrams. These multi-datagrams include extra attributes within a packet which carry the information about the sequence of the datagrams and mechanism used for fragmenting. This extra information is used at the receiver end to combine all the multiple datagrams into a single block of information.

5.4 ENCODING AND DECODING

A message transmitted from the source node to destination node is in the form of single datagram or multi-datagram. The message delivered at the destination or receiver node should be complete and correct. Therefore, the structure of the datagram should also be known at the receiver node in order to understand the complete properties of the datagram received. The receiver node should have the complete information related to the datagram available at sender node in order to maintain the consistency and integrity of data. To ensure this the datagram to be sent to the receiver node should be converted into a form which can be transmitted through the communication channel and accordingly on receipt of the packet the same should be converted back to the original form at the receiver node. The process of converted the original data packet into a stream that is compatible with the communication channel is known as encoding of the message and the process of reverted the received message to the original message at the receiver node is known as decoding. The received node encodes the original message and sends the same through the communication channel or buffer to the receiver node where the encoded message is decoded to get the original message back at the receiver node. Different methods are used for encoding and decoding and the two basic representation of encoding and decoding process are Tagged representation and untagged representation.

In the tagged representation all the details about the object along with the data value is encoded and then send through any communication channel or buffer to the receiver. At the receiver node the encoding done using tagged method reverted back to its original form. The data packet received by the receiver node is simple to decode and implement as the information about all the properties of the data packet along with the data is available. However, in untagged representation program objects do contain only data which does not make the data packet delivered at the receiver node as self-explanatory. The receiver node must have the information about the encoding method or mechanism used at the sender node in advance to decode the data packet to its original form.

Check Your Progress

1. What is buffering?
2. What are multi-datagrams?

5.5 ANSWERS TO CHECK YOUR PROGRESS

1. The memory area which is used to store the message till the receiver node receives it is known as buffer and the process is known as buffering.
2. If the size of the datagram is more than the maximum transfer unit then the datagram is divided into smaller datagrams in order to communicate these multiple datagrams from sender to receiver. The multiple datagrams communicated from sender to receiver are known as multi-datagrams.

5.6 SUMMARY

- The memory area which is used to store the message till the receiver node receives it is known as buffer and the process is known as buffering.
- Null buffering doesn't use any buffer rather the send process remains in suspended mode till the receiver node in a position to receive the message.
- Single message buffering uses a single buffer either at the receiver node address space in order to ensure that the message is readily available to the receiver as and when the receiver node is ready to accept the same.
- Multiple message buffering communication mechanism is generally used in asynchronous type of communication.
- If the size of the datagram is more than the maximum transfer unit then the datagram is divided into smaller datagrams in order to communicate these multiple datagrams from sender to receiver. The multiple datagrams communicated from sender to receiver are known as multi-datagrams.
- The process of converted the original data packet into a stream that is compatible with the communication channel is known as encoding of the message and the process of reverted the received message to the original message at the receiver node is known as decoding.

5.7 KEY WORDS

- **Buffering**: It is the process which uses a memory area for message storage till the receiver node receives it.
- **Encoding:** It is the process of converting the original data packet into a stream that is compatible with the communication channel.
- **Decoding:** The process of retrieving the received message to the original message at the receiver node.

5.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. Define buffer.
2. What do you understand by multi-datagram messages?

Long Answer Questions

1. Explain the different types of buffering used based on the requirement of a process.
2. Explain the requirement of encoding and decoding in communication system.

5.9 FURTHER READINGS

Tanenbaum, Andrew S. and Maarten Van Steen. 2006. *Distributed Systems: Principles and Paradigms*. New Jersey: Prentice Hall.

Garg, Vijay K. 2002. *Elements of Distributed Computing*. New Jersey: Wiley-IEEE Press.

Sinha, Pradeep K. 1996. *Distributed Operating Systems: Concepts and Design*. New Delhi: Prentice-Hall of India.

UNIT 6 PROCESS ADDRESSING AND FAILURE HANDLING

6.0 INTRODUCTION

In this unit, you will learn about the process addressing and group communication. Process addressing is required in distributed system to identify the nodes for which the communication process takes place. There may be several reasons that results in communication failure which are also discussed in this unit. Group communication system is a system where a number of members can communicate with each other.

6.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain the process addressing in distributed operating system
- Understand failure handling
- Explain the process of group communication

6.2 INTRODUCTION TO PROCESS ADDRESSING

The process of communication from any two nodes in a distributed operating system cannot be complete without the address of the source node and target node is known. The address of a node or a computer while communicating messages from one node to another is known as addressing or naming. The addressing or naming helps the network to identify the nodes with unique value which is very important in establishing connection between any two nodes of a distributed system. The two basic addressing modes used in distributed operating system are given below:

1. **Explicit Addressing:** when the message is explicitly destined for a process then the message is sent using the explicit mode of addressing. In this

addressing mode, the process-identification (x) along with message (y) is sent to the receiver node (z). The receiver node (z) will only receive the message from process-identification (x). If any other message from process-identification (k) is available it will not be received by the receiver node (z).

2. **Implicit Addressing:** when a message is destined for any receiver node that requires the services of the message then implicit mode of addressing is used. In this addressing mode, the service-identification (x1) along with message (y) is intended to be sent to any receiver node where the service-identification(x1) is offered. The implicit mode of addressing allows a sender node to send the message to more than one node within a network provided the sender has to name a service rather than a process. This type of addressing mode is feasible for client-to-server communication where a client requests for a service and sends the message to all the available servers. The server in turn will receive the message and treat it as a process. Similarly a server can also send a message to all clients to access a service and in-turn only clients which are allowed to use the service can receive the message from the server and acknowledge the same.

The naming of a node is carried out using different methods where the address of the machine is used as the address of the node and in some cases the address of the machine along with the process identification number is used as the address of a node.

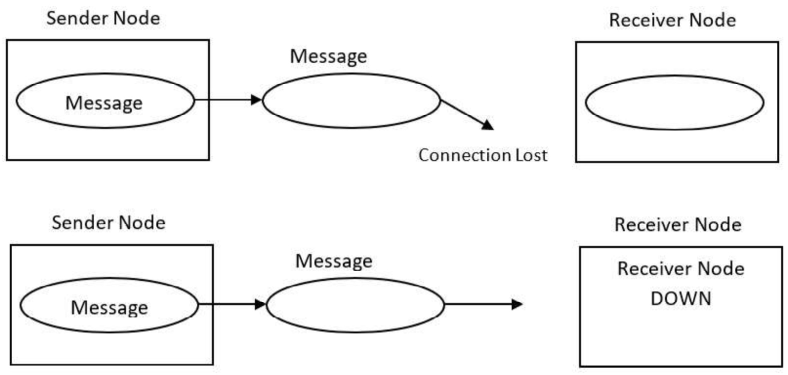
Suppose a node has been assigned an address 159 which for humans is a numerical number and the same number is understood by a machine in the form of machine instruction which is always in 1's and 0's. In case any node intends to send a message to the node which has been given 159 number will be sending the message by creating a data packet where the address will be mentioned as 159. Once the data packet is broadcasted in the network the node with 159 address will accept the data packet and send the acknowledgement to source node. The kernel of an operating system will have information about the task of sending the message from the sender node to the receiver node if only one process is existing within a network. The statement will not be true in case of distributed operating system where at a given point of time one or more number of processes will be existing and all the processes are required to be monitored by the kernel of an operating system. Therefore the address method that has been discussed above will not work for distributed operating system. Another method of address can include more than one segments in the address for example if a machine address is 159 and the process identifier of requesting process is 29 then we can have an address like 159@29 which is self-explanatory to the operating system as the first component gives information about the address of the machine and the second component of the address gives the information about the process identification. The machine address and process identification are unique numbers which are assigned to a machine and a process respectively within a network. Therefore, 159@29 gives the information to the network that the message is destined to machine number 159 where process number 29 requires the message. In this addressing system the load balancing becomes difficult if the number of running processes is more in number. The problem of load balancing can be reduced by including third attribute in the address which can contain the information about the machine address of the last known location. Therefore, the addressing mechanism will have three segments i.e. Machine_Number@Process_Identification_Number @Machine_Number_ lastknown. Once a process is created the first two segments will have constant value till the completion time of the process. However, the third segment of the address will keep on changing as per the movement of the message from one node to the other. The addressing mechanism with three segments is also known as *link-based addressing*. This addressing mechanism will be overloaded, if a process has moved from one node to many nodes while reaching the destination as the overheads in this regard will increase as the message is moving via many nodes. The second problem in this addressing mechanism is that it does not support portability.

In order to mitigate the impact of the above mentioned problems a two level addressing mechanism can be used which include a high level address and a low-level address. The high-level address will be an ASCII string which will be independent of the machine that is being used in order to enhance portability and migration of a process from one system on a network to other system on another network. The low-level address will contain the information about the machine and the process locally. The low-level address will contain the machine identification number and the process identification number. However, this addressing mechanism requires support of the naming server as the ASCII string as high-level address needs to be translated to low-level address which will be stored on naming server. In this addressing mechanism the onus lies on the name server as the same is responsible for generating the low-level address based on the high-level address. In case the name server response time is more the whole system will take more time complete the processing of a process.

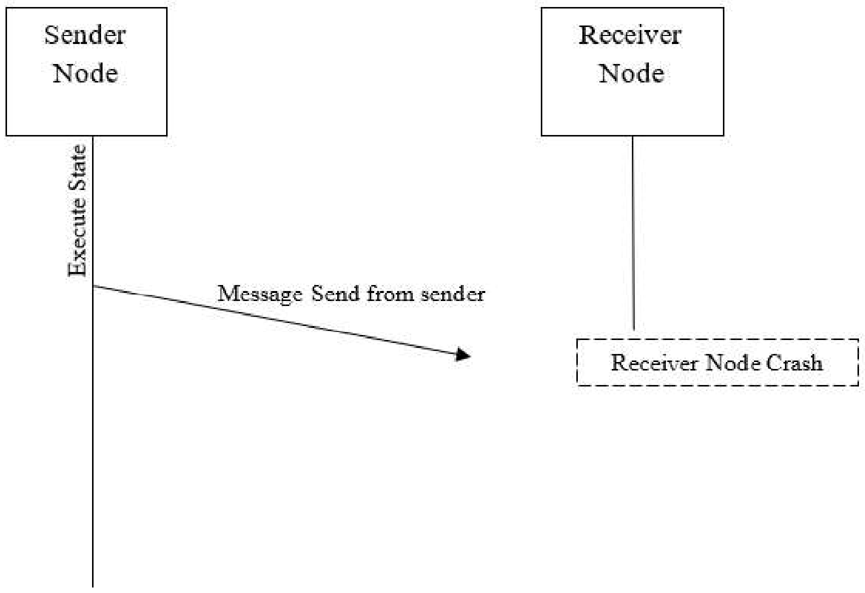
6.3 FAILURE HANDLING

The messages sent from the sender node are not received at the receiver node due to different failures which may occur in a distributed operating system. The scalable and robust design is always pro-active in order to give seamless services to users interacting with a distributed operating system. The failures that can occur while communicating a message within any two or more nodes of a network are discussed below:

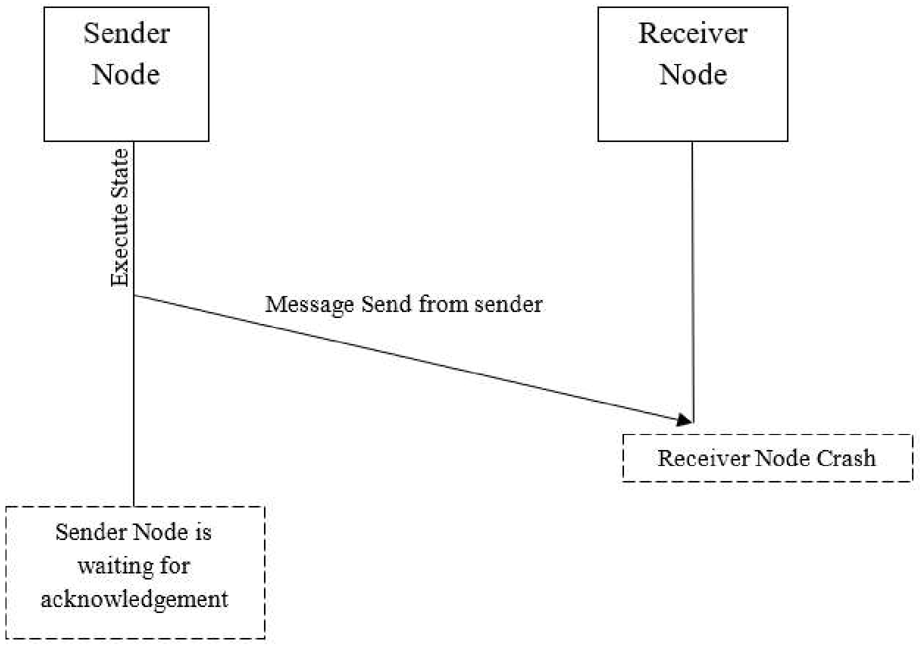
1. **Request Message Lost:** This problem can occur if the sender node sends a message and the receiver is down due to breakdown of the communication link between the sender node and the receiver node.



In case the receiver node is down or gets crashed the communication between the sender node and the receiver node will not be possible and the same is shown in figure given below:



In case the receiver node receives the message but crashes prior to sending the acknowledgement will result failure of communication as the sender node will not receive the acknowledgement. The scenario of receiver crash after receiving the message is shown in figure given below:



The impact of this problem is mitigated by implementing the concept of timeout. In case the sender is not able to receive acknowledgement from the receiver node, the kernel of the operating system will wait for a particular time limit known as timeout. Once the time limit has reached the sender node tries to send the message again and if the acknowledgement is not received the kernel of the operating system initiates the process of freeing the resources acquired by message sending process.

2. **Node/Computer Crashes**: The problem occurs when either a sender node or the receiver node crashes in the process of communication. The kernel of the operating system initiates the process of freeing the resources after waiting for timeout. In case the sender node crashes after sending the message and the receiver node does not receive the message, the kernel of the operating system waits for the timeout limit. After the expiry of the timeout the kernel frees the communication channel and clears the message and the processes associated with the message. Similarly, in case the receiver node crashes after sending the acknowledgement to the sender node which in turn has to reply to the receiver node in order to complete the process and free the resources acquired by the process at the receiver end. The kernel of the operating system will initiate the process of freeing the resources acquired by the process after the timeout period is over. The two main cases where a node may crash are given below:
   (a) Receiver node crashes after receiving message
   (b) Sender node crashes after sending the message

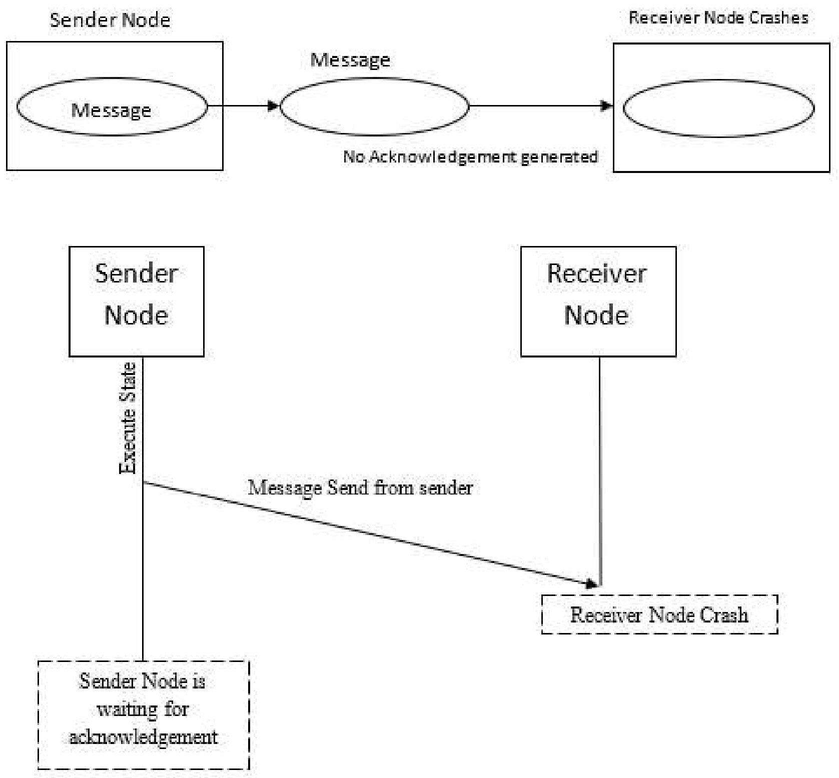


***Fig.6.1 (a)** Pictorial Representation of Receiver Node Crashes after Receiving the Message*

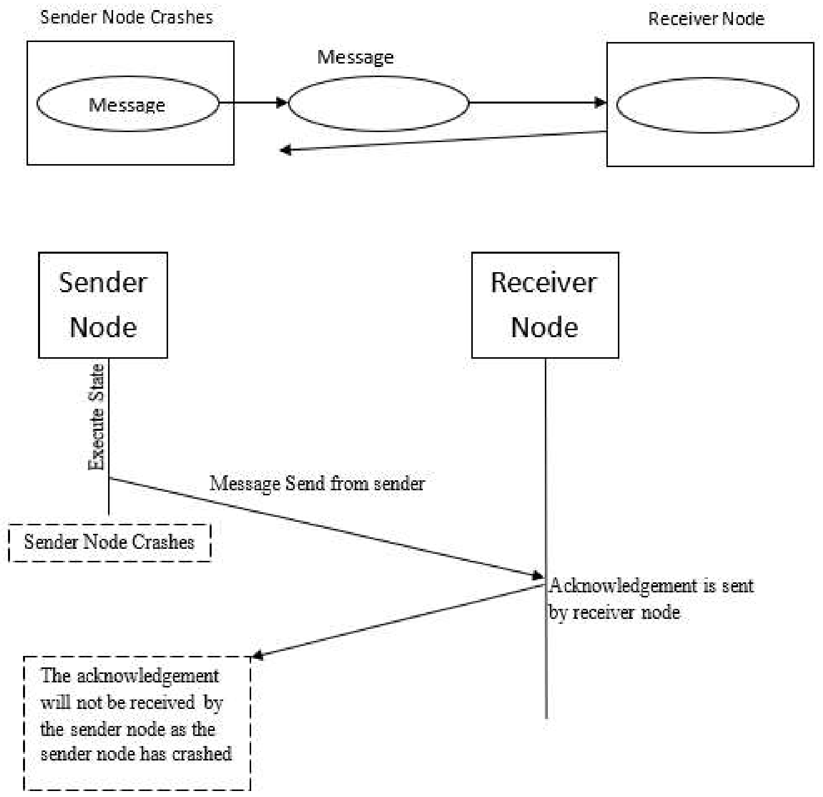
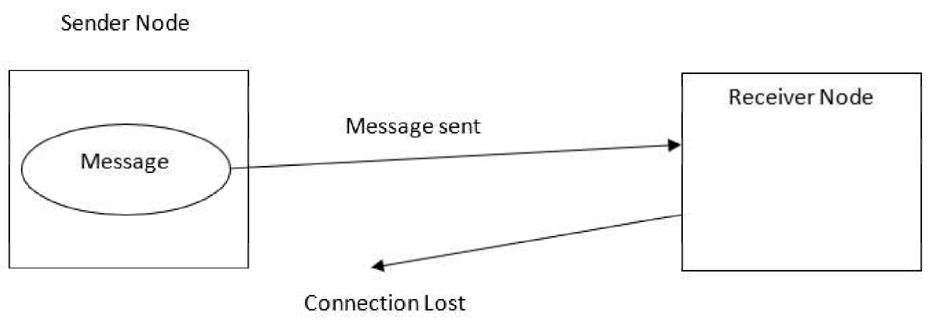

Sender Node Crashes
Message
Message
Receiver Node
Sender Node
Receiver Node
Execute State
Message Send from sender
Sender Node Crashes
Acknowledgement is sent by receiver node
The acknowledgement will not be received by the sender node as the sender node has crashed

***Fig. 6.1 (b)** Pictorial Representation of Sender Node Crashes after Receiving the Message*

3. **Response Message Lost:** This problem generally occur when a receiver node receives the message and the acknowledgement is not received by the sender due to breakdown of the communication link or sender node is down after sending the message to the receiver node.



The impact of this problem is mitigated by implementing the concept of timeout. In case the sender is not able to receive acknowledgement from the receiver node, the kernel of the operating system will wait for a particular time limit known as timeout. Once the time limit has reached the sender node tries to send the message again and if the acknowledgement is not received the kernel of the operating system initiates the process of freeing the resources acquired by message sending process.

The whole system of communication is susceptible to errors or failures which need to be handled properly in order to mitigate the impact of communication failures on distributed operating system. In order to handle the situations which may lead to a failure different tools and techniques are used in distributing computing environments.

One of the failure handling technique is to create checkpoints for all the message passing routines which are required for process completion. The checkpoints give the information about the state of the message passing sub routine to the kernel in order to decide whether a *commit* or a *rollback* is required in order to complete the process of communication between any two nodes within a network. The *commit* point gives information to the kernel about the exact status of a message passing process prior to the node crash in a network. The status information in *commit* is then implemented and stored permanently in the system in order to ensure the integrity and consistency of data within a distributed operating system. The *rollback* point gives the information about the status of a message passing routine within a distributed operating system and the same information helps a kernel to roll back all the changes made as per the information available in the rollback point. The rollback of all the computations is implemented by the operating system by reinstating the status to the previously stored commit point.

Another method to handle failures in communication process is by using retransmission or a message between a sender node and a receiver node where the retransmission of a message is implemented after the expiry of timeout. In this method the kernel of the corresponding machine will wait for the acknowledgement or reply from the corresponding nodes till the timeout limit is reached and after the expiry of the timeout limit the kernel of the machine will retransmit the message. As an example if a sender node sends a message to the receiver node and the sender node is not receiving the acknowledgement from the receiver due to any failure that has occurred during the communication process. The sender node will not retransmit the message unless the timeout is reached. After the timeout limit is reached the sender node will retransmit the message to sender node. Similarly, if a receiver node has received a message and has sent an acknowledgement to the receiver. The receiver node in turn has to send a reply to the receiver node in order to change the status of the communication process to complete. In case the receiver node does not receive the reply from the sender node, it will wait for the timeout limit to expire. Once the timeout limit expires the receiver node will retransmit the acknowledgement to the sender node in order to complete the communication process and free the resources associated with the process.

6.4 GROUP COMMUNICATION

The communication between two processes in RPC is established by calling remote procedures. This technique is not suitable for an environment in which multiple members are involved: for example, consider that multiple servers are cooperating on a single, fault tolerant file system. In this system, the client has to send the message to all the servers so that the task is performed even if one of the servers crashes. RPC cannot handle a communication involving one sender and multiple receivers. It has to perform separate RPCs with every one.

Introduction to Group Communication

A group consists of a collection of processes, which perform the task together in the system. If a message is sent to the group, all the processes receive it. Different members of a group can communicate in two ways, one-to-many communication and one-to-one communication. In the one-to-many communication, one sender and many receivers are involved. One-to-many communication involving one sender and many receivers is shown in Figure 6.2.

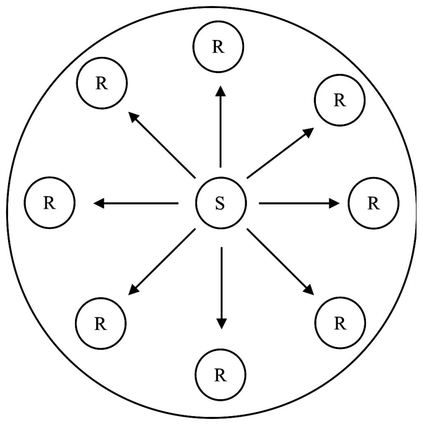


Fig. 6.2 *One-to-Many Communication*

These groups are dynamic and therefore, new groups can be created and old groups can be deleted at any time. A process can join or leave any group and can be the member of many groups at a time. While sending a message to the group, the process does not need to know the number of members in the group. The implementation of communication in groups depends on the hardware. A network address can be assigned to the network, which can be used to send the message in the group. When the message is sent to that particular address, it is automatically delivered to all the members. This process is called multicasting. Multicasting provides a simple technique to implement groups by assigning different multicast addresses for each group. Networks, which do not have multicasting, broadcast the packet. In broadcasting, the packet is sent to all the members of all the groups. Broadcasting the packet is less efficient because when the packets are delivered to every group, the software is required to check whether or not the packet is intended for the computer. If the packet is not delivered, then the packet has to be discarded. Thus, extra time is required for this purpose.

If neither multicasting nor broadcasting can be implemented, then the group communication can be achieved by sending separate packets to each member of the group. However, n packets are required for a total of n members in all the groups. Sending of message from a single sender to a single receiver is called unicasting or point-to-point communication. The process of transmitting data from a single user to a single receiver is shown in Figure 6.3.

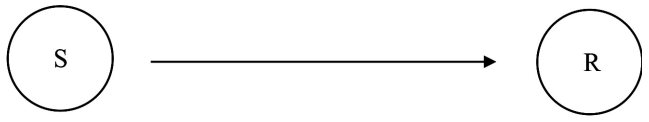


Fig. 6.3 *Point-to-Point Communication*

This process is efficient if the size of groups is small and is not suitable in case the size of the groups is big.

Design Issues related to Group communication

Different types of group communications can be designed to establish communication among multiple users of the system. There are a lot of design possibilities used in the designing of the group communication system. The regular message passing and primitives based communications are examples of such design possibilities. Designing of the group communication system is entirely dependent on the internal organization of a group. The types of group communication systems are stated as follows:

- **Closed group**: This group refers to the group in which outsiders are not allowed to send messages to the group as a whole.
- **Open group**: This group refers to the group in which an outsider can send message to any group involved in the network.
- **Peer group**: This group refers to the group in which every member of the group is connected to the other members of that group.

- **Hierarchical group**: This group refers to the group in which one member of the group acts as a coordinator of the other members of that group.

Closed Groups versus Open Groups

Group communication system can be categorized on the basis of the permission given to a sender, who sends message to a group. Closed group system and open group system are examples of such group systems. In closed groups, only the members of the group can send a message to the other members of the group. A computer, which is not the member of the group, cannot send the message in the group. Figure 6.4 shows the closed group.

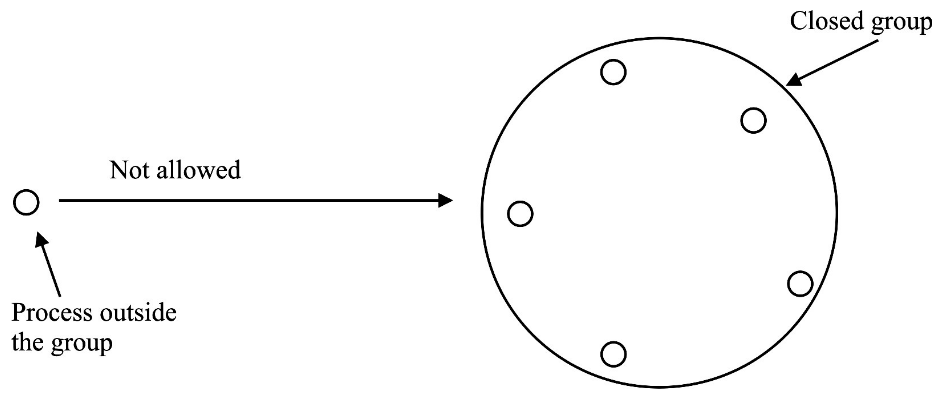


Fig. 6.4 *Closed Group*

Closed groups are used for parallel processing of data: for example, a collection of processes running for computing the moves in a chess game or a group of processes copying a file from one location to another form a closed group. These groups do not interact with the computers outside the groups.

In open groups, any computer can send the message to any other computer or member in the group. Open group is shown in Figure 6.5.

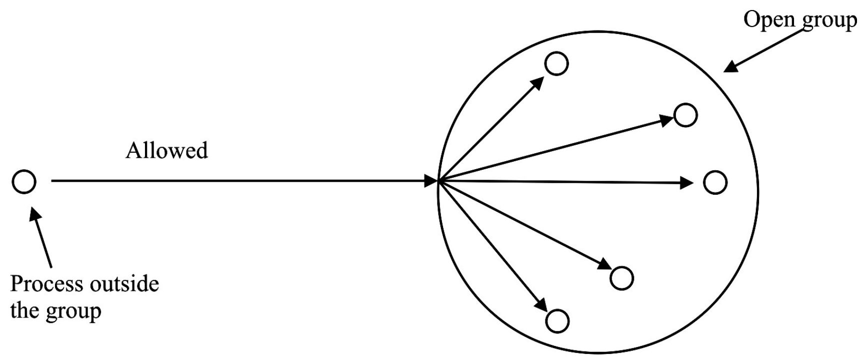


Fig. 6.5 *Open Group*

Open groups are used when coordination between servers is required for performing some task. In this situation, it is required that the computers, which are not members of the groups, should be able to send the message to the other groups.

Peer Groups versus Hierarchical Groups

The members of the group can also be differentiated on the basis of internal structure of the group. In some groups, all the members are considered equal and can make the decisions collectively. This type of group is called peer group. Peer group is shown in Figure 6.6.

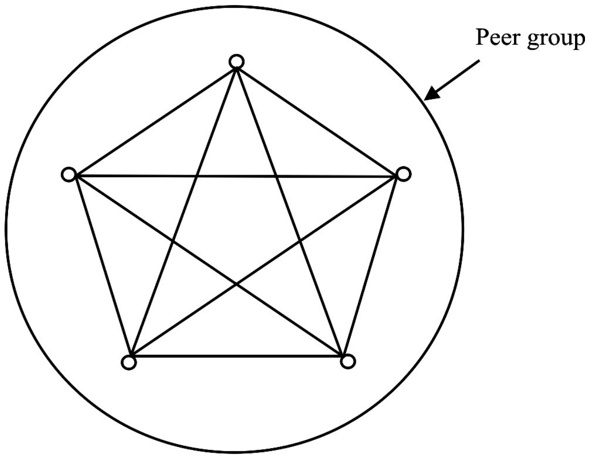


Fig. 6.6 *Peer Group*

A peer group is symmetric and does not fail. If any one member crashes, the other members can take over the job. The disadvantage of this group is that decision-making is very complicated in this group. Voting is performed to decide anything.

In other types of groups, an hierarchy of members exists. One process is made the coordinator, which makes the decisions for all the members of the group. This type of group is called hierarchical group. Hierarchical group is shown in Figure 6.7.

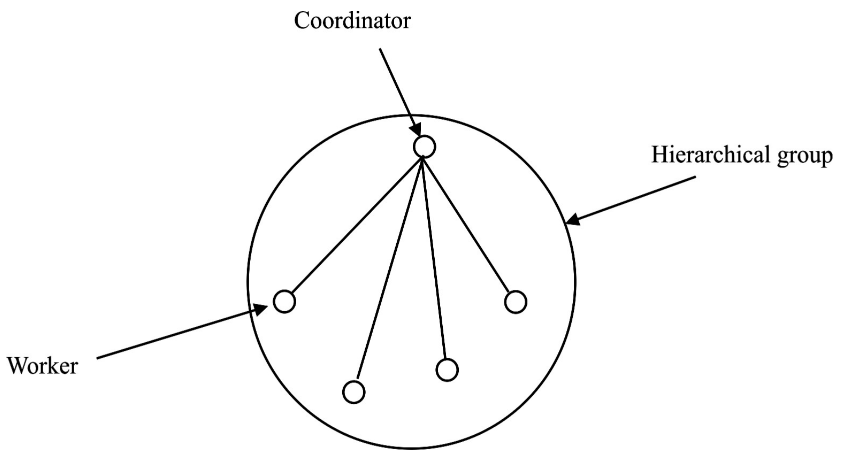


Fig. 6.7 *Hierarchical Group*

When a request for any task is generated by an external client or any member of the group, the coordinator decides which member is best suited to complete the task. In this system, if the coordinator crashes, the whole system comes to a halt. But, if any other member crashes, then it does not affect the coordinator but the performance of the system decreases.

Group Membership

When group communication is employed, then a method is required to create and delete groups and also one method is required to allow the members to leave and join groups. One method can be to have a group server to which all the requests of this type can be sent. The group server maintains the database for all the groups and their members. However, there is a problem with this technique as well. If the group server crashes, then the group management also fails and all the groups have to be reconstructed from the beginning by terminating all the processes. The other method can be to have the membership in a distributed technique. Any computer can send the message to any group to become the member of the group. In case of closed groups also, the members have to send the message to join the group. If any member wishes to leave the group, then it can simply send the message to all the other members.

There are a few problems associated with both of these techniques related to the membership of groups. These problems are stated as follows:

- If any one of the members crashes, then the other members will not be able to know whether the member has really crashed or left the group. The other members have to be certain that the member, which is not responding, has really crashed or left the group.
- One more matter of concern is there, i.e. the process of leaving and joining the group has to be synchronous with the messages. In other words , the computer should start receiving messages as soon as it joins the group and should also be able to send messages to the other members of the group. Besides as soon as the member leaves the group neither it should receive any message from the group nor the members of the group should receive any message from it.
- The final issue is that if many members in the group crash, then some mechanism is required to reconstruct the whole group. Since in such a situation, the group will no longer be able to complete the task, one of the working members has to take up the job of reconstructing the whole group. There could be a problem if more than one member starts reconstructing the group. A mechanism is required to control this problem as well.

Group Addressing

If any computer wants to send a message to a group, then there must be some mechanism to specify the group to which it wants to send the message. The groups can be provided with some unique address to address the groups. If multicast is supported by the system, then the address of the group can be associated with the multicast address. In this way, every message, which is sent to the group, can be multicasted. In this technique, the message is sent only to those groups, which need the message. Group multicasting is shown in Figure 6.8.

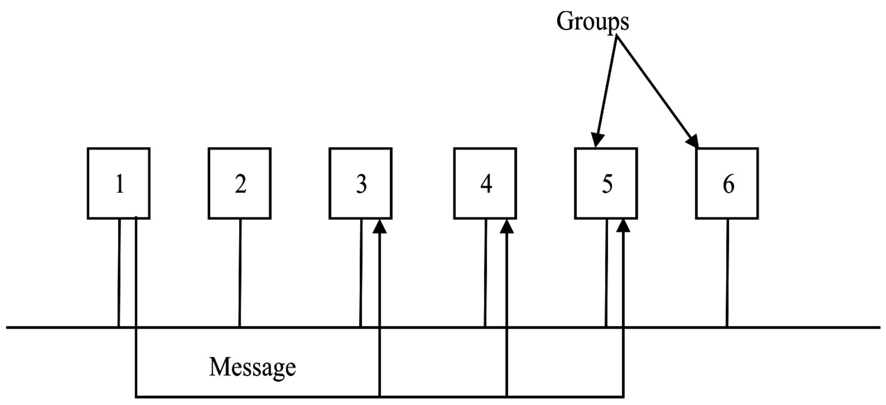


Fig. 6.8 *Group Multicasting*

If the system supports broadcasting but not multicasting, then the message can be broadcasted to all the groups. The groups who do not need the message just discard the message. Group broadcasting is shown in Figure 6.9.

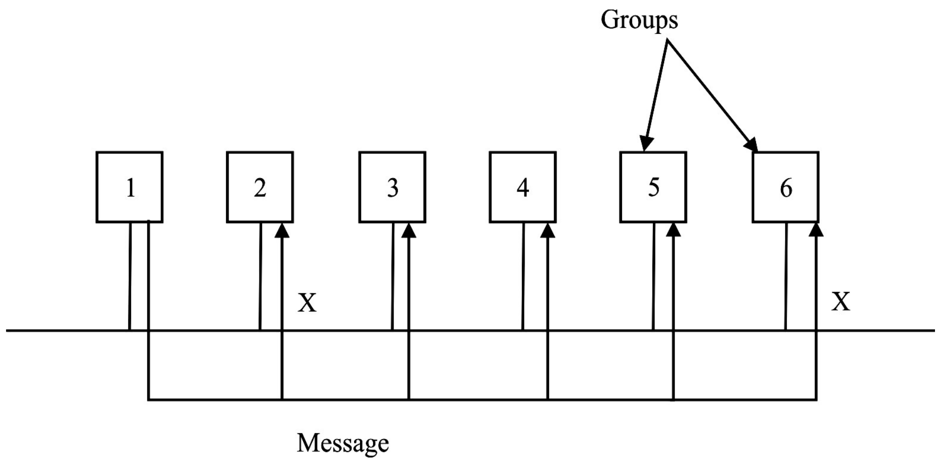


Fig. 6.9 *Group Broadcasting*

If neither of the two techniques is supported b the system, then the kernel has to send the message individually to the groups intended to. This process uses point-to-point communication called unicast for sending the message to all the groups. Point-to-point messaging in the group is shown in Figure 6.10.

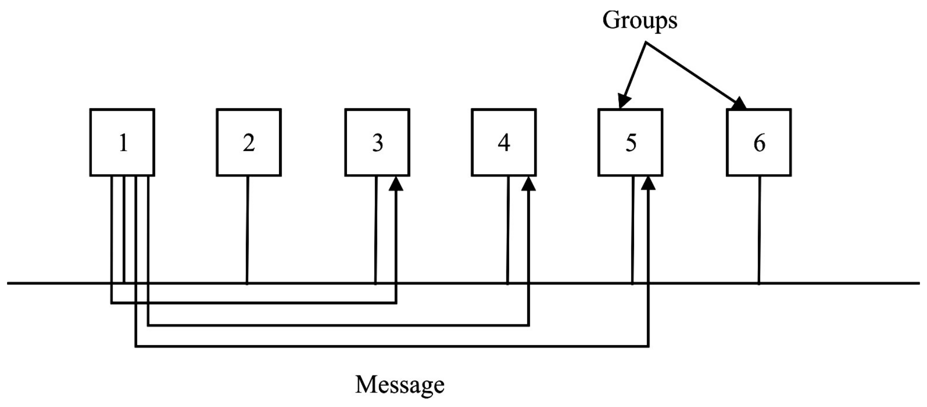


Fig. 6.10 *Point-to-Point Messaging*

In all the three processes, the computer sends the message to the group address and it is delivered to all the members. OS decides the process of sending the message to the groups. The sending computer is not aware of the process used to send the message to the groups.

Another method of sending the message to the group can be to provide an explicit list of the entire destination, such as IP address to the sender. If this method is used, then the location to the destination contains a pointer to the list of addresses as a parameter. In this method, each member of the groups is required to have the information about the other members of the group. If the members in the group are added or removed, then the list of members has to be updated. This task of updating the list is performed by the kernel.

Send and Receive Primitives in Group Communication

The send and receive primitives are also used in the group communication system. In group communication system, n number of replies can be transferred in order to answer a request message. To deal with a number of members of the group, explicit calling of one-way send and receive primitive is used. The send primitive has two parameters, the destination address and the message. If the destination address is related to a process, then message is sent to that process and if the destination parameter contains the address of a group, then the message is delivered to all the members of a group. The receive primitive is used to check whether or not the message is received by the destination. The send and receive primitives can be of any type, such as buffered, unbuffered, blocking and non-blocking.

Atomicity

Atomicity refers to a condition when a message is sent to a group and it is received correctly either by all the members of the group or by none of the member of the group. It is one of the most important properties of the group communication system. Sometimes this property is known as all-or-nothing or atomic broadcast. Atomicity helps in making communication easier among all the members of a distributed system. In other words, when a process of a system having atomicity sends a message to a group, then it does not have to bother about the proper delivery of the message. It also enhances the fault tolerance capability of a distributed system by holding information about the failure of the machine involved in the system.

Message Ordering

Message ordering is another important property of the group communication system. It refers to the right order of the messages received by the different members of a group. To understand the right order of the messages, consider an example of a group, which consists of five processes namely, Process 1, Process 2, Process 3, Process 4 and Process 5. Process 1 and Process 5 have to send message to the other processes of the group. First, Process 1 sends messages to Process 2, Process 3, Process 4 and Process 5 and then Process 5 sends messages to Process1, Process 2, Process 3 and Process 4. Figure 2.23 shows sending of messages in a group.

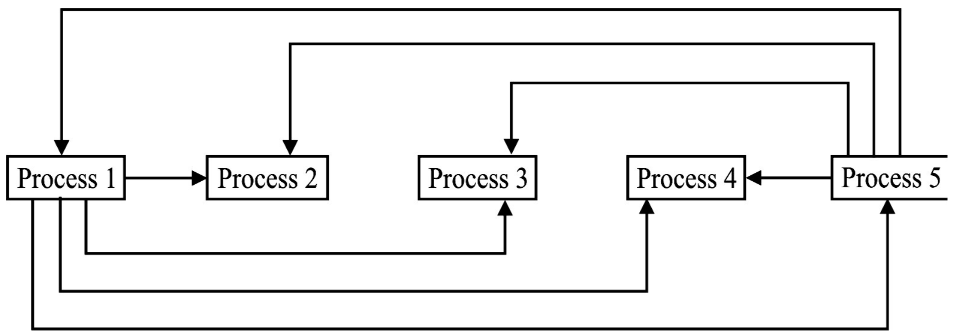


Fig. 6.11 *Message Ordering between Processes*

In Figure 6.11, Process1 sends message to Process2 and then other processes; whereas Process 5 sends message to Process 4 first. In this case, Process 4 receives the message of Process 5 before receiving the message of Process 1. If both the processes, i.e. Process 1 and Process 5 send messages related to updation of the database, then the order of receiving messages can cause problems with the database.

To avoid such type of problems, a proper message ordering is maintained by the system so that the order of receiving the message at the destination end is similar to the order of sending the messages.

Overlapping of Groups

A computer can be the member of any number of groups at a time. This can lead to problems and inconsistency in the groups as well: for example, consider the situation in which there are two groups A and B. Group A consists of members m1, m2 and m3 and group B consists of members m1, m2 and m4 respectively. If

NOTES

the two groups send messages simultaneously to all the members in the group, it can lead to inconsistency in the two groups. Suppose the messages are sent using unicasting so that the member m2 receives the message first from group A and then from group B and m3 first receives the message from group B and then from group A. This process is shown in Figure 6.12.

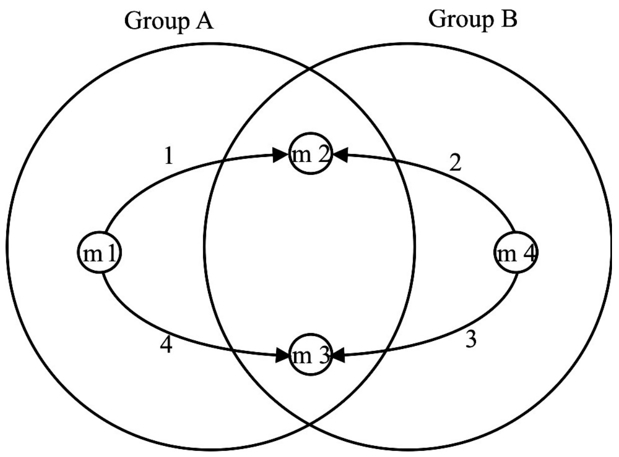


Fig. 6.12 *Overlapping of Groups*

Check Your Progress

1. What is the need of addressing?
2. Name the different communication types used in group communication system.
3. What do you mean by atomicity?
4. What is the function of group server?

6.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The addressing or naming helps the network to identify the nodes with unique value which is very important in establishing connection between any two nodes of a distributed system.
2. Different members of a group can communicate in two ways, one-to-many communication and one-to-one communication.
3. Atomicity refers to a condition when a message is sent to a group and is received correctly either by all the members of the group or by none of the members of the group.
4. The group server maintains the database for all the groups and their members.

6.6 SUMMARY

- The process of communication from any two nodes in a distributed operating system cannot be complete without the address of the source node and target node is known. The address of a node or a computer while communicating messages from one node to another is known as addressing or naming.
- The addressing or naming helps the network to identify the nodes with unique value which is very important in establishing connection between any two nodes of a distributed system.
- One of the failure handling technique is to create checkpoints for all the message passing routines which are required for process completion.
- The checkpoints give the information about the state of the message passing sub routine to the kernel in order to decide whether a *commit* or a *rollback* is required in order to complete the process of communication between any two nodes in a network.
- The *commit* point gives information to the kernel about the exact status of a message passing process prior to the node crash within a network.
- The communication between two processes in RPC is established by calling remote procedures. This technique is not suitable for an environment in which multiple members are involved.
- A group consists of a collection of processes, which perform the task together in the system. If a message is sent to the group, all the processes receive it. Different members of a group can communicate in two ways, one-to-many communication and one-to-one communication.
- If neither multicasting nor broadcasting can be implemented, then the group communication can be achieved by sending separate packets to each member of the group.
- Designing of the group communication system is entirely dependent on the internal organization of a group.
- Closed groups are used for parallel processing of data: for example, a collection of processes running for computing the moves in a chess game or a group of processes copying a file from one location to another form a closed group.
- Open groups are used when coordination between servers is required for performing some task.
- A peer group is symmetric and does not fail. If any one member crashes, the other members can take over the job.
- Atomicity refers to a condition when a message is sent to a group and it is received correctly either by all the members of the group or by none of the member of the group.

- Message ordering is another important property of the group communication system. It refers to the right order of the messages received by the different members of a group.

NOTES

6.7 KEY WORDS

- **Closed Group**: This refers to the group in which outsiders are not allowed to send messages to the group as a whole.
- **Open Group**: This refers to the group in which an outsider can send a message to any group involved in the network.
- **Peer Group**: This refers to the group in which every member of the group is connected to the other members of that group.
- **Hierarchical Group**: This refers to the group in which one member of the group acts as a coordinator for the other members of that group.
- **Atomicity**: It refers to a condition where a message is sent to a group and is to be received either by all the members of the group correctly or none of the members of the group.

6.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. Discuss the significance of process addressing.
2. What do you mean by group membership?
3. Write a short note on group communication.
4. What are the design issues related to the group communication?

Long Answer Questions

1. What are the different ways of process addressing?
2. Explain the various reasons of failures and how they can be handled.
3. What do you understand by group membership? Explain.
4. What is group addressing? Explain.

6.9 FURTHER READINGS

Tanenbaum, Andrew S. and Maarten Van Steen. 2006. *Distributed Systems: Principles and Paradigms*. New Jersey: Prentice Hall.

Garg, Vijay K. 2002. *Elements of Distributed Computing*. New Jersey: Wiley-IEEE Press.

Sinha, Pradeep K. 1996. *Distributed Operating Systems: Concepts and Design*. New Delhi: Prentice-Hall of India.

NOTES

BLOCK - III
DISTRIBUTED SHARED MEMORY

UNIT 7 INTRODUCTION TO DSM

7.0 INTRODUCTION

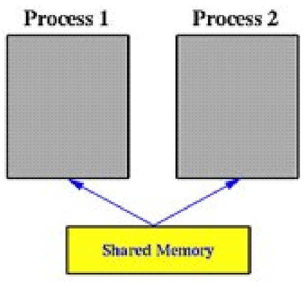
In this unit, you will learn about the distributed shared memory, its architecture, and design and implementation issues. By the literal meaning of "Shared Memory," it can be said that for the execution of multiple processes the same memory space is being shared among the various processes. It is considered as the fast inter-process communication paradigm where an underlying operating system maps a memory segment in the address space of diverse processes in such a fashion that the communication between the memory and executing processes doesn't further require the involvement of operating system to write or read from or to the shared memory segment. In order to streamline the execution of various processes accessing the common shared memory, a much need synchronization mechanism is developed.

In the figure given below, two processes Process 1 and Process 2 share a memory segment attached to address space of both processes. Let's consider the case of process p1 requires to share some data with process p2 the process p1 first needs to read the address of the shared memory and later on can write data into it and similarly the process p2 will also read the address then read the data as shown below:

Data = Read (address)

Write (address, data)

NOTES



7.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain the architecture of DSM
- Discuss the design and implementation issues of DSM
- Explain the term granularity
- Understand the commonly used approaches to build a shared memory space
- Explain the different types of consistency models
- Understand the replacement strategies

7.2 GENERAL ARCHITECTURE OF THE DSM SYSTEM

The general architecture of distributed shared memory system is the representation and arrangement of various components that constitute the working paradigm used by different processes on different nodes to communicate by sharing a common virtual address space to fulfill services and operations. The basic architectural design of a DSM system is shown in the Figure 7.1.

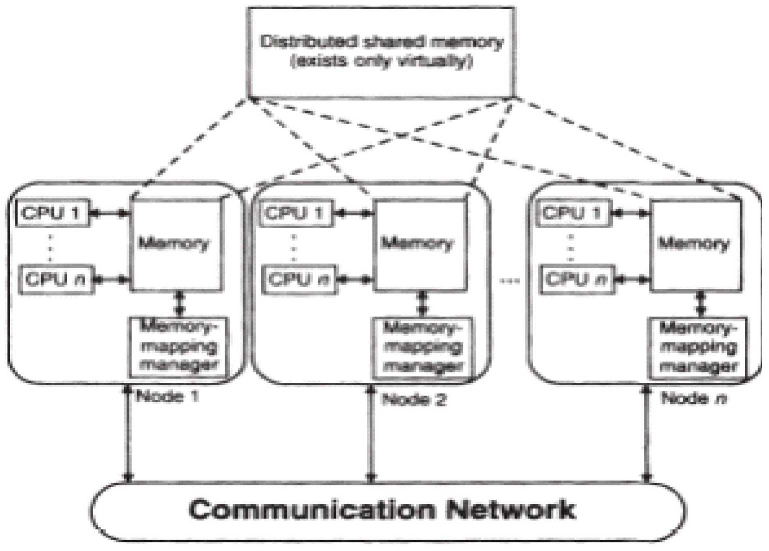


Fig. 7.1 *Architectural Design of a DSM*

The distributed shared memory can be built by interconnecting or organizing various nodes or simply systems in a distributed network arrangement. Each node has one or more processing units that are CPU and associated local memory. All the constituent nodes or systems are connected with one another by a dedicated communication link which in turn is connected to a high-speed communication network. The distributed shared memory in contrast to physical memory is tightly coupled with the processor and can also be looked like a virtual address space build from various individual memories coupled with various connected processors. This shared memory acts like a global address space for all the processors of the distributed network and this address space can be as large as the collection of all the individual memory spaces that is local to each processor in each node. A memory-mapping manager that is in each node of the architecture maps local memory onto the shared memory to constitute a virtual address space, which is global and accessible to all the connected nodes. This virtual memory space is partitioned into various blocks to facilitate better mapping. In order to overcome the latency issues associated with multiple shared accesses, the local memory of each node is also treated as a big cache memory of the shared address space accessible to each individual processor of a node. This local cache is mapped using the memory-mapping manager routine.

The access strategy of DSM architecture works when a process emerging from any node of the network tries to access some data from the shared memory. The request of data access is accepted by memory-mapping manager routine and the data requested is first searched on the cache that is on the local memory. If the data is found on the local address space the access is fulfilled without any latency. However, if the data is not available on the local cache then the memory-mapping manager triggers a network fault and the control is passed to the underlying operating system. The operating system generates a request to the desired node that is the node whose local memory contains the desired data. The data found is later transferred to the requester nodes cache. This pattern of request-identify-retrieve or transfer is performed whenever required to fulfill access operations. However, this underlying architectural shift is not visible to user processes as for them the memory is like a shared global address space. The caching of copies of data on the local memory avoids or reduces the access latency.

7.3 DESIGN AND IMPLEMENTATION ISSUES OF DSM

There are various issues and challenges associated with the design and implementation of a distributed shared memory system. The most prominent issues and challenges are mentioned as under:

- **Granularity:** Whenever a request for data transfer/access on the DSMS is made, the extent of the data quota that can be standardized to be accessed or moved across the processors of various nodes on the network if there is network block fault. The selection of a particular standardized unit to be designated as a memory/data block is an important component in DSMS design.

- **Structure of Shared-Memory Space:** The structure of the data block shared on the network among various nodes typically is influenced by the application type that DSMS is expected to support.
- **Memory Coherence and Access Synchronization:** When similar copies of the data are moved across the shared memory of DSMS. The coherence or the persistence of the data across different memories remains an issue because, whenever a modification is made it is necessary to update all the copies to ensure consistency in DSM. In DSMS, there can be concurrent data access requests from several processes in order to address consistent data accessibility over the DSMS, the access mechanism needs to be synchronized.
- **Data Location and Access:** The mechanism of locating, retrieving and transferring of requested data in DSM by user process needs to be properly designed to respond to block faults effectively by availing consistent data image.
- **Replacement Strategy:** In the situation like whole local cache is fully occupied and the process has to respond to any data transfer request generated by any node on DSMS. If the data requested is not the one that is currently in the local cache, the requested data needs to be retrieved from DSM and brought into local cache by replacing previously held content. Therefore, the replacement of the cached content in this replacement approach is a challenging issue in DSMS.
- **Thrashing:** This is the problem that emerges in DSMS when two processes from two different nodes request the same data block. In this situation the data block is moved back and forth among these competing processes quickly, the quickness is as fast as it seems no data transfer was carried out.
- **Heterogeneity:** If the nodes and their underlying architecture used to design DSMS are homogeneous then no heterogeneity issues can emerge. However, if the underlying environment is different then the heterogeneity takes place. The DSMS architecture design should facilitate positive and productive data communication when DSMS is heterogeneous in nature.

7.4 GRANULARITY

As mentioned above granularity of a particular data block that the DSMS transfers across the nodes need to be fixed at the design level of DSM architecture. The fixed and representational granularity always establishes criteria to quantify the efficiency associated with the network. In order to arrive at a specific unit to granule, the data quantum different criteria specifying different parameters are referenced as mentioned below.

1. Parameters influencing Block Size Selection

- ***Paging overhead:*** The feature of "locality of reference" attributed with shared memory programs to perform a data transactions on DSMS, the process is expected to access a larger component on the shared address space within a small quantum of time. This paging overhead associated to access large memory is comparatively less than the paging overhead required for small block size.
- ***Directory Size:*** In order to maintain the log to record all the data transactions on the DSMS and the overhead associated with it, the larger block size is preferred in contrast to small block size. Small block size means more data transfers therefore, more log maintenance in comparison to large block size with fewer transfers and less log maintenance on DSMS.
- ***Thrashing:*** As many processes may be concurrently accessing the same data reference present in a particular data block. This competing process from different nodes may be trying to update the data instance causes an exponential increase in data transfers over the network stalling the program execution efficiency. This current updating requests made by processes on the same data block causes thrashing. More data block has more changes of more thrashing overheads in comparison to small data blocks as small data may result in fewer data operations.
- ***False Sharing:*** When two different processes from separate nodes requests to access two different data instances residing on the same data block causes no data to be transferred across the network have a direct impact on the productive aspect of DSMS. More data block size has more chances for false sharing in comparison to small data block size.

2. Using Page Size as Block Size.

The above-mentioned factors make it challenging to frame the architectural design for DSMS, address these issues and to ensure efficient data operations among the different processes residing on different nodes of the network. Whether it is a large data block or small data block both are vulnerable to challenges. Therefore, to fix the granularity of a particular data block is itself a challenge to specify a block size to be flashed over the DSMS. To overcome this granularity several DSM architectures address this issue by using page size approaches of a conventional virtual implementation. The implementation of page size in comparison to block size has advantages as mentioned below:

1. The implementation of paging scheme exploits the existing mechanism used to address the overheads like page-faults to separate routines are required.

The memory consistency, cache coherence are resolved by page-fault handlers itself.

2. Integrates the access right mechanism associated with paging to fulfill shared memory access.
3. No undue overhead is imposed on DSMS until the page size overruns packet.

7.5 STRUCTURE OF SHARED MEMORY SPACE

The structure of the shared memory defines the abstract view of the shared memory space accessed by different processors from different nodes on DSMS that may appear to its programmers as storage for words or storage for data objects.

The commonly used approach to build a shared memory space of a DSM system are:-

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

1. ***No Structuring:*** In most of the DSM systems the shared memory space is not structured but is simply an arrangement of liner array of words. The main advantage of this unstructured DSM space is its connivance in choosing any suitable page size to represent a particular data block. When there is no fixed structure for page size it is, therefore, easy to implement in designing DSMS.

2. ***Structuring by data type:*** In this approach, the shared memory layout is structured and the memory space is organized either as a collection of objects or as a collection of variables in the source language. Therefore, the granularity of the unit is also defined either as an object or a variable. Since the behavior of object or variable is not fixed but is varying in nature and depends on the fundamental of application and its underlying language manifestations therefore, DSM systems use variable grain size. This behavior or dependency of grain size on the application nature creates overheads in the design and implementation of DSMS.

3. ***Structuring as a Database:*** In this particular approach of designing a DSMS, the shared memory is well structured like a database. The shared space in this approach is organized into ordered structures called as tuple space. Tuples are commonly designed as a row/column format. The data within the tuple space is accessed and addressed by the content that the tuples are holding. In order to perform any transaction over the network, the processes select the tuples by directly targeting the tuples by specifying the number of their fields and their values or types. Apart from tuple organization, the access mechanism is non-transparent which in contrast is transparent in other DSMS approaches.

7.6 CONSISTENCY MODELS

Different applications that run on DSMS have different consistency requirements attributed to them. The architectural design mentioned above clearly represents that in the DSMS network the count of nodes and their processors is not limited. Therefore, the shared space that is obtained as a virtual global space resulted from the all individual local memory spaces associated with each processor in the whole DSMS. As the memory is shared among the processes therefore, the consistency of the data instances needs to be consistent. The consistency model describes the degree of consistency that is being maintained for the successful and correct data access. Consistency models are designed on the basis of certain protocols that the competing processes must follow to ensure consistency of the share data instances in DSMS. To provide consistent data to a different application in DSMS several approaches were considered to design various consistency models among them the most popular ones are discussed below:

- ***Strict Consistency Model:*** The consistency models in real essence enforce "principle of coherence" on the DSMS to grant mechanism for consistent data. Strict Consistency model is considered as the strongest model approach that strictly adheres to "principle of coherence". The DSMS is said to be a strict consistency model if the process intended to read any variable from some data block on shared memory, the data variable to be accessed is the latest copy of the data variable written in shared memory. In a general context, it can be said that if there is any kind of update operation on any data variable that the latest copy of the data is instantly updated at all places of its existence. The implementation of strict consistency model requires the absolute global clock to synchronize the processes, variables or objects with persistence.
- ***Sequential Consistency Model:*** It is proposed by Lamport is the design mechanism of shared memory where all the processes of the network has got a similar order to access the shared memory to execute different operations. However, there is no restriction on interleaving among the different access operations like read, write. Let's consider that there are three processes to perform read, write and read operations. The DSMS that supports the implementation of sequential consistency model ensures that no operation on the shared memory that lasts till other previous operation is completed. Sequential consistent memory provides one-copy/single copy semantics as all the processing sharing the memory location will encounter the same data contents stored sequentially on DSMS.

- ***Causal Consistency Model:*** It is proposed by Hutto and Ahamad represents a weakening of sequential consistency approach that it makes a refinement between events that are possibly causally related and those that are most certainly not. In other words, causal consistency represents that all the execution are the same as if causally-related read/write operations were executed in an order that reflects their causality. All concurrent operations may be seen in different orders. Memory reference operations that are not potentially causally related may be seen by different processes in a different order. Any two memory reference can be treated as casual if the first processes get influenced by another process in any way. A shared memory system is, therefore, said to support the causal consistency model if all the specific operation on shared memory are potentially causally related and are seen by all other processes in the same order. The implementation of Casual Consistency Model takes into account the dependability of processed with each other which is achieved by maintaining a dependency graph for shared operations.
- ***Pipelined Random-Access Memory Consistency Model*:** It is proposed by Lipton and Sandberg provides again a weaker consistency semantics and is also known as FIFO consistency. All the processes see that all the write operation order made by some process looks different to another process only ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed as if all the write operations performed by a single process are in a pipeline. Write operations performed by different processes may be seen by different processes in different orders". In this model, the consistency is preliminarily posed on write operations.
- ***Weak Consistency Model:*** It is proposed by Dubois et al. is the consistency approach where the "Synchronization accesses (accesses required to perform synchronization operations) are sequentially consistent. Before synchronization access can be performed, all previous regular data accesses must be completed. Before regular data access can be performed, all previous synchronization accesses must be completed. This essentially leaves the problem of consistency up to the programmer. The memory will only be consistent immediately after a synchronization operation". The write operations made by some processes is not necessary to be shown to another process. In order to fulfill the operational mechanism of weak consistency model following recommendations must be adopted.
- All accesses made to synchronization variables must be performed under sequential consistency semantics.
- All the necessary update or write operations on some data variable in shared memory must be completed before access to synchronization variable is permitted.

- All operations on the synchronization variable must be furnished before making access to any non-synchronized variable.
- ***Release Consistency Model:*** Release consistency is essentially the same as weak consistency, but synchronization accesses must only be processor consistent with respect to each other. Synchronization operations are broken down into acquiring and release operations. All pending acquires (e.g., a lock operation) must be done before a release (e.g., an unlock operation) is done. Local dependencies within the same processor must still be respected. Release consistency is a further relaxation of weak consistency without a significant loss of coherence.
- ***Entry Consistency Model:*** Like other variants of release consistency model, it requires the programmer (or compiler) to use acquires and release at the start and end of each critical section, respectively. However, unlike release consistency, entry consistency requires each ordinary shared data item to be associated with some synchronization variable, such as a lock or barrier. If it is desired that elements of an array be accessed independently in parallel, then different array elements must be associated with different locks. When an acquisition is done on a synchronization variable, only those data guarded by that synchronization variable are made consistent.
- ***Processor Consistency Model***: Writes issued by a processor are observed in the same order in which they were issued. However, the order in which writes from two processors occur, as observed by themselves or a third processor, need not be identical. That is, two simultaneous reads of the same location from different processors may yield different results.
- ***General Consistency Model:*** A system supports general consistency if all the copies of a memory location eventually contain the same data when all the writes issued by every processor have completed.

7.7 REPLACEMENT STRATEGY

The space that is made available to cache the shared data on DSMS and facilitates the dynamic migration or replacement of data blocks in the local cache with new or demanded data blocks must consider the following issues to ensure efficient caching and withdrawing data blocks from the local cache of a node in DSMS.

1. Prediction of the block to be replaced to accommodate the new or demanded data block in DSMS
2. Prediction of space to hold the thrashed or replaced block from cache.
3. **Block replacement prediction:** In order to understand the replacement mechanism in DSMS, we have replacement paradigms or algorithms used to perform page replacement for main memory operations and also in shared-memory multiprocessor systems. The classification and categorization of

various replacement algorithms or paradigms mainly are based on the following specified criteria.

- ***Frequency of usage**:* In this case of DSMS the replacement algorithm monitors the usage of a particular data block in cache and the frequency of access operations referenced. If the particular data block within the cache has been cited to be referenced or accessed more frequently than that data block is not replacement with the demand page and accommodated within the DSM. The block that is observed to be the least referenced will be replaced with the incoming data block.
- ***Fixed space or variable space:*** The fixed space algorithms are designed on the principle that the underlying memory-space or cache is of fixed size while as the variable size algorithms assume that the size of the cache is dynamically fixed on the basis of the data block to be accommodated in the cache. That means in variable space approach the cache size depends on the data block size. The replacement in fixed space approach replacement simply selects the particular cache and in case of variable space the fetch operation performed to acquire the data from within the shared memory does not correspond to a replacement approach similarly a swap-out can execute even with a fetch operation. Variable space approach is usually not recommended approach in DSMS design.
- Read-owned and Writable Blocks for which a copy of the data block exists on a cache of any other node in the DSMS network. The block in the cache that holds like data has the highest priority to get replaced. Before replacement, the ownership of the current block is transferred to the cache that holds the mirror copy.
- Read-Owned and writable blocks for which the particular node had solely the ownership status that means that particular node cache holds the data block have the lowest priority of replacement. The lowest priority is assumed because of the overhead associated with the transfer of block and block ownership right to another node.

4. Space prediction to accommodate the replaced page or block

After the replacement of a particular block from cache is accepted the important factor that the page replacement approach has to consider is, to ensure that the data that held by the block to be replaced is not lost. Therefore, the space in shared memory needs to be defined to accommodate such replaced blocks from a cache of different nodes in DSMS. However, the block that is accommodating unused, nil or read-only status has to be replaced without reconsideration to data loss. Similar, replacing and not backing up the page which is read-owned and writable and whose replica is not present on other nodes in DSMS will probably lead to data loss. It is, therefore, very much important factor to consider the best approach to undertake this replacement so safeguard the replaced pages with information which otherwise may be lost or problematic. The best-preferred approaches opted to undertake the replacement guarantee this situation are:

1. Storing/ saving the replaced block on secondary store associated with nodes in network or
2. Or extending to memory space of other nodes to make the replica of the block predicted to be replaced from parent cache.

7.8 THRASHING

As mentioned earlier, thrashing is the situation that emerges in the system usually in shared-memory systems when the allocated cache is filled with the pages that may no longer is needed by processes. The occupation of cache or shared memory by pagers other than the pages that might be required by the processor to get a particular process executed causes the processor to wait till the page is made available in the cache. Process that to looks into the cache to access the page and the cache is full, the available pages need to be flushed away from the shared memory to free space required for pages to remove the chances of further page faults or page misses. The availability of multiple pages in cache facilitates multiprogramming efficiently. However, the logic that can build a standard approach to fill cache with those blocks of data that are an immediate requirement for the processor to perform multiprocessing and avoid those pages of the process to loaded into cache that are not an immediate requirement or necessary to be loaded into cache to avoid unusual filling and frequent page replacement. This unusual page faults and page misses keep processors busy in thrashing pages in and out from cache and reduces the efficiency of a processor to perform multiprogramming efficiently. Various conditions in DSMS that can result in page faults or may cause thrashing are discussed below:

1. Interleaved data access by a processor on multiple nodes results into the movement of data block back and forth across these nodes.
2. Invalidation of data blocks with read-only permission just after their replication in other nodes.

Strategies to overcome the overhead caused by thrashing in DSMS

1. Implementation of an efficient local replacement algorithm.
2. Implementation of application-controlled locks to lock the data block from being accessed by other nodes for a short duration.
3. Locking the data block associated/accessed by some node in DSMS by barring another node to accesses or take away the data block until a specific or designated time period completes.

NOTES

4. By modifying or training the cache coherence algorithm in DSMS as per the context related to a particular data block. In order works, it means that there is a need to have separate coherence algorithm/protocols for each data block based on its characteristics to reduce the overheads caused by thrashing.

Check Your Progress

1. What does structure of the shared memory provides?
2. What does consistency model describes?
3. What is thrashing?

7.9 ANSWERS TO CHECK YOUR PROGRESS

1. The structure of the shared memory defines the abstract view of the shared memory space accessed by different processors from different nodes on DSMS that may appear to its programmers as storage for words or storage for data objects.
2. The consistency model describes the degree of consistency that is being maintained for the successful and correct data access.
3. Thrashing is the situation that emerges in the system usually in shared-memory systems when the allocated cache is filled with the pages that may no longer is needed by processes.

7.10 SUMMARY

- The distributed shared memory can be built by interconnecting or organizing various nodes or simply systems in a distributed network arrangement. Each node has one or more processing units that are CPU and associated local memory.
- A memory-mapping manager that is in each node of the architecture maps local memory onto the shared memory to constitute a virtual address space, which is global and accessible to all the connected nodes.
- The structure of the shared memory defines the abstract view of the shared memory space accessed by different processors from different nodes on DSMS that may appear to its programmers as storage for words or storage for data objects.
- The consistency model describes the degree of consistency that is being maintained for the successful and correct data access. Consistency models are designed on the basis of certain protocols that the competing processes must follow to ensure consistency of the share data instances in DSMS.

- Thrashing is the situation that emerges in the system usually in shared-memory systems when the allocated cache is filled with the pages that may no longer is needed by processes.

7.11 KEY WORDS

- **Distributed Shared Memory:** It is a memory architecture in which addresses the physically separated memories as one logically shared address space.
- **Thrashing:** It is the situation that emerges in the system usually in shared-memory systems when the allocated cache is filled with the pages that may no longer is needed by processes.

7.12 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. Define the term granularity and thrashing.
2. What are the parameters that effect to decide the unit of granule?
3. What do you understand by replacement strategy?

Long Answer Questions

1. Explain the general architecture of DSM.
2. What are the design and implementation issues of DSM? Explain.
3. What are the commonly used approach to build a shared memory space of a DSM?
4. Describe the various types of consistency models.

7.13 FURTHER READINGS

Tanenbaum, Andrew S. and Maarten Van Steen. 2006. *Distributed Systems: Principles and Paradigms*. New Jersey: Prentice Hall.

Garg, Vijay K. 2002. *Elements of Distributed Computing*. New Jersey: Wiley-IEEE Press.

Sinha, Pradeep K. 1996. *Distributed Operating Systems: Concepts and Design*. New Delhi: Prentice-Hall of India.

UNIT 8 APPROACHES TO DSM

NOTES

8.0 INTRODUCTION

You have learnt that the distributed shared memory is a memory architecture in which addresses the physically separated memories as one logically shared address space. In this unit, you will learn about the design approaches of DSM and heterogeneous DSM. In case of a Heterogeneous System Architecture, the memory management unit (MMU) of the CPU and the input–output memory management unit (IOMMU) of the CPU have to share certain characteristics, like a common address space.

8.1 OBJECTIVES

After going through this unit, you will be able to:

- Discuss the design approaches of DSM
- Explain the heterogeneous distributed shared memory
- Discuss the advantages of DSM

8.2 DESIGN APPROACHES OF DSM

On the basis of cache management in distributed shared memory system (DSMS), there are three main design approaches of DSMS. These approaches are as follows:

1. Management of data caching by operating System.
2. Management of data caching by Memory Management Unit (MMU)
3. Management of data caching by language runtime system

1. ***Management of Data Caching by Operating System:*** In this design approach, each node in DSMS has got their local cache/memory and in order to access data block available on other cache associated with any other node on DSMS the underlying operating system triggers trap interrupt. The operating system establishes communication between the nodes through message passing and helps the node to fetch and acquire the desired data block. The different processes like communication process & fetch process the acquiring and migrating data blocks across the nodes on DSM are controlled by operating system. Some of the examples of this design approach of DSMS are IVY and Mirage systems.

2. ***Management of Data Caching by Memory Management Unit (MMU):*** This design approach uses multiprocessors with hardware cache for designing DSMS. The implementation of DSM is completed either by fully or partially using hardware. The behaviour of cache design in DSM depends on bus orientation. If the processors are interconnected using a single bus topology, the cache corresponding to individual node or processor is kept consistent by snooping on the bus and implementation of DSM is carried out in hardware mode. However, if the interconnection topology is connected using switches the directories are used in addition to hardware cache. The persistence of data blocks on different cache blocks associated with individual nodes on DSMS implemented in hardware is controlled by memory management unit. One of the examples where this approach is implemented is DEC Firefly workstation.

3. ***Management of Data Caching by Language Runtime System:*** In this approach, the data caching on DSM is controlled & managed by language runtime system and the nature of DSM is structured. The DSM is collection of various programming construction, tokens or objects. In this approach the placement, replacement or migration of various programming constructs or objects is performed using language runtime systems in coordination with the operating system. In order words it can be said that if any process needs to access any variable the language runtime system accepts the access request and later in coordination with operating system completes the access operation of DSMS.

8.3 HETEROGENEOUS DSM

DSMS is a an integrated heterogeneous system wherein systems with varying architectural configuration are interconnected and is also known as distributed shared memory system. The heterogeneous nature is more exclusively related to the individual components in DSMS like personal PC's, multiprocessors or supercomputers have different specifications attributed to them on the basis of their functionality and objectivity. This heterogeneous environment establishes more dynamic and computable platform to perform diverse operations. The heterogeneous DSMs describe a system with multiple nodes having different architectural makeup to access shared-memory paradigm efficiently in order to fulfill the desired objective successfully. The overhead associated with heterogeneous DSM is to cast data variables into acceptable format and to fix acceptable granularity that is selection of proper block size. These two overheads along with other related parameters are described below.

1. Type casting or Data conversion

In DSMS, different nodes may possess different byte ordering or floating point representations on the basis of their underlying architectural specification. Type casting or data conversion becomes implicit implementation in DSMS while trying to transfer or share information between any two nodes which are of different type. Therefore, it is very important in DSMS to perform type conversion of data variables as per the specification available at the source node before starting the data access at the destination node. It is important to mention here that the DSMS itself doesn't have any idea about the type & application layout that the destination data variable or block possess. Therefore, the DSM takes application programmers into account to acquire the type, details of the data block to be accessed. There are two approaches that are being adopted to perform block or data conversion in DSMS.

a. ***By structuring the DSMS as a collection of source language object*:** In this particular approach the DSM is designed in a structured manner where all the programming constructs are identified as collection of variables or objects in the source language in order to start the data transfer process in terms of objects rather than blocks. The type conversion of programming constructs in source language into objects is carried out by implementing compiler having embedded type conversion routines. The functioning of this conversion scheme is to access the data within the shared-memory. The DSM in turn first checks the type of the data on both source and destination nodes prior to initiating the process of data access. In case the type of data is same then no type conversion are required. However, type conversion is performed before the actual data is accessed from shared-memory or before starting the transfer of data.

b. ***By allowing a single type of data in data blocks:*** In this approach the data blocks are maintained as pages and each page can hold data blocks of one type only. In order to identify the type associated with each page of data the page table is created wherein all the pages and their corresponding type and extent of data contained in that page is notified. This indexing of page and page type in page table helps in identifying the type of page requested by any remote node on shared-memory in DSMS. Whenever the request to a page is made if the type of page is not identical to that of the access request node then the type conversion is performed prior to the migration of page.

2. Block Size Selection

Due to heterogeneous architecture of different nodes connected in DSMS network the page or virtual page size may also have varying size in DSMS. Therefore, selection of particular block size or standard granularity among the nodes with varying setup is complex which results in complex situation which need to be handled and are the matter of concern while designing DSMS. In order to mitigate the impact of this problem different algorithmic models have been developed in order to solve this problem. Some of the solutions to mitigate the impact of this problem are given below:

a. ***Largest page size algorithm*:** In this approach the page size or block size taken into account is as large as the maximum size of a virtual page of any node in DSMS network. This approach helps to reduce frequent page faults cited because of small page size.

b. ***Smallest Page Size algorithm*:** In this approach the page size or block size taken into account is as large as the smallest virtual page size of any node in DSMS network. This algorithm reduces data contention however, the demerit of this approach is increased influx of communication and table management overheads.

c. ***Intermediate page size algorithm*:** This approach tries to minimize the implications caused by either large or small virtual page blocks and maximizing the efficiency in migrating and replacing pages among the communicating nodes on DSMS network. The block size in this scheme is always between smallest and largest virtual memory size associated with nodes in whole heterogeneous DSMS network.

8.4 ADVANTAGES OF DSM

- Hide data movement and provide a simpler abstraction for sharing data. Programmers don't need to worry about memory transfers between machines like when using the message passing model. Easier to implement than RPC since the address space is the same
- Nodes implementing data blocks using sequential programming constructs/ paradigms directly run on DSMS.
- In DSMS complex data structures or data blocks are migrated across nodes on network by simply passing reference by simplifying the algorithmic approach for distributed applications.
- The principle of "locality of reference" facilities the movement the entire page containing the data requested rather than just transferring a small piece of data.
- DSMS is comparatively cheaper architectural design approach than multiprocessor systems to perform parallel executions.

- In DSMS the much larger virtual memory space is built by combining all the local memory associated with each local processor or node in DSMS. The formation of large shared memory space helps to reduce or overcome the overhead caused by disk latency like swapping in case of traditional distributed systems.
- In DSMS large number of heterogeneous nodes can be connected to form DSMS network therefore, much larger shared-memory system accessed by nodes dynamically. While as in case of multiprocessor systems where main memory is accessed via a common bus topology limiting the size of the multiprocessor system.
- The programs developed to share or access shared memory space in multiprocessors can run on DSM systems easily.
- The migration of data blocks from one node to another node on DSMS is comparatively easy to handle as both the processors/processes are accessing the same shared address/ memory space.

Check Your Progress

1. What are the design approaches of DSM based on cache management?
2. What are the two overhead associated with heterogeneous DSM?

8.5 ANSWERS TO CHECK YOUR PROGRESS

1. The design approaches of DSM based on cache management are as follows:
   (i) Management of data caching by operating System.
   (ii) Management of data caching by Memory Management Unit (MMU)
   (iii) Management of data caching by language runtime system
2. The two overhead associated with heterogeneous DSM are type casting and block size selection.

8.6 SUMMARY

- Distributed shared memory is a memory architecture in which addresses the physically separated memories as one logically shared address space.
- Management of data caching by operating System, management of data caching by memory management unit (MMU) and management of data caching by language runtime system are the three design approaches of DSMS.
- The heterogeneous DSMs describe a system with multiple nodes having different architectural makeup to access shared-memory paradigm efficiently.

- The two overhead associated with heterogeneous DSM are type casting and block size selection.

8.7 KEY WORDS

- **Memory Management Unit (MMU):** It is a computer hardware unit having all memory references passed through itself, primarily performing the translation of virtual memory addresses to physical addresses.
- **Distributed Shared Memory:** It is a memory architecture in which addresses the physically separated memories as one logically shared address space.

8.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. What do you understand by heterogeneous DSM?
2. Discuss the overhead associated with heterogeneous DSM.

Long Answer Questions

1. Explain the various design approaches of DSM.
2. What are the advantages of DSM?

8.9 FURTHER READINGS

Tanenbaum, Andrew S. and Maarten Van Steen. 2006. *Distributed Systems: Principles and Paradigms*. New Jersey: Prentice Hall.

Garg, Vijay K. 2002. *Elements of Distributed Computing*. New Jersey: Wiley-IEEE Press.

Sinha, Pradeep K. 1996. *Distributed Operating Systems: Concepts and Design*. New Delhi: Prentice-Hall of India.

UNIT 9 SYNCHRONIZATION

NOTES

9.0 INTRODUCTION

In this unit, you will study synchronizing clocks in distributed systems. Clocks are divided into logical and physical clocks for simplicity in computers. Computers that do not need to synchronize their clocks with real time, implement a logical clock whereas computers that need to synchronize their clocks with real time implement a physical clock. Different algorithms, such as Cristian's algorithm, are used to synchronize the computer clock called timer with the other systems in a distributed environment or with the real-time clock.

In addition, some methods are required to prevent/separate the different processes operating in a computer system from interfering with each other's data and variables, which is referred to as mutual exclusion. A number of algorithms, centralized and decentralized, have been developed to achieve mutual exclusion. All these algorithms make use of a coordinator process to act as head and make the other processes coordinate with each other. Therefore, a method is needed to elect the coordinator process. Election algorithms such as the Bully algorithm, are the methods used to elect the coordinator process. Mutual algorithm techniques require you to fully concentrate on the working of processes and details of low-level processes.

9.1 OBJECTIVES

After going through this unit, you will be able to:

- Analyze the basics of clock synchronization
- Understand clock synchronization algorithms
- Evaluate algorithms for mutual exclusion
- Explain the algorithms used to elect a coordinator
- Analyze the deadlocks in distributed systems, their prevention, detection and avoidance

9.2 CLOCK SYNCHRONIZATION AND EVENT ORDERING

Synchronization is very important in distributed systems. Since distributed systems make use of shared memory, it is important to achieve proper communication between the processes. Distributed algorithms are used when the information is not collected in one place and some processes examine it to make certain decisions regarding the tasks to be carried out. Distributed algorithms are characterized by the following properties:

1. The relevant information is distributed across multiple machines.
2. The processes make decisions only on the basis of local information.
3. The system should not have a single point of failure.
4. A common clock or some other source of precise global time must exist in a distributed system for synchronizing time.

The first three points emphasize that all the information should not be collected at a single place for processing: for example, it is unacceptable to send all the requests for processing to a single-manager process. This puts a heavy burden on the process, especially in large systems. Also, a single point of failure renders the system unreliable, which opposes the concept of distributed systems. This is because a distributed system should be more reliable than an individual system so that if one system fails, the other systems should be able to continue the function. In distributed systems, if the resource allocator machine goes down, it will result in the failure of a large number of systems. Therefore, it is important to achieve synchronization without centralization of processes. The fourth property of distributed algorithms is crucial in itself. The reason is in a centralized system, time is unambiguous because it is known only by making a system call to the central machine. Thus, a process A will get a different time value from a process B that has asked the time after process A. Since there is a single-time component, the time value received by process A will be lower than that received by process B.

The problem that occurs due to the non-implementation of a global time can be best understood by the example of the Unix make program. In Unix, large programs are split into a number of files so that if one source file is changed, only that particular file needs to be recompiled, which greatly increases the speed of processing. When a programmer finishes changing the entire source files, the make program examines the times at which the changes were last made to the source and the object files. Suppose a source file xyz.c has time 11:51 and its corresponding object file xyz.o has the time 11:50, then the compiler knows that the source file has been changed and needs to be recompiled. On the other hand, if xyz.c has time 11:43 and xyz.o has time 11:44, then no recompilation is required.

The above-described scenario will be completely different in a distributed system where there is no global time agreement. Suppose xyz.c has time 11:44 and shortly after some time it is modified and assigned time 11:43 because that machine's clock is slightly slower. Now, as per norms, make will not call the compiler and the resulting program will contain a mixture of object files from old and new sources. As a result, it won't work as desired and the programmer will not be able to understand why. Thus, a measure is required to synchronize all the clocks in a distributed system.

9.2.1 Logical Clocks

All computers are equipped with a timer to keep track of time. A timer is a machined quartz crystal, with a number of crystals which, when kept under tension, oscillates at a well-defined frequency. The frequency depends on the type of crystal used, how it is cut and the amount of tension generated. There are two registers associated with each crystal, a counter register and a holding register. With each oscillation of the crystal, the counter decrements by one. An interrupt is generated each time when the counter goes to zero and is reloaded from the holding register. It is possible to generate an interrupt of sixty times within a second, where each interrupt is referred to as one clock tick.

When the system boots initially, it asks you to enter the date and time, which is then converted into a number of ticks and stored in the memory. The interrupt service procedure adds one to the time stored in the memory after every clock tick, thus keeping the computer's clock up to date. In a single-computer system, if the clock differs from a small amount, it does not make any difference because all the processes use the same clock and will be internally consistent with respect to each other. Therefore, the users here are concerned with relative times and not actual times. However, consider the situation in a distributed system with multiple Central Processing Units (CPUs), each having its own clock. It is not possible to run the crystals in each CPU exactly at the same frequencies, although their frequencies are quite stable. This leads to all the clocks of a distributed system going out of sync and give different time values. This difference in time values is called clock skew and as a result of this, the programs dependent on time (as in the case of the make program) can fail.

Thus, in an attempt to achieve clock synchronization, the computer scientist Lamport suggested that clock synchronization does not need to be absolute. Therefore, if two processes do not communicate, it is not necessary that their clocks be synchronized, as the lack of synchronization will not be noticeable and not cause any problem. Lamport also specified that when processes interact, it is important that the events occur in an order and the value of time does not really matter. It only matters whether xyz.c is older or newer than xyz.o.

In many cases, it is required that the machines should agree on the same time but it may not be according to the real time. This means that all the machines must show a time value of say, 11:00 but the actual time is 11:02: for example, in case of make program, it is only required that all the machines should agree to a common time, say 11:00, even if in real time it is 11:02. The clocks implementing such algorithms are called logical clocks. When the constraint that the clocks must be in accordance with the real time and also show a common time is applied, then such clocks are called physical clocks.

For synchronization of logical clocks, Lamport defines a relation called happens-before. The expression, $a \rightarrow b$ is read as a happens-before b and means that all the processes agree to the fact that the event a occurs first and then the event b. This relationship can happen in two situations, which are as follows:

1. If a process has events a and b such that a occurs before b, then $a \rightarrow b$ is true.
2. If a is an event of sending a message by a process to event b of another process, where event b is the event of receiving the message, then $a \rightarrow b$ is true. This is because an event cannot receive a message before it is sent.

Also, Lamport defines happens-before relationship to be transitive, i.e. if $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$. If two processes have different events, x and y and they do not interact with each other by any means, they are said to be concurrent and the order of their occurrence does not matter.

Now, a way is needed to calculate time, C(a) for an event so that all the processes agree to it. Also, if $a \rightarrow b$, then $C(a) < C(b)$ and the clock time, C must always move forward and not backward. Therefore, time can be corrected by adding a positive value to it and not subtracting it.

Lamport's algorithm for assigning time to events can be explained by considering the following scenario. There are three processes running on three different machines and each has its own clock running at its own speed. Figure 9.1 shows the clocks of three processes running at different frequencies.

NOTES

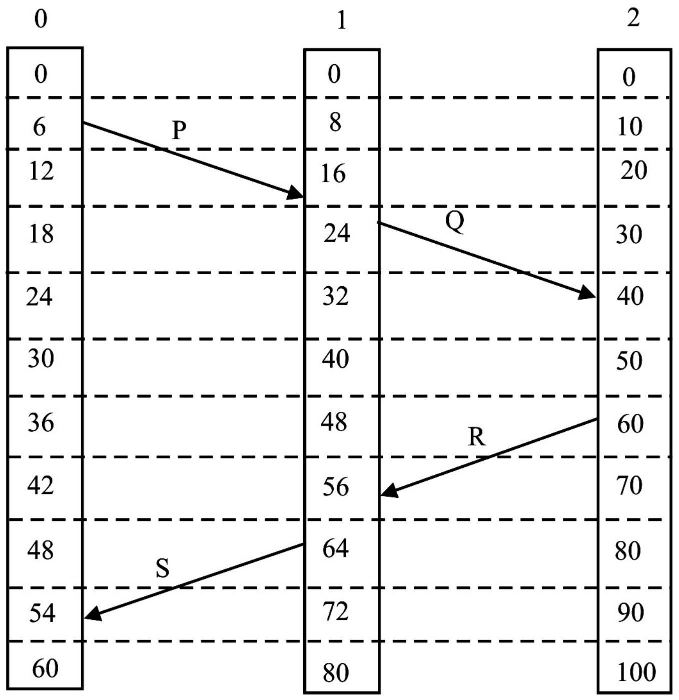


Fig. 9.1 *The Clocks of Three Processes running at Different Rates*

When the clock of process 0 ticks 6 times, the clock of process 1 has ticked 8 times and that of process 2 has ticked 10 times. Every clock runs at a constant rate but the rates of each clock vary due to differences in crystals. At the 6th tick, process 0 sends a message P to process 1. The time taken by the message to reach a process depends on the selected clock. In Figure 9.1, when the message arrives at process 1, its clock reads 16. Therefore, if the message carried the starting time as 6 with it, process 1 will consider that it took 10 ticks for the message to make the journey. Similarly, when process 1 sends the message O to process 2, it takes 16 ticks to make the journey. Both these cases are plausible. Now, message R is sent by process 2 to process 1. The message starts at time 60 and reaches at time 56. Similarly, message S starts at time 64 and reaches at time 54. Both these cases are not possible and such situations must be prevented.

Lamport provides the solution to this situation directly from the happens-before relationship. Since message R left at 60, it must arrive at a time value greater than 60. Since each message carries the sending time (starting time) according to the sender's clock, therefore, when a receiving clock shows a time value prior to the sending time, it fast-forwards its clock by one more than the sending time. Figure 9.2 shows how Lamport's algorithm solves the problem of clock synchronization.

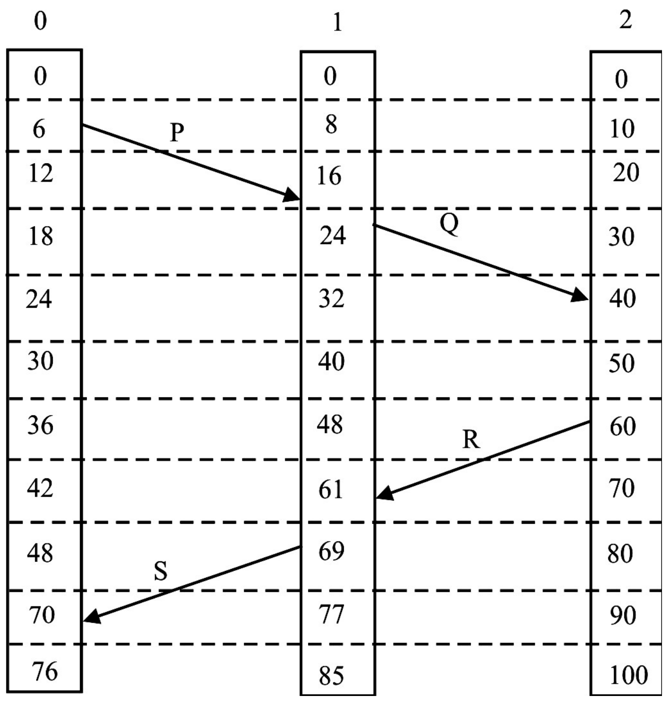


Fig. 9.2 *Lamport's Algorithm corrects the Clocks*

In Figure 9.2, by applying Lamport's algorithm, message R now reaches at time 61 and message S arrives at time 70. Also, if an additional constraint is applied between every two events, the clock must tick at least once and that makes this algorithm meet the requirements for global time. Therefore, if a process sends two messages in quick succession, the clock must tick by at least one in between the sending of two messages.

Some cases also require imposing an additional constraint that no two events can occur at the same time. To accomplish this constraint, the process number in which an event occurs can be attached to the lower-order end of the time, separated by a decimal point. Therefore, this algorithm provides you a way to allot time to all events in a distributed system and hence order them properly.

9.2.2 Physical Clocks

While many systems can do with absolute times, some systems require proper synchronization with real time. Such systems require external physical clocks, and to achieve efficiency and provide redundancy, multiple clocks are required. However, this leads to the problem of synchronizing them with real-world clocks and synchronizing them with each other as well.

In order to solve the above problems, you first need to understand how time is actually measured. In the seventeenth century, time was measured in an astronomical way by using the sun. The event when the sun reaches its highest point in the sky is called the transit of the sun and the interval between two consecutive transits is called the solar day. As there are 24 hours in a day, with each hour containing 3600 seconds, a solar second is 1/86400th of a solar day. Figure 9.3 shows the geometry of calculating the mean solar day.

NOTES

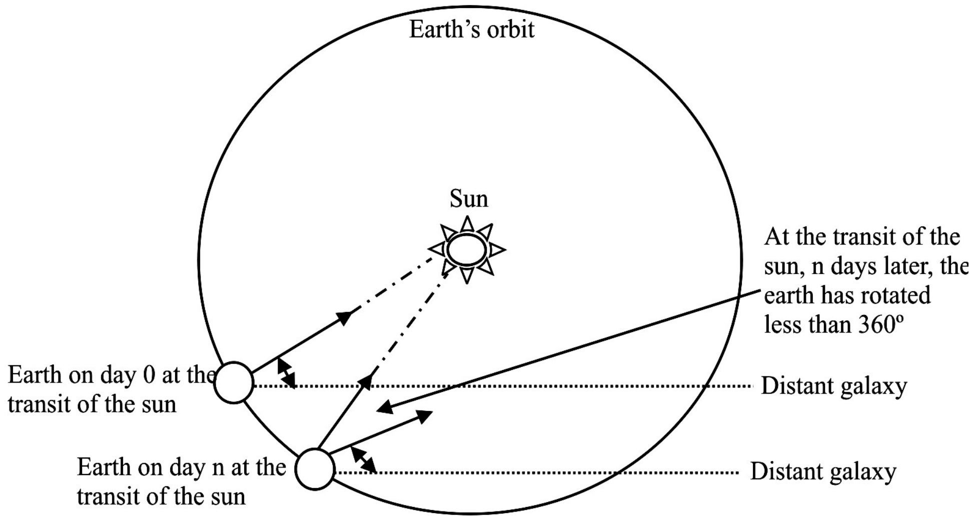

Earth's orbit
Sun
At the transit of the sun, n days later, the earth has rotated less than 360º
Earth on day 0 at the transit of the sun
Distant galaxy
Earth on day n at the transit of the sun
Distant galaxy

Fig. 9.3 *Computation of the Mean Solar Day*

However, it was discovered in 1940 that the period of the earth's rotation is not constant as the earth is slowing down attributed to tidal friction and atmospheric drag. The variations also occur in the length of a day due to turbulence in the earth's core of molten iron. Therefore, astronomers computed the length of the day by taking a large number of days and dividing it with 86,400. This measurement was called a mean solar second.

In 1948, the atomic clock was invented, which made it possible to measure time much more accurately and independent of the earth. This clock measured time by counting the transitions of the Cesium 133 atom. Now, physicists defined the second to be the time taken by the Cesium 133 atom to make exactly 91,92,631,770 transitions. Currently, many laboratories around the world have Cesium 133 clocks. Periodically, each laboratory informs the Bureau International de l'Heure (BIH) how many times their clock has ticked. BIH then produces International Atomic Time (TAI) by averaging the BIH times.

TAI is highly stable but has one problem associated with it. TAI seconds counted to 86,400 and are 3 msec less than a mean solar day as it is getting longer all the time. However, BIH solved this problem by introducing leap seconds. Leap seconds are introduced whenever time discrepancy between TAI and solar time grows to 800 msec. This correction gives rise to a new time system based on constant TAI seconds called Universal Coordinated Time (UTC). UTC remains in phase with the apparent motion of the sun. Nowadays, UTC has replaced the old time system, Greenwich Mean Time (GMT) that was based on astronomical time.

Power companies base their clocks on UTC so that when BIH announces a leap second, the companies raise the frequency of their clocks by 1 Hz for 50 or 60 seconds. Since 1 second is a noticeable time for computer, a special software is required to account for it.

The National Institute of Standard Time (NIST) operates a shortwave radio station, WWV from Fort Collins to provide precise UTC time to people. At the start of each UTC second, WWV broadcasts a short pulse, the accuracy of which is itself ±10 msec. As a result, it is difficult to obtain time with extremely high accuracy.

9.3 CLOCK SYNCHRONIZATION ALGORITHMS

Clock synchronization algorithms aim at keeping all other machines synchronized to the machine having the WWV receiver. In case no machines have WWV receivers, every machine keeps track of its own time and aims at keeping its times as close as possible. The various algorithms designed for synchronization are as follows:

- Cristian's algorithm
- Berkeley's algorithm
- Averaging algorithm
- Multiple external time sources

All the algorithms have a common underlying system model, where every machine has a timer that generates an interrupt H times a second. The interrupt handler increments one to the software clock when the timer goes off. Suppose that the value of this clock is C and when the UTC time is t, the value of time on a machine p is $C_p(t)$. In a perfect case, $C_p(t) = t$ for all p and t or you can say that dC/dt = 1.

However, real timers do not interrupt exactly H times a second. An ideal timer having H = 60 should generate 2,16,000 ticks per hour. In practice, the error in modern chips is 10^{-5}, which means that a machine will get a value in the range of 2,15,998 to 2,16,002 ticks per hour. In other way, let there be a constant ρ such that,

$$1 - \rho \le \frac{dc}{dt} \le 1 + \rho$$

Then, the timer is said to work within its specification. The manufacturer determines the constant ρ and is known as the maximum drift rate. Now, if two clocks start to drift from the UTC time in the opposite directions, at a time Δt, then they are 2ρΔt apart. Therefore, if you want that the clocks do not differ from each other by a time more than δ, then you need to re-synchronize the clocks every δ/2ρ seconds.

9.3.1 Cristian's Algorithm

Under this algorithm, one machine, called time server, is equipped with a WWV receiver, and the algorithm aims at synchronizing all the other machines with the time server. Periodically, i.e. in a time less than or equal to $\delta/2\rho$ seconds, every machine sends a message to the time server, requesting for the current time value. The time server machine responds as soon as possible with a message carrying the current time (C_{UTC}). Figure 9.4 shows how a machine gets the current time from the time server.

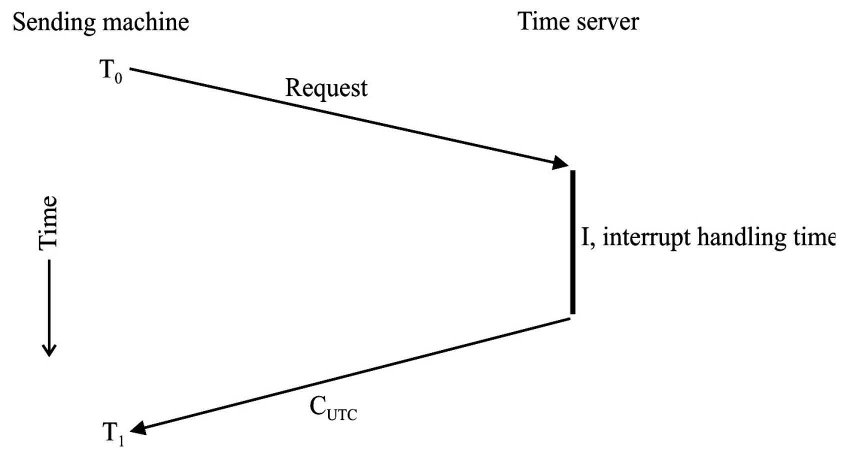


Fig. 9.4 *Getting the Current Time from the Time Server*

Now, the sender can set its clock to C_{UTC} as soon as it gets the reply. However, there are two problems associated with this algorithm. The first problem is that time must not run backward. Therefore, if the sender's clock is faster than the receiver's clock, accepting the C_{UTC} can cause serious problems, such as an object file compiled a little while after the clock has changed, having a time earlier than the source that was modified just prior to the clock change. In such cases, the change must be gradual: for example, if a timer generates 100 interrupts per second, each interrupt will ideally add 10 milliseconds (msec) to the time. So, to slow down the clock, the interrupt is set to add only 9 msecs each time till the time value has been corrected. In a similar way, you can make a slow clock fast by adding 11 msecs at each interrupt till the correct time value has been achieved.

The other problem associated with this algorithm is that it takes a non-zero amount of time for the server's reply to reach back the sender. This delay can even get longer depending on the network load. To deal with this, the computer scientist Cristian devised a way to measure it. The sender can easily and accurately measure the interval between sending the request and arrival of reply. Both these times, i.e. sending time, T_0 and arrival time, T_1 are measured with the same clock and therefore, will be accurate. When no other information is present, the propagation time is given as $(T_1 - T_0)/2$. Therefore, when the reply is received, the sender can add the propagation delay to the C_{UTC} time. However, this estimate time can be further improved if it is known how much time the server takes to handle the interrupt and process the request. Let I be the time taken to handle the interrupt. Then, the time devoted for propagation of message is $(T_1 – T_0 – I)$. Therefore, the best estimation of one-way propagation is $(T_1 – T_0 – I)/2$.

In addition, to improve the measurement, Cristian suggested taking a number of measurements, discarding the measurements that exceed certain threshold value and calculating the average of rest of the measurements. Another way can be to consider the message that came back the fastest assuming that it was not the victim of network traffic.

9.3.2 Berkeley's Algorithm

Berkeley's approach is entirely opposite to Cristian's approach, where the time server was passive. In Berkeley's approach, the time server keeps polling every machine periodically for their current time. On the basis of the reply, it calculates the average of all the times and informs each machine to advance or slow their clocks according to the new time. Figure 9.5 shows the working of Berkeley's algorithm.

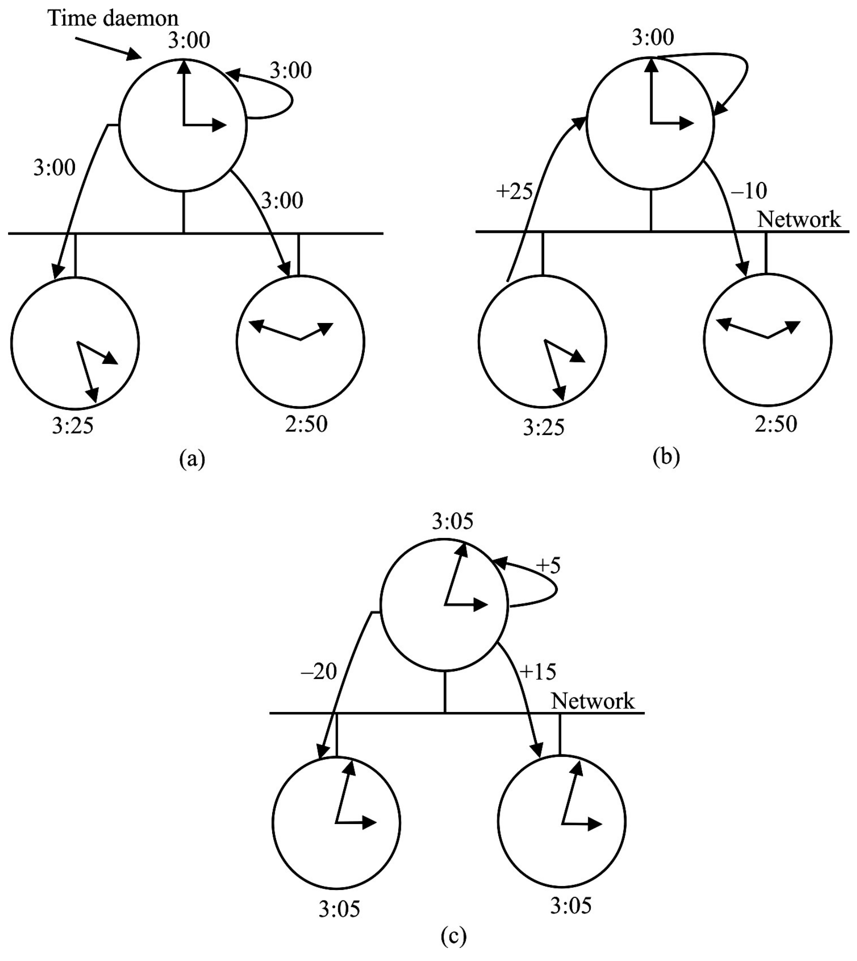


Fig. 9.5 *Working of Berkeley's Algorithm*

In Figure 9.5 (a), the time daemon sends messages to other machines at time 3:00, telling them its time and requesting them to send their time. In Figure 9.5 (b), the machines respond by sending a positive or negative value that indicates how far ahead or behind they are from the time daemon. With these values, the time daemon computes the average value of time and informs all the machines to adjust their clocks accordingly, as shown in Figure 9.5 (c). Thus, Berkeley's algorithm is suitable for a system where no machine has a WWV receiver. Also, the clock for the time daemon has to be set manually.

9.3.3 Averaging Algorithms

Both the algorithms, Cristian's and Berkeley's, are centralized algorithms with usual disadvantages, such as a single point of failure associated with them. However, decentralized algorithms are also known to compute time by averaging the time values of all the machines. One of the decentralized algorithms divides time into fixed-length re-synchronization intervals. The starting time for i^{th} interval is $T_0 + iR$ and the ending time is $T_0 + (i + 1)R$, where R is a system parameter and T_0 is some past time. All the machines broadcast their current times at the beginning of each interval. Since the clocks on different machines run at different rates, these broadcasts do not occur simultaneously.

After a machine has broadcasted its time, it starts a local timer to collect the other broadcasts during an interval S. After all the broadcasts have arrived, an algorithm, such as averaging the values, computes the new time using the values from these broadcasts. A variation for the averaging algorithm can be achieved by discarding m highest and m lowest values prior to averaging. This greatly enhances accuracy as it discards up to m number of faulty clocks from sending out the wrong timing.

Another variation for the averaging algorithm can include an estimation of the propagation time to each message. The estimate can be made from the network topology or by computing the time it takes for probe messages to be echoed.

9.3.4 Multiple External Time Sources

In systems where extremely accurate synchronization is required with UTC, the system can be equipped with multiple receivers for WWV, Geostationary Earth Orbit Satellite (GEOS) and other sources. However, as all the sources are inherently inaccurate and deflect from each other due to fluctuations in the signal path, it is best to establish a time interval in which UTC falls. Generally, different time sources produce different time ranges so that the machines attached to them need to come to an agreement.

To come to an agreement, every processor having a UTC source broadcasts its range periodically. However, no processor will receive the time packets instantly. Furthermore, the transmission-reception delay depends on the cable distance and the number of gateways through which the packets have to pass which is different for each (UTC source, processor) pair. In addition, delays due to collisions also play an important role. Additional uncertainty to time can be added if a processor is busy in handling a previous request. In such a case, the processor may not even look at the request for some time.

Check Your Progress

1. What are the properties of distributed algorithms?
2. Define a second in terms of Cesium 133 clock.
3. Name a few clock synchronization algorithms.

9.4 MUTUAL EXCLUSION

Critical regions are used to program multiprocess systems such that when a process has to read or update shared data structures, it has to enter a critical region so as to achieve mutual exclusion. This ensures that no other process uses shared data structures at the same time. In single-processor systems, semaphores and monitors are used to protect critical regions. In distributed systems, mutual exclusion is implemented using centralized, decentralized and token ring algorithms.

9.4.1 Centralized Algorithm

In a centralized algorithm, one of the many processes is elected as a coordinator. A coordinator can be a machine with the highest network address. When a process wants to enter a critical region, it sends a request message to the coordinator. The message states the critical region that the process wants to enter and asks for permission. The coordinator replies with a message granting permission, if no other process is using the critical region in question currently. Figure 9.6 shows the working of a centralized algorithm.

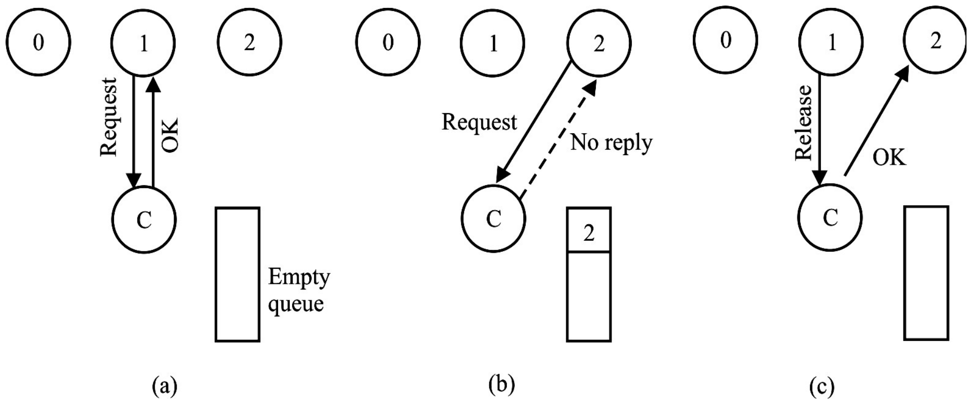


***Fig. 9.6** Working of Centralized Algorithm*

Figure 9.6 (a) shows process 1 requesting the coordinator, C to grant access to the critical region. Since the queue was empty and no other process was using the critical region, C replied granting access to process 1. In Figure 9.6 (b), process 2 requests permission to have access to the same critical region that is being accessed by process 1. So, the coordinator knows that it cannot grant permission as some other process is using the demanded critical region. Therefore, it does not reply, thereby blocking process 2, which is now waiting for the reply from the coordinator. It can also send a 'permission denied' message, depending on the implementation of the operating system.

Process 1 sends a release message to the coordinator when it exits the critical region. The coordinator then sends an OK message to process 2 granting its request to the critical region, which was previously occupied by process 1. Figure 9.6 (c) shows the connection release of process 1. The coordinator takes the first item from the queue and grants it the permission to enter the critical region.

This algorithm guarantees mutual exclusion as the coordinator allows only one process at a time to enter a critical region. It is also just because requests are granted in the order in which they are received and so there is no starvation. Also, this algorithm only requires three messages per use of critical region.

However, a centralized algorithm also has certain drawbacks. It follows a centralized approach that has a single point of failure, i.e. the coordinator. If the processes block after making a request, then they cannot distinguish between a dead coordinator and permission denied state, as no message is sent in both the cases. A single coordinator can also cause performance bottleneck in large systems.

9.4.2 Distributed Algorithm

To avoid a single point of failure, distributed algorithms were developed, such as Ricart and Agrawala. This algorithm requires a complete ordering of the events in the system. A process wanting to enter a critical region sends a message containing its process number and current time to all the other processes, including itself. The messages sent are acknowledged either singly or using group communication.

When a request message is received by a process from some other process, the action taken by it depends on the state of the process with respect to the critical region requested in the message. Three actions can be taken, which are as follows:

- If the receiver process is not using the critical region and is not interested in entering the critical region, it will send an OK message to the sender process.
- If the receiver is already in the critical region, it queues up the request and does not send any reply.
- If the receiver wants to enter the critical region, but has not entered it as yet, it compares the timestamp in the incoming message with the timestamp of the message it sent to everyone. On the basis of this comparison, the following two actions are possible:

o If the timestamp of the incoming message is lower than that of the sent message, an OK message is sent by the receiving process.

o If the timestamp of the receiver's own message is lower, the receiver queues up the request and does not send a reply.

After sending requests to enter the critical region, the process waits till it receives permission from other processes, after which it can enter the critical region. When a process exits the critical region, it sends back OK messages to all the other processes on its queue and deletes them from the queue.

This algorithm works very properly if there is no conflict. However, let's see how it works in case of a conflict, i.e. when two processes demand the same critical region simultaneously. Figure 9.7 shows the scenario when two processes have a conflict.

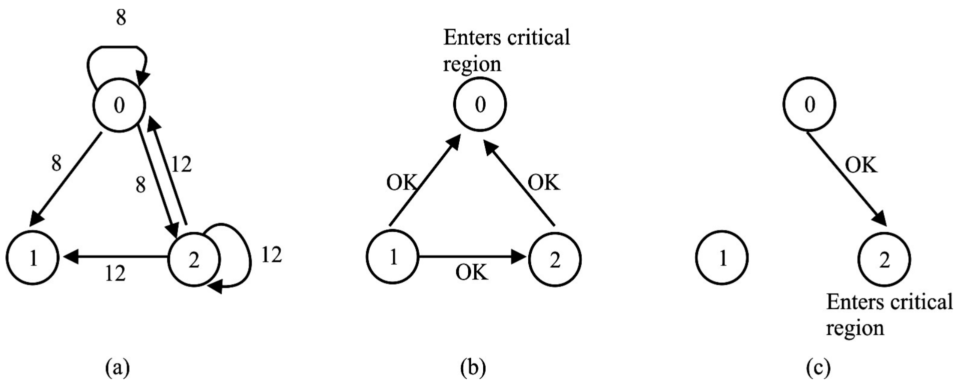


Fig. 9.7 *Two Processes trying to enter a Critical Region Simultaneously*

In Figure 9.7 (a), process 0 and process 2 send a request to every other process, including them. However, process 0 has the timestamp of 8 while process 2 has the timestamp 12. Since process 1 is not interested in entering the critical region, it sends an OK message to both the requesting senders, as shown in Figure 9.7 (b). However, process 0 and 2 find the conflict and compare the timestamps. Since process 2 has a higher timestamp than process 0, it losses and sends an OK message to process 0. When process 0 has finished its work in the critical region, it removes the request of 2 from its queue and sends an OK message to process 2. Figure 9.7 (c) depicts the scenario of process 0 sending an OK message to process 2 so that it can enter the critical region.

Thus, in the distributed algorithm 2(n-1), messages are required to enter a critical region, where a system has n number of processes. Also, there is no single point of failure. However, instead of a single point of failure, now there exist multiple (n) points of failure. This is because if any process crashes, it will not respond to the requests, which would be interpreted by other processes as the denial of permission. This will block all the attempts of all the processes from entering the critical region. However, this problem can be corrected by sending a denial message also if permission is not to be granted.

This algorithm is associated with another problem that it must use either a group communication or every process must maintain a group membership list. The group membership list must include the processes entering a group, the processes leaving the group and the processes that have crashed.

9.4.3 Token Ring Algorithm

In a token ring algorithm, there is a bus network with unordered processes. In software, a logical ring is created and each process is assigned a space on the ring. Figure 9.8 shows the bus and the logical ring structures.

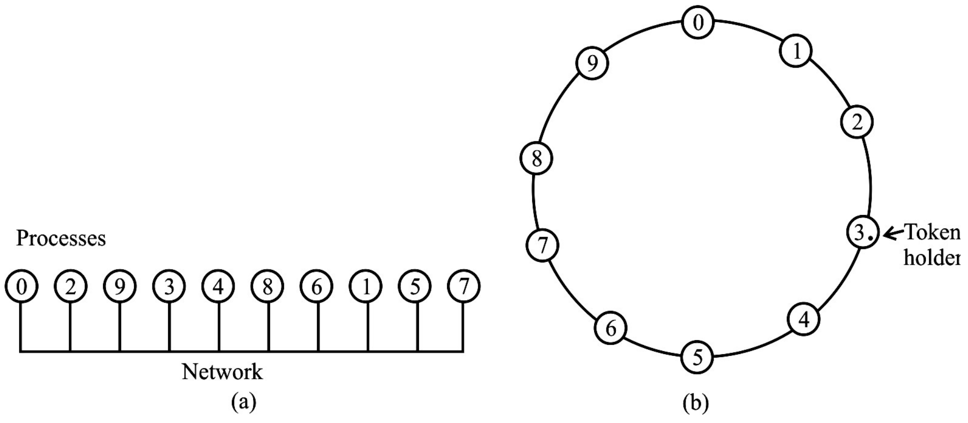


Fig. 9.8 *Bus and Logical Ring Structures used in Token Ring Algorithm*

In the token ring algorithm, the processes need to be ordered in some way, such as according to their network addresses or process ID. It is important that each process knows its next neighbouring process. Initially, process 0 is given the token that circulates around the ring. The token is passed from process k to k+1 in point-to-point messages. When a process acquires a token, it checks whether or not it wants to enter the critical region. If the process wants to enter the critical region, it enters the critical region, completes its work, exits the region and passes the token to the next process in the ring. A process cannot enter a second critical region with the same token. When no process on the ring wants to enter a critical region, the token keeps on passing from one process to another across the ring.

This algorithm ensures that only one process enters a critical region at a time as only one process gets hold of the token at any instant of time. Also, since the token circulates around the ring in a well-defined manner, no starvation occurs. At the most, a process will have to wait for every other process to enter and leave a critical region, before it gets the permission to enter the region.

However, there are certain problems associated with this algorithm too. The token must be regenerated if it is lost. Even before that, it is difficult to detect that the token has been lost because the time of successive appearances of the token on the network is unbounded. Furthermore, it is not necessary that the token is lost if it hasn't been spotted on the network for a long time. This can also mean that some process is still using it.

Another problem is that of process crashing. In such a case, the algorithm can be programmed in such a way that a process acknowledges the receipt of the token to the process from which it receives it. Therefore, when a sending process does not receive acknowledgement, it takes it like the process is dead and sends the token over the dead process to the next process.

9.5 ELECTION ALGORITHMS

As discussed earlier, most distributed algorithms require one process to act as a coordinator, distributor or sequencer. Any process can take up this responsibility, but some algorithms need to determine or elect the coordinator process. If every process is the same, with no distinguishing characteristics, there is no way to select a process. Therefore, it is assumed that every process has a unique number or ID, such as the network address. Generally, the election algorithms attempt to trace the process with the highest process number and designate it as the coordinator. It is also assumed that each process knows the process ID of every other process, but they do not know which of the processes are active (currently up) and which are currently down. The aim of any election algorithm is to elect the new coordinator and make all the processes agree to it. Various election algorithms are as follows:

- The bully algorithm
- The ring algorithm

9.5.1 Bully Algorithm

Bully algorithm was devised by Garcia–Molina in 1982. Under this algorithm, a process initiates an election when it fails to receive response from another process. The election is held by a process, P by performing the steps as follows:

1. An ELECTION message is sent to all the processes with higher process numbers than P.
2. If no process sends a response, P wins the election and becomes the coordinator.
3. If a higher numbered process answers, it takes over the job from P.

When a higher-numbered process receives an ELECTION message from a lower-numbered process, it sends an OK message indicating that it is alive and will take over. The receiver then holds an election in a similar way as explained above. Ultimately, all the processes give up, and one process that is left is the new coordinator. The new coordinator sends a message to all the other processes announcing its victory and telling them to start immediately.

If a process that was previously down comes back, it holds an election and if it is the highest-numbered process running, it will win and become the coordinator. Thus, the algorithm gets its name from the fact that biggest-numbered process wins and bullies the smaller ones; hence the name, bully algorithm. Figure 9.9 shows the working of a bully algorithm.

NOTES

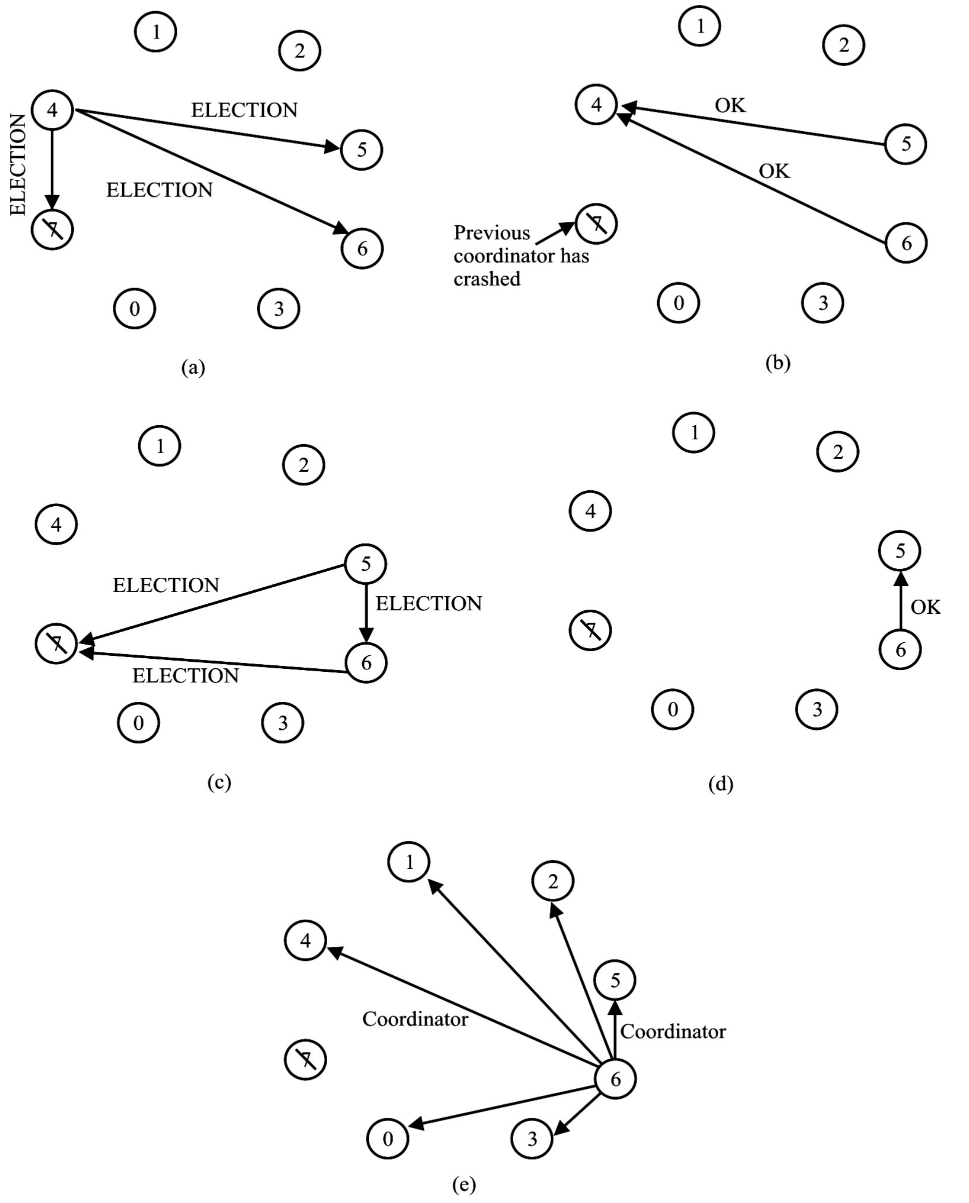


Fig. 9.9 *Working of a Bully Algorithm*

In Figure 9.9, there are eight processes (0 to 7) in a group. Process 7 was the coordinator but it has crashed and therefore, a new coordinator needs to be elected. Process 4 is the first one to notice that the coordinator has failed. Therefore, it sends an ELECTION message to all the other process, higher than it, i.e. to processes 5, 6 and 7, as shown in Figure 9.9 (a). Since process 7 was previously dead, it does not respond, while processes 5 and 6 send an OK message, as shown in Figure 9.9 (b). As soon as process 4 gets a response, it knows that its job is over, so it sits back and waits for the coordinator to be elected. As shown in

Figure 9.9 (c), both processes 5 and 6 hold the election and send an ELECTION message to higher processes, i.e. process 5 sends a message to processes 6 and 7, and process 6 sends a message to process 7. In Figure 9.9 (d), process 6 informs process 5 that it will take over and process 6 already knows that process 7 is dead as it does not receives a response from process 7. Now, process 6 knows that it is the winner and when it is ready to take over, it sends a coordinator message to all the other processes.

When process 4 receives the coordinator message, it can continue with its operation, which it was previously trying to accomplish. In case, process 7 ever starts again, it will send a coordinator message to all others.

9.5.2 Ring Algorithm

A ring algorithm uses a ring structure without a token and assumes that each process knows its successor. Also, the processes are logically or physically ordered. When a process realizes that the coordinator is dead, it builds an election message, which contains its own process number and sends the message to its successor process. In case the successor is down, the sender process keeps on skipping the processes till it finds a running process. The sender keeps on adding its own process number to the list in the message, at every step.

In a matter of time, the initiating process gets back the message and it recognizes this event when it sees its own process number in the message. At this point of time, a coordinator message is circulated to inform everyone of the new coordinator. When the coordinator message has been circulated once, it is discarded and the processes return to their work. Figure 9.10 shows the ring election algorithm.

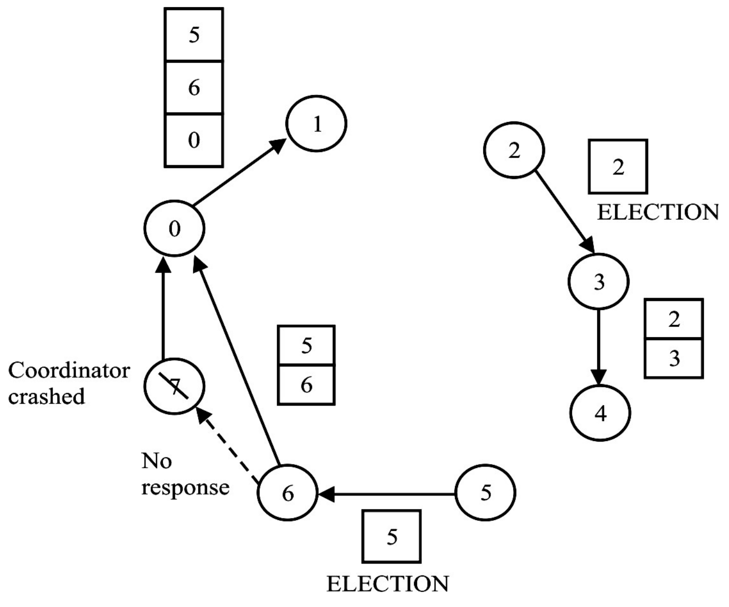


Fig. 9.10 *Ring Election Algorithm*

In Figure 9.10, processes 2 and 5 simultaneously discover that the coordinator has crashed. Therefore, both these processes build an ELECTION message each, with their own process numbers and circulate it. In the end, both the messages will go round the ring and processes 2 and 5 will convert them into coordinator messages. Thus, in this way an extra message will circulate, which is of no harm. It just uses some extra bandwidth.

9.6 DEADLOCKS IN DISTRIBUTED SYSTEMS

Deadlock is a situation in which two or more processes attempt to access a resource that is locked by another process. Deadlocks in distributed systems are similar to deadlocks in single-processor systems. There are two kinds of distributed deadlocks: communication deadlocks and resource deadlocks. A communication deadlock is a situation in which each member process is trying to communicate with another member process but is unable to communicate as they both wait for each other to answer a query. Resource deadlock is a situation in which member processes are arguing over exclusive access to I/O devices, files, locks or other resources. The various strategies that are used for handling deadlocks in distributed systems can be classified into four major groups: ostrich algorithm, deadlock detection, prevention and avoidance. Ostrich algorithm is popular in distributed systems as it is in single-processor systems. In distributed systems, it is used for programming, process control and office automation. Deadlock detection and recovery techniques allow deadlocks to occur and then apply certain methods to recover from the deadlock. These techniques are very difficult to implement. Due to the presence of atomic transactions, deadlock prevention is also possible in distributed systems.

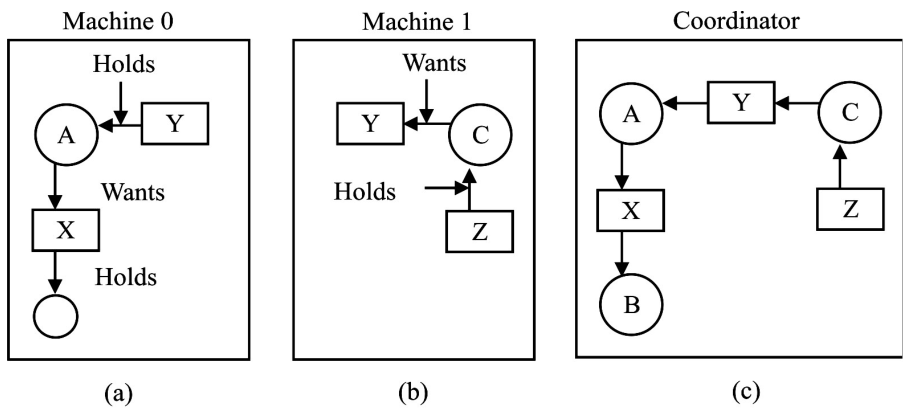
Finally, the deadlock avoidance techniques acquire information in advance about which resource a process will claim at which stage of execution. Distributed operating system can assume that deadlock will never happen or rarely occur and fully ignore it.

9.6.1 Distributed Deadlock Detection

Since it is very difficult to find out the methods for preventing or avoiding distributed deadlocks, researchers have started dealing in detecting the occurrence of deadlocks in distributed systems. Deadlock detection detects the state of the system to determine whether a deadlock has occurred or not. It also helps processes to recover from the deadlock condition. The presence of atomic transactions in some distributed systems make a major conceptual difference. If a deadlock is detected in a conventional operating system, you can break the deadlock by killing one or more processes. When a deadlock is detected in a system, which is based on atomic transactions, then abort all the deadlocked processes. This means, one of the deadlocked processes is aborted and it is verified whether the deadlock is over or not. This procedure is continued until a system reaches safe state.

Centralized deadlock detection

In a centralized deadlock detection algorithm, each machine has a resource graph for its own processes and resources and the deadlock detection coordinator maintains the resource graph for the entire system. When the coordinator detects a cycle, it destroys one process in order to break the deadlock. In distributed systems, due to delay in information many deadlock algorithms generate false deadlocks. Figures 9.11 (a), (b), (c) and (d) show the different processes running on different machines.



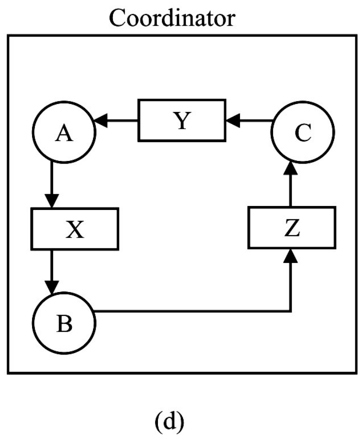


Fig. 9.11 *Different Processes running on Different Machines*

In Figure 9.11, processes A and B are running on machine 0 and process C is running on machine 1. X, Y and Z are the three resources that are used by these machines. In Figures 9.11 (a) and (b), process A holds resource Y but it wants resource X, which is being used up by process B. Process C holds resource Z but it wants to use resource Y. Figure 9.11 (c) shows the coordinator's view of the world. When process B is finished, resource X is used by process A, which in turn releases the resource Y for process C.

After some time, when process B releases resource X, it requests for resource Z, which is perfectly legal and safe swap. Machine 0 then sends a message to the coordinator to release resource X and machine 1 sends message to the coordinator that process B is waiting for its resource Z. However, the coordinator receives the first message from machine 1 leading the coordinator to create the graph as shown in Figure 9.11 (d). Here, the coordinator is mistaken that a deadlock exists and therefore, destroys some process. This situation is called false deadlock.

NOTES

Chandy–Misra–Haas algorithm

In the Chandy–Misra–Haas algorithm, the processes request for multiple resources at once due to which there is an increase in the growing phase of transaction. But the result of this change in model is that the process has to wait on more than two resources at the same time. Figure 9.12 shows the Chandy–Misra–Haas algorithm.

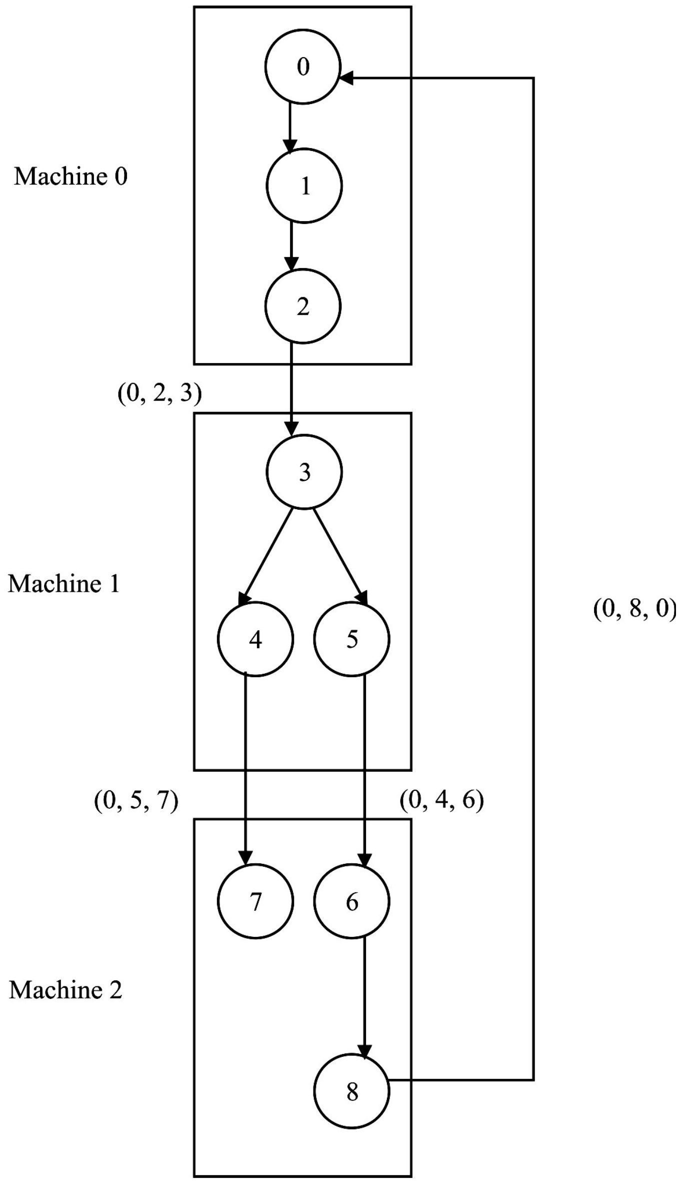


Fig. 9.12 *Chandy–Misra–Haas Algorithm*

In Figure 9.12, process 3 residing on machine 1 is waiting for two resources where one resource is held by process 4 and another by process 5. In this algorithm, when a process waits for some resource, a special probe message is created, which is then sent to the process containing the required resources. This message consists of three numbers: the process that is blocked, the process that is sending the message and the process that is receiving the message.

When the message appears, the recipient verifies whether it itself is waiting for any process or not. In case the recipient is waiting for any process, then the message is updated by keeping the first number unchanged and replacing the second and third numbers by their corresponding process numbers. The message is then sent to the process containing the required resources. After travelling all the way, if the message comes back to the original sender that initiates the probe, then a cycle exists and the system is deadlocked.

Check Your Progress

4. Name a distributed algorithm for achieving mutual exclusion.
5. Mention some problems associated with the token ring algorithm.
6. What is the principle of bully algorithm?

9.7 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Distributed algorithms are characterized by the following properties:
   A. The relevant information is distributed across multiple machines.
   B. The processes make decisions only on the basis of local information.
   C. You should avoid a single point of failure in the system.
   D. A common clock or some other source of precise global time must exist.
2. A second is defined as the time taken by Cesium 133 atom to make exactly 9,19,26,31,770 transitions.
3. Some of the clock synchronization algorithms are as follows:
   A. Cristian's algorithm
   B. Berkeley's algorithm
   C. Averaging algorithm
   D. Multiple external time sources
4. Ricart and Agrawala is a distributed algorithm for achieving mutual exclusion.

5. Some problems associated with the token ring algorithm are as follows:
   A. It is difficult to detect a lost token.
   B. If lost, the token must be regenerated.
   C. Process crashing
6. Bully algorithm is based on the principle that the highest-numbered process is elected as the coordinator.

9.8 SUMMARY

- Synchronization is very important in distributed systems. Since distributed systems make use of shared memory, it is important to achieve proper communication between the processes.
- Some systems require the processes to be synchronized with each other, irrespective of the real time. Such processes are said to follow absolute time and implement logical clocks.
- Systems that require to be synchronized with real time implement physical clocks. A physical clock nowadays remains in sync with the UTC that has replaced the GMT.
- Clock synchronization algorithms have been developed for synchronizing the clocks with each other and with the BIH or UTC time. Some algorithms, such as Cristian's, involve a passive time server called time daemon, whereas, some others, such as Berkeley's, involve an active time server.
- Mutual exclusion in distributed systems can be achieved either through centralized algorithms that make use of a coordinator process or through distributed algorithms like Ricart and Agrawala.
- Centralized algorithms make use of a coordinator and other algorithms also require one process to act as the leader or serve a particular responsibility, election algorithms are needed to choose this process. Since all these algorithms require you to get highly involved with the intricacies of the processes and the system, transactions were introduced as a clean and efficient way of synchronization in distributed systems.
- Deadlock detection involves detecting the deadlock and trying to recover from it. Deadlock prevention involves making deadlocks structurally impossible, whereas deadlock avoidance involves avoiding deadlocks by carefully allocating the resources.

9.9 KEY WORDS

- **Timer**: A timer is a machined quartz crystal, with a number of crystals which, when kept under tension, oscillate at a well-defined frequency.

- **Transit of the sun**: The event when the sun reaches its highest point in the sky is called the transit of the sun.

9.10 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. What are the properties of distributed algorithms?
2. Explain Lamport's algorithm for assigning time in logical clocks.
3. What is the centralized algorithm for mutual exclusion?
4. Trace the differences between the three mutual exclusion algorithms.

Long Answer Questions

1. Explain the concept of clock synchronization. Explain any two clock synchronization algorithms.
2. Explain token ring algorithm for mutual exclusion.
3. Explain Bully algorithm.

9.11 FURTHER READINGS

Tanenbaum, Andrew S. and Maarten Van Steen. 2006. *Distributed Systems: Principles and Paradigms*. New Jersey: Prentice Hall.

Garg, Vijay K. 2002. *Elements of Distributed Computing*. New Jersey: Wiley-IEEE Press.

Sinha, Pradeep K. 1996. *Distributed Operating Systems: Concepts and Design*. New Delhi: Prentice-Hall of India.

BLOCK - IV
DISTRIBUTED FILE SYSTEM

NOTES

UNIT 10 INTRODUCTION TO DFS

10.0 INTRODUCTION

In this unit, you will learn about the distributed file system (DFS). The DFS has the facility of sharing the files among all the clients who are authorized to access the content and a client can store the information locally as well. The distributed environment allows all the computers or nodes share the data stored as per the requirement and can be stored locally or on the server as the access rights given to a node within the distributed computing environment.

10.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain the distributed file system
- Discuss the desirable features of DFS
- Understand the various types of file models and file assessing models

10.2 DFS

Distributed File Systems (DFSs) are designed to support the sharing of information across a network in the form of files. If the system is designed well then it provides the same reliability and performance in accessing the files stored at the server as files stored on some local host. In computing, it allows the programs to store and access files remotely from any computer in an intranet. Thus, multiple users on multiple computer systems can share files and resources. The client systems can access the storage over the network using its protocol. The design of the protocol restricts access to the file storage depending on the access lists on both the server side and client side. All the machines have equal access to the storage when the file system is located in shared disk and the access control on these systems resides on the client. Distributed file systems must cover the facilities of scalability, fault tolerance and replication. In distributed file systems, files are stored on one or more systems which are called servers and these files are accessible by various systems which are clients. It is easy to provide backups and security of the data in the distributed file systems as it needs only servers to be secured and backed up. There are a lot of problems in designing a proper distributed file system. Accessing many files over the network may create network bottlenecks and server overload which may slow down the performance of the network. Security of the data is also an important issue. Network and server failures may also create problems in designing a distributed file system. Most often, client computers are more reliable in comparison to the network connecting them.

Some services, such as authentication service, printing service and name service can easily be implemented in DFS or Distributed File System. In the organizations where Web servers are used for external and internal access through intranet, these Web servers access and store data from a distributed file system.

In distributed object oriented programming, there is a requirement for the storage and distribution of the shared objects. This can be achieved by serializing the objects and storing and retrieving the serialized objects using the files. But if the objects change rapidly then this will not work and therefore, some more direct approaches have been developed. Java RMI and CORBA provide access to remote, shared objects but they do not ensure the persistence of the objects.

The current developments include Distributed Shared Memory, i.e., DSM and persistent objects stores for the distribution of the distributed information. DSM copies segments or memory pages at each host which provides an emulation of the shared memory. For optimizing the performance of the distributed programs all storage systems depend on caching. When the replicas of the memory pages or segments are used then it is difficult to achieve strict consistency. The Web uses caches both at the client and the proxy server sides. The explicit user actions maintain consistency between the replicas stored at original server, proxy server and the client caches. A client must keep the local copies of the data up to date explicitly as the updation on the server side is not notified to the client side.

10.3 DESIRABLE FEATURES

The desirable characteristics of a distributed file sharing environment are listed below:

(i) Transparency: In a distributed file sharing environment, a client can store a file or data at a centralized location which can be a server or a memory

NOTES

storage device mapped to the server or other clients designated for storing data by the distributed operating system kernel. The storage process is either initiated by a client with a user process or by a kernel of an operating system to store the data or file. However, the process of storage and the storage media along with location should be transparent in order to allow all the corresponding nodes which want to store information. A client node stores the data at a centralized location where a server can store the data or file at one location or more than one location in order to ensure data integrity and security of data. The characteristic of providing transparency to the clients related to storage mechanism and the location where data is stored is also known as structural transparency.

A client while retrieving a file or data should be able to access the same irrespective of whether a client is requesting the access remotely or locally. The transparency in the access mechanism is also known as access transparency. This characteristic enables a client to access a data using any of the two methods without being dependent of the mode of accessing the data within a distributed file sharing system.

A client should be able to copy or move a file stored in a distributed file sharing environment from one node to another without making any changes to the names of a file. This allows transparency in the naming structure within a distributed file sharing environment and the same is also known as naming transparency. In case a client replicates a file or data at different nodes, the distributed file sharing environment should not make the details of all the copies available and the storage locations to any clients providing transparency to the replication procedures and policies within a distributed file sharing environment.

(ii) **Portability**: The distributed file sharing environment should be designed to allow users to store data or files while working on any nodes which is connected the network of a distributed operating system in order to provide better flexibility. This feature provides portability to a user in order to access the data or file from any node which is connected to the network of distributed operating system.

(iii) **Scalability**: The distributed file sharing environment should provide better flexibility by adapting the new developments in order to be scalable for futuristic use. The technological advancements need to be implemented by developing new routines which need to be incorporated in the distributed file sharing environment. Any increments or incorporation made in the distributed file sharing environment should not reduce the performance like time taken to respond to the request, load balancing, data storage and retrieval time taken.

(iv) **Performance**: The performance in a distributed file sharing environment is measure based on the turnaround time that a client is required to wait from the time of initiating a process for retrieval or storage of data and receiving the resources from the node where data is stored. The turnaround time includes the time taken while communicating the request from client to server and retrieving the file or data from the node where data is stored.

(v) Ease-of-Use: The user interface provided should be easy to use in order to allow a user to understand the syntax and the method of writing a request should be simple to understand to the users. The design of a distributed file sharing environment should have relevance with the conventional file systems in order to make the system easy to use at user level.

(vi) Reliability: Any operating system when used may encounter failures at any time, however, a good operating system is one which will handle the failure situations effectively and mitigate the impact of failures by ensuring that the system doesn't stop working. Similarly a reliable distributed file sharing system may also encounter failures as discussed in chapter 2 of this book where failures related to message passing where discussed in details and the how to handle to failure situation were also discussed. A good file sharing system will ensure that the failures will be handled efficiently and effectively by ensuring the integrity, correctness and completeness of data within a distributed operating system.

(vii) Security: A distributed file sharing environment should provide better security to the data stored on any node in order to ensure the privacy of data. This feature is also proportional to the reliability of a distributed file sharing environment. Any unauthorized users should not be allowed access to the data on the network within a distributed operating system. However, only authorized users should be allowed to access the data stored at a node in a distributed file sharing environment. Similarly only authorized users should be allowed to modify or edit the data in order to ensure privacy of data which will result in increase in the reliability quotient of a system.

(viii) Data Integrity: In a distributed file sharing environment more than one users may be accessing same data or file simultaneously which will lead to the data consistency problem. For example if two users are accessing same data or file and one of the users modifies data and saves the changes but before saving the changes another user is reading the data from the data store which will fetch information which is not updated. In order to resolve this problem more than one user are allowed to read a file simultaneously but when a write instruction is to be executed, the same is given by activating locks in order to block any other write request from another process within a network. Once the write operation is completed read requests are executed to provide updated data or file to the end-user.

(ix) Heterogeneity: At present different technologies have been developed which work on different file systems and platforms which results in a challenge of data migration from one platform or file system to other. In order to

NOTES

address this problem a distributed file sharing environment should facilitate the transfer of data from one file system to another without any hassles. This increases the ease-of-use level and reliability of a distributed file sharing environment. Similarly the storage device used at one node may vary when compared with another node where the storage media can be different. A distributed file system should have the facility to store data or file on different storage media without compromising on the performance of the system. This feature of facilitating the storage & retrieval of data across platforms is also known as heterogeneity.

Check Your Progress

1. What is the significance of DFS?
2. What do you understand by the structural transparency?

10.4 FILE MODES

The basic file model is based on the structure used in a file system within a distributed file sharing environment and the same are categorized as structured and unstructured files. The second categorization of file models within a distributed file sharing environment is based on modifiability are mutable and immutable files.

(a) **Structured File System:** In this file system, the details of a file are known to the storage server where the files are stored. Every file in the system is a collection of records which are in an ordered sequence. The record is the lower most unit is this file system which can vary in size from one file to the other. The sharing of files in this file system are not as simple as unstructured file system. The structured file system is used using two methods as indexed and non-indexed method. The indexed file system stores the records in an indexed sequence where any record can be accessed by specifying the value of one or more key field and the same can be addressed by giving the values of the key fields. In structured file system method the records of a file are maintained using structures and one of the commonly used architecture is B-Tree. However, in non-indexed file system a record is traversed by mentioning the position of a record in a file. For example if the 8th record in a file needs to be accessed the traversing is executed by exploring the first record of the file first and then moving to eighth record in order to retrieve the data from the storage server.

(b) **Unstructured File System:** This file system is the simplest form of a file system where the details about a sub structure are not known to the storing server where the files are stored. The distributed operating system kernel does not require to know about the sub structures of or details of file and data stored on the storage servers which gives applications the access to understand the details of the sub structures of the data stored. Some of the examples where this method of storage was used are MS-DOS and UNIX.

Another categorization of file models based on modifiability are given below:

(a) Mutable Files: Once a file is stored on a storage server, the possibility of modifying the contents of a file is required. When the content of a file are modified the file is not re-created rather the existing file is updated by overwriting the existing file with a new file. This method is known as mutable file method. This method is commonly used in most of the operating system at present as the method reduces the overheads required to manage the number of in case every modification request re-creates a file again and again.

(b) Immutable Files: In this file system the contents are modified by re-creating a file. In this method every modification requests will create a new file and the information about the previous versions of a file is stored as history of the file modified. A separate subroutine is used for managing the versions of a file which helps the operating system to find out the recent updated file. This increases the load on the operating system and increase the quantum of storage space required to store a file.

10.5 FILE ACCESSING MODELS

The file accessing modes are the methods of accessing a file within a distributed file sharing environment and the list of the generally used file accessing modes is given below:

(a) Accessing Remote Files: In order to access a file in a distributed file sharing environment different modes are used one of them is access remote files method where a file is accessed remotely from any location which is having access to the network. The two different methods of Access Remote Files are given below:

(i) Remote Service model: In this method of file access the client requests for file access and the file is sent to server. The server processes the client's request and the output is sent to client. The file is not sent to the client node rather the file is delivered to the server. The server processes all the requests and the output is deliver in the form of a message to the client. The communication between the client and the server is in the form of data packets. In this file access mode method the communication overheads & message overheads are more while communicating a request from a client to server and while communicating the output from a server to the client. The protocols for communication and file access need to be designed appropriately in order to minimize the overheads generated by communication and messages.

NOTES

(ii) **Data caching model:** This model of file access mode helps in reducing the load of communication channel in comparison with Remote Service Model method of file access. In this model when a client has a file access requests, the availability of file is first checked locally on the node and if the file is not available locally then a copy of the file from the server is stored locally on the client node. The client node processes the files access request locally and generates the output from the locally stored file. The file is stored locally in the cache of the client node. However, a file can be stored locally at one or more client nodes simultaneously but while updating the file on the server all the nodes have to perform write operation. Therefore, the write operation will lead to increase in overheads as the write operation from more than one client needs to be managed by ensuring the integrity and consistency of data stored. The bottleneck in this file access mode is to ensure that the data stored on the cache of client node is consistent at the server node as well is also known as cache consistency problem.

(b) **Unit of Data Transfer:** The data caching model of file access mode a copy of the file from the server is stored locally on the client node cache. However, the size of the data packet transferred needs to be fixed in the design of the operating system. In this file access mode the categorization is done based on the unit of the size transferred and the same are given below:

(i) **File Level Transfer Model:** In this file access mode type when a client requests for access to a file from the server, the complete file is transferred from the server to the client. This method reduces the file access at the server node and improves the performance and scalability of the distributed file sharing environment. However, this method requires more storage space in order to store a complete file on the local storage of the client node. In case the client node does not require the access to complete file then the storage of complete file will lead to wastage of storage space resource.

(ii) **Block Level Transfer Model:** In this file access mode type when a client requests for access to a file from the server, the file blocks are copied from the server to the client. The file blocks are arranged in contiguous blocks where the size of the file block is fixed. However, in some cases the file block size is equal to the page size of virtual memory which is known as page level transfer model of file access. The storage space required at the client side is not more as compared with the file transfer model of file access. However, this method also increases the communication overheads as the traffic on the network will increase when a file is transferred in file blocks resulting in degradation of performance.

(iii) **Byte Level Transfer Model:** In this file access mode type when a client requests for access to a file from the server, the data transfer happens in bytes which are copied from the server to the client. This file access method is more flexible in comparison with the previous methods of file access. However, the cache management becomes more difficult as the number of bytes in a file are more in number.

(iv) **Record Level Transfer Model:** As discussed earlier, about structured file models where a file is a collection of records. Therefore, this file access mode will transfer the data of file in unit of records where a complete record will be transferred from the server node to the requesting node. This method of transfer is more suitable for structured file access method.

Check Your Progress

3. What are the two types of file models based on modifiability?
4. What are the two models for accessing a remote file in DFS?

10.6 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Distributed File Systems (DFSs) are designed to support the sharing of information across a network in the form of files. If the system is designed well then it provides the same reliability and performance in accessing the files stored at the server as files stored on some local host.
2. The characteristic of providing transparency to the clients related to storage mechanism and the location where data is stored is also known as structural transparency.
3. There are two types of file models based on modifiability i.e. mutable and immutable.
4. The two types for accessing a remote file in DFS are:
   (i) Remote service model
   (ii) Data caching model

10.7 SUMMARY

- Distributed File Systems (DFSs) are designed to support the sharing of information across a network in the form of files. If the system is designed well then it provides the same reliability and performance in accessing the files stored at the server as files stored on some local host.

NOTES

- Distributed file systems must cover the facilities of scalability, fault tolerance and replication. In distributed file systems, files are stored on one or more systems which are called servers and these files are accessible by various systems which are clients.
- There are a lot of problems in designing a proper distributed file system. Accessing many files over the network may create network bottlenecks and server overload which may slow down the performance of the network.
- The distributed file sharing environment should be designed to allow users to store data or files while working on any nodes which is connected the network of a distributed operating system in order to provide better flexibility.
- The basic file model is based on the structure used in a file system within a distributed file sharing environment and the same are categorized as structured and unstructured files. The second categorization of file models within a distributed file sharing environment is based on modifiability are mutable and immutable files.
- The two types for accessing a remote file in DFS are: remote service model and data caching model.

10.8 KEY WORDS

- **Distributed File System (DFS):** It is a method of storing and accessing *files* based in a client/server architecture.
- **File Accessing Modes:** These are the methods of accessing a file within a distributed file sharing environment.

10.9 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. What do you understand by the distributed file system?
2. List the various types of file models.

Long Answer Questions

1. What are the desirable features of distributed file system? Explain.
2. What are the various types of file models? Explain.
3. Explain the various types of file access models in distributed environment.

10.10 FURTHER READINGS

NOTES

Tanenbaum, Andrew S. and Maarten Van Steen. 2006. *Distributed Systems: Principles and Paradigms*. New Jersey: Prentice Hall.

Garg, Vijay K. 2002. *Elements of Distributed Computing*. New Jersey: Wiley-IEEE Press.

Sinha, Pradeep K. 1996. *Distributed Operating Systems: Concepts and Design*. New Delhi: Prentice-Hall of India.

NOTES

UNIT 11 FILE SHARING AND REPLICATION

11.0 INTRODUCTION

In this unit, you will learn how a file is shared in a distributed file system. Various semantic models have been introduced such as UNIX semantics and immutable files that specify the manner in which files can be shared. A file system in a distributed operating system is assigned the work of storing, controlling and managing data, which is stored on the disks in the form of files. The management of a file system helps to maintain the consistency of data when multiple users access the files simultaneously.

11.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain the concept of files and a file system
- Discuss the semantics of file sharing
- Understand the different mechanisms of DFS
- Explain the structure of directory in DFS

11.2 BASIC CONCEPT OF FILE

Files are stored permanently on secondary storage devices, such as a hard disk. The file system is a part of the distributed operating system that is responsible for controlling the secondary storage space. It provides a uniform logical view to the users. The various functions of the file management system are as follows:

- It enables the users to give user-defined names, to create, modify and delete files.
- It maps the user-defined names with low-level identifiers, which are the names in machine understandable format. Low-level identifiers help to identify a file uniquely.
- It provides a uniform logical view of data to users rather than a physical view, i.e., internal structure by giving a user-friendly interface.
- It controls the transferring of data blocks between secondary storage and main memory and also between different files.
- It provides semantics or file sharing rules among multiple processes and users.
- It allocates and manages space for files on secondary storage devices, such as disks or magnetic tapes. Space management is an important function of file management system.
- It protects the files from system failures and applies measures for recovery and backup of the files.
- It provides security measures for confidential data, such as electronic funds or criminal records.
- It also provides encryption and decryption facilities to the users. Encryption is a mechanism of converting data into some format that cannot be read by everyone except the recipient.

The file system is implemented as a layer of distributed operating systems and is placed between the kernel and the memory manager. It consists of utility programs which run as constrained applications that are used to control the access on files. The users can access files through the file system only.

11.2.1 File Attributes

A file attribute is required by a file system to manage a file. It is the information about a file that is linked with every file. Some of the attributes, such as file location, name and date and time of creation of the file, are unique for each and every filein the disk. Among them, only few attributes are accessible to the users, such as access privileges, name or size of a file; whereas some of them are specifically assigned to a file for the file system usage.

File attributes vary from one operating system to the other, but some of them are common to each and every operating system. Every file can contain different attributes, which are as follows:

- User-defined attributes
- System-defined attributes

User-Defined Attributes

User-defined attributes contain those file information, which are specified by a user. The different types of user-defined attributes are as follows:

- **File name**: It is an identifier chosen by a user to address the file. Usually, a string of alphanumeric characters and some operating systems allow the use of special characters, such as #, *, or $ in file names and must be unique in its file directory.
- **File type**: It refers to the type of information stored, i.e. binary file, text file, picture file or program file.
- **Owner**: It refers to the name of the creator of a file that controls and provides access privileges, such as read only or read-write to other users.
- **Permitted privileges**: It contains information that determines read privileges, write privileges and execute privileges.

System-Defined Attributes

System-defined attributes contain the file information, which are not specified by a user. An operating system automatically stores some of the information related to the creation and storage location of the files. The different types of system-defined attributes are as follows:

- **Location**: It is the address of the sector or the memory area where the file is stored on the disk. Usually, this information is used as a pointer, which is a value used by programs to find the location of certain file.
- **Low-level identifier**: It is a computer understandable name, usually in binary digits, that is used by hardware to identify a file by mapping it with the user-defined name. This attribute also consists of numbers.
- **File size**: It is the current size of a file in bytes, words or blocks.
- **Allocated size**: It is the total allocated size of a file by the file system.
- **Creation date**: It contains the date and time of file creation.
- **Date of last modification**: It contains the date and time of the last updation, insertion and deletion in a file.

All the attributes of a file are stored in the directory where the file exists. The storage of attributes takes more than 1 KB space per file. Some of the attributes, such as file size or creation date are stored in the header record associated with each file.

11.2.2 Semantics of File Sharing

When two or more users share a file at the same time in a network, it is called file sharing. In a single-processor system, such as UNIX, the read operation follows the write operation and the value that is returned by the read operation is the value that is just written. Similarly, when two write operations are executed in a sequential order after the read operation, then the value read is the value that is stored by the last write. This kind of model is called UNIX semantics. This model is easy to understand, as all the updates are immediately visible when the operation is performed. In the distributed system, UNIX semantics can be accomplished easily as there is only one file server and the clients also do not cache files. All the read and write operations issued by the processors are directly transferred to the file server to be executed sequentially in a program. This will cause performance problem in a distributed system, as the clients have to go to the server for every single operation. The performance problem can be solved with the help of cache caching in which one client can modify the cached file and another client reads the file from the server. The second client will get an obsolete file because the file has been locked for modification by the first client.

One way to solve this problem is by making all the changes to the cached file back to the server. Another solution is to reduce the semantics of file sharing. If it is required to read the effects of all the write operations, the user can use one rule, which states that the changes to an open file are initially visible only to the process that modified the file. When the file is closed, the changes are visible to other processes also. When the user closes the file, it sends a copy back to the server so that the reads command can take a new value. This semantic rule is implemented and is called session semantics. When a file is opened, it is fetched from the server and is placed in cache on the local disk. All reads and writes are then carried out on the cache copy. When the file is closed, it is uploaded back to the memory.

Another approach to the semantics of file sharing in a distributed system is to make all the files immutable. It means that the files can be opened only for creating and reading operations. The file cannot be opened for modifications. If it is required to modify a file, then the user has to create a new file or directory under the previous existing file.

Atomic transaction is yet another approach to the semantics of file sharing in the distributed system. In this approach, the process first executes BEGIN TRANSACTION primitive to signal that all the operations should be executed indivisibly for accessing a file or a group of files. After completing all the tasks, an END TRANSACTION primitive is executed. If more than two transactions are performed at the same time, then the system makes sure that the final output is the same as if they were all running in the some sequential order.

11.3 MECHANISMS OF DFS

The distributed operating system uses various mechanisms for enhancing the speed of Input/Output and CPU performance. Using these mechanisms, a user can overcome the bottleneck of centralized file system that occurs when the file server of a centralized system is heavily loaded with a large number of requests. The various mechanisms used by the distributed file system include naming schemes, remote file access or caching files and file replication.

11.3.1 Naming

Every object, which includes files, has a name associated with it. These names are referred by the end users and are mapped with low-level machine understandable identifiers that are used for identifying a file. Low-level identifiers are made up of binary digits, 0 and 1.

Naming service refers to the process of mapping user-defined file names with transparency. It is used when the users do not know the location of a file: for example, if the clients access a remote file with the help of naming service, it appears to them that the file is located on their own computer. In a distributed system, naming service is of two types, which are as follows:

- **Multi-level mapping**: It is the mapping of names in hierarchy. In this scheme, a file is identified by the user with the help of its textual name. The machine, on the other hand, identifies a file by machine level or low-level identifiers. Now, in order to find a file, these user-defined names and low-level identifiers should be mapped with each other. The mapping of user-defined names and low-level identifiers with the location of file in the network is known as multi-level mapping.
- **Multi-valued mapping**: The concept of multi-valued mapping arises with the replication of files. In case of distributed systems, the resources are shared to satisfy the simultaneous requests. The files are replicated on different servers and are controlled by distributed operating system. The textual name is sent by the user to access a file. This user-defined name is then mapped with the names of all the replicated files and the mapping returns different values. After mapping, the users are provided with a set of derived values that denote the different locations of all the file replicas on the network.

The objectives of a distributed system that are required before implementing the naming services are stated below:

- **Name transparency**: It refers to the hiding of actual location of a file to make it look like the file is located on the local disk.
- **Location independence**: It specifies that the user-defined name cannot be changed by changing the location. This name remains the same and is used

frequently by an end user irrespective of the file location. This is the most important feature of naming service as it allows mapping of the same user-defined name of a file with different locations at different times.

- **Replication transparency**: It specifies that an end user does not see any duplicate files. However, when the end user works on any duplicate file, it seems as if he is working on the original file.
- **Adapt accordingly for change in file location**: The naming service should be capable of handling the address changing dynamically by the distributed system.

Approaches to Naming Schemes

The distributed system can use any of the three approaches to implement the naming service. A proper implementation of naming service is very important for an effective distributed system network. The three approaches to the naming scheme are as follows:

- In the first approach, the file name is made up of two parts: local name and location address. The following code shows the two parts of a file name:

  ```
  <Local-name:  Path>
  ```

  This approach is not useful because it does not provide transparency, as the location of file in the network is visible to the user.
- In the second approach, the remote directory tree structure of a file system is mapped with the local directory tree structure of the file system. This mapping is performed to provide transparent access to the users.
- In the third approach, a global name structure is provided to all the files and directories that cover all the types of files and directories present in the file servers.

The various ways that are used to implement naming services are as follows:

- **Static map**: It is a very simple mechanism, which is used where the location of a file or a directory does not change. It does not support resource or file migration because it statically binds the resource name with its address. It maps the resource name with the location of the resource. Every host in the network maintains static maps.The structure of a static map is shown below:

  Hostname: Resource address

  For example:

  Site s/Host 9: D:/Bin/prog
- **Broadcasting**: In this mechanism, the desired resource name is broadcast to all the servers and the server that has the required resource or file gives reply to the host. Broadcasting is the best way to save the cost incurred on maintaining the server names or making table names.

Broadcasting does not require any global table name as the name of the resources is broadcast. Hence, it saves the cost of maintaining global table name. The main problem with this mechanism is that it provides very poor performance because every name server has to search its local table name for finding the required resource. Also, the servers that do not have the desired resource have to bear the cost in terms of time for searching the table name.

- **Name servers**: These are the special servers, which are installed on a network to control the naming service. These servers return the host address after receiving the request for resource location.

  The name servers can support resource migration because they are capable of decoupling the names. But the main problem with these servers is that there is degradation in their performance because these servers cause communication delay during installation.

- **Prefix table**: It is a combination of static tables and broadcasting names of files or resources. Every host maintains a table related to a complete path or partial path of all the resources, which it has accessed in the past. The host searches the longest matching part of pathname in the prefix table and directs the request to the related server. If the host does not find the required path in the path table, then it broadcasts the request to all the servers. The server that contains the required path sends a message to the host along with its own ID. When the host receives the message, it updates the prefix tables dynamically.

11.3.2 Remote File Access

In a distributed system, the files are located on specialized servers. These servers are responsible for holding large disks, such as Redundant Array of Inexpensive Disks (RAID) for storing files. These disks provide an entire file or a part of the file that is requested by a host to perform I/O operation or data processing. This mechanism of accessing files is known as remote file access. It provides transparency to an end user who can access any file remotely, regardless of its location. You can perform remote file access in the following ways:

- **Uploading**: In this method, queries are sent to the location of the files for processing. This method reduces the bottleneck that occurs in the network but leads to an overloading of the server because all the processing is done at the server side.
- **Downloading**: In this method, files are sent to the host or the client side. This mechanism is considered as the simple and efficient method when the client requires the entire file for processing. The problem associated with this method is that the entire file is processed when a portion of it is only required. In such case, this method becomes expensive and uses a lot of disk space and time in downloading a complete file.

- **Remote access**: In this method, the client remotely access the files that are present in the network. The client does not interfere with the application that is running either at the server side or at the client side. If the application is running at the server side, then the client sends the query to the server. The query is processed at the server and the result is forwarded to the client. If the application is running at the client side, then a portion of the requested file is transmitted to the client and the processing of query is done locally. Moreover, an operating system stores files in the cache to avoid repeated searching.

Remote file access uses cache for increasing the performance of network by reducing the network traffic and disk I/O because transferring of complete file is not required.

11.3.3 Cache Mechanism

Cache is a temporary storage area where blocks of files are stored for fast recovery. If the desired block is not present in the cache, then a copy of this block is accessed from the file server where it is stored.

Cache contains data, which is a copy of the master file stored on the server. This means when the end user modifies the copy of data, the master copy also needs to be modified so that consistency of data can be maintained. The cache is always consisted of more data than it requires in order to satisfy the different requests simultaneously. Figure 11.1 shows the working of cache in the distributed system.

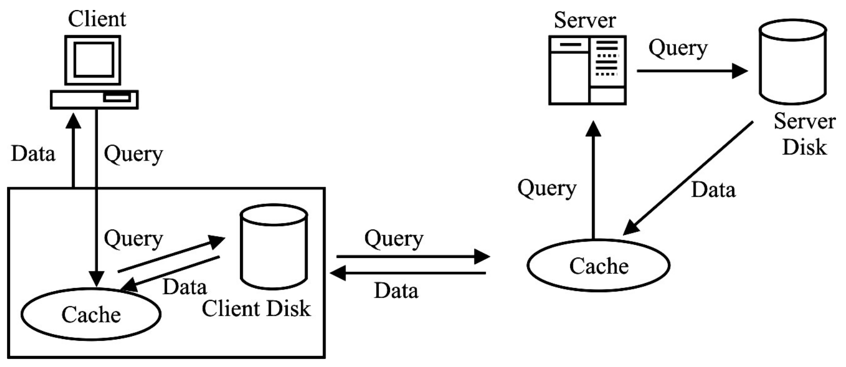


Fig. 11.1 *Working of Cache in Distributed System*

In the cache mechanism, the client machine requests a query and searches its local cache for the required block of data. If the client machine is not able to find the required data, then it searches the required block in the server cache. Again, if the client machine does not get the required block, the query is transferred to the server. The server performs the required search and stores the desired block into its cache from where the block is transferred to the client's cache.

If the data in the cache is modified, then the modified data is sent back to the server for permanent storage and to maintain data consistency.

Location of the Cache

Data is cached at the server side as well as the client side. It is an important issue for distributed operating system to select the appropriate place for cache mechanism. Generally, the cached data is stored at two places, which are as follows:

- Main memory
- Disk

The main memory and the disk have their own advantages and disadvantages. Disk caches are reliable and can survive system crash because the data can be recovered from the disk, as it is stored permanently on the disk. On the other hand, memory caches are faster than the disk cache because speed of searching data from the main memory is faster than the speed of searching data from a disk. Hence, they help in increasing the performance of a system. The four different places where blocks of files are cached are enumerated below:

- **Client main memory**: When the client main memory is used as a cache, the access speed of memory gets increased but network traffic is reduced. It also causes file inconsistency, which occurs when various processes modify the data in the main memory and the system crashes before storing it permanently.
- **Client disk**: When the client disk is used as cache, it reduces network traffic and the performance of a system.
- **Server main memory**: When the server main memory is used as cache, it increases the disk access rate but on the other hand slows down the performance of a system.
- **Server disk**: When the server disk is used as cache, it allows all the clients to access the files and provides huge space for cache mechanism. The major problem with this cache is that it increases the network traffic.

Cache-Update Policy

Blocks of data are transferred from files to server and from server to files constantly through cache. This transfer of data makes data inconsistent, which is critical for a file system. For maintaining the consistency and reliability of files, distributed operating system has to adopt some policies for writing the modified data onto the master copy. The policies adopted by the distributed operating system are as follows:

- **Write-through**: It is the most reliable policy that maintains data consistency even if the system crashes. In this policy, the master copy is updated as soon as the data in cache is modified. This policy has a disadvantage that it provides poor performance because the system has to perform an update function for master copy after every modification of data in the cache.

- **Delayed write**: In this policy, data is updated in cache and the master copy is updated after a short interval of time. The drawback of this policy is its poor reliability because the system crashes result in loss of unwritten data.
- **Write-on-close**: This policy is a new version of delayed write policy. It is suitable when files are required for a long period of time and are accessed frequently. In this policy, the master copy is updated only after the file is closed. The main disadvantage of this policy is that the user might loose the data when the system crashes.
- **Write periodically policy**: In this policy, the master copy is updated after a definite interval of time: for example, the master copy is updated after every 40 seconds. It is also the new version of the delayed write policy. In this policy, the data can be lost only at the last interval of time

Cache Consistency

The block of files, which reside in the cache, become outdated after the request is satisfied. In this, more than one process can access the same block of data multiple times. For this, the latest copy of the file is required to be cached. The file system adopts the following two approaches for verifying the validity of data blocks:

- **Server initiated approach**: In this approach, a server maintains the records of each file that resides in the client cache. It also detects the potential conflicts that arise by caching a file by two or more clients and solves these conflicts by implementing session semantics. Session semantics describe that writes are visible only after the completion of a session or a short time interval and after the client closes the file in which the client is working. Server also notifies the client that is using same file for checking the invalidity of that cache file. Clients have to declare the mode of operation before opening the file. If two or more clients want to write on the same cache, then server disables the cache.
- **Client initiated approach**: In this approach, the clients check the validity of a cache file. The client contacts the server and asks if the copy of the file, which is present in the cache, is consistent or not with the master copy. The client initiates the check after every access or after a definite interval of time or at the time of opening the file.

11.3.4 File Replication

In a distributed file system, file replication is broadly used. File replication improves the availability of files to clients as clients can access the replica of the file stored at the nearest site in the network. Hence, it helps in saving the server service time and also reduces the delay time while communicating.

File replication is also important in case of system failures. Replicas of files are stored on the machines in which failure does not occur. These machines are linked with other replicas. When any of the machines on which the copy of a file is stored, crashes, then the machine, which contains another copy of this file, continues the processing. File replication can be created in the following three ways:

- **Explicit file replication**: A programmer controls the complete process of the replication of files. To create multiple copies of a file, a programmer needs the permission of the directory server. This directory server associates the address of each copy of file along with the file name.
- **Lazy file replication**: In a distributed system, the server controls the replication of files. In this type of replication, a programmer creates the copy of a file on the server. After the copy is created on the server, the server further creates multiple copies of the same file on other servers also. The server creates multiple copies only when the system is not heavily loaded.
- **Group communication based replicas**:The system calls are sent to all the servers that create multiple copies at the same time when the original was made.

The replication of files should be transparent to the clients. Every replica should have the same user-defined names. It is the task of the distributed system-naming scheme to map single high-level names with multiple low-level names.

Files are distributed in a distributed system. The files available on single server are considered as groups. Every file in a file-group is associated with a unique ID. All the replicas of a file have the same ID number. Hence, the files are uniquely identified by a pair < file-group id: file-id >.

File replication also leads to a consistency problem. Any update in one replica is reflected in all other replicas. A distributed operating system uses update protocol algorithms in order to ensure consistency between the replicas. The following are the two most commonly used protocols:

- **Primary copy replication algorithm**: According to this protocol, updated replica is sent to the primary server that holds the master copy. This primary server makes some permanent changes in the master copy and issues commands to the secondary servers. These secondary servers hold the replicas of this file to make necessary changes.
- **Voting**: According to this protocol, clients require permission from multiple servers that hold the replicas before performing any I/O operation, such as read or write on any of the replica. The file replication scheme improves the load-balancing feature of the distributed operating system: for example, if two processes require the same information, then one of the two processes can be sent to some other client machine to continue processing by using the replica of that information.

11.3.5 System Structure

System structure is the basic arrangement of file servers and directory servers in an organization. In some systems, there is no difference between clients and servers.

All the machines run on one single software. Here, offering of file service is similar to exporting the names of directories so that other machines are able to access them. In other systems, such as client-server model and distributed systems, clients and servers are basically different machines in terms of hardware or software. A single organization combines the file service and directory service into a single server and then manages all the directories and file calls itself. Since files and directories are not related to each other, keeping them separate is more flexible. Consider a situation where directory and file servers are separated from each other. In general, a client sends a symbolic name to the directory server, which in turn returns the binary name that can be easily understood by the file server. In Figure 11.2, a directory hierarchy is partitioned among multiple servers.

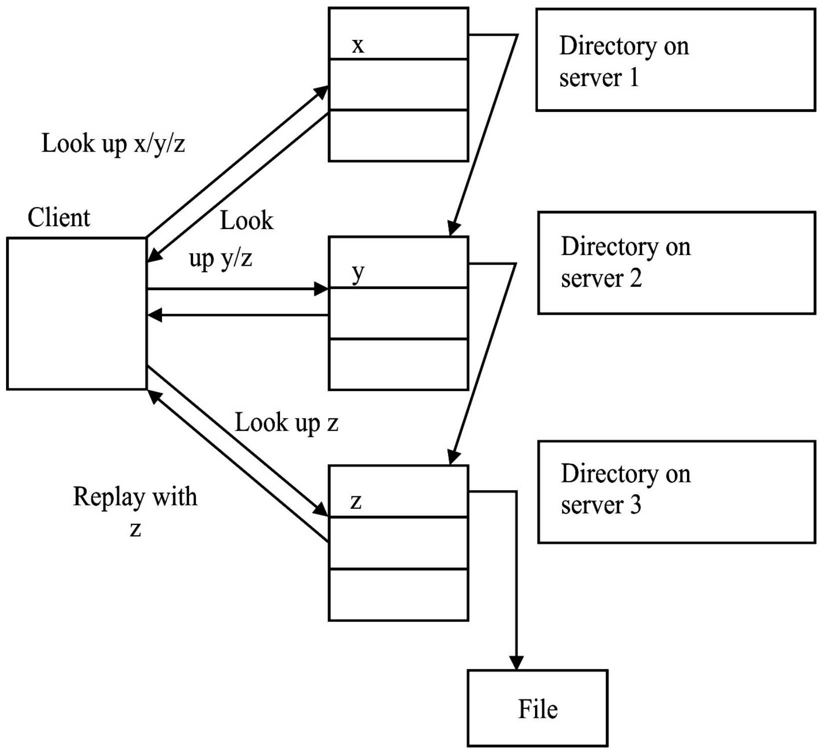


Fig. 11.2 *Directory Hierarchy among Multiple Servers*

In Figure 11.2, there is a system consisting of the current directory on server 1 having x entry, another directory on server 2 having entry y. The third directory on server 3 contains entry z along with its brand name. To look up x/y/z, the client sends message to the server 1 where it finds entry x but then observes that the binary name refers to another server. Now, server1 has two choices that it can tell the client which server contains the entry y and the client will look up y/z by itself. Another choice is that the server can forward the remainder of the request to the server 2 itself. This entire process of looking up path names in multiple directories becomes very complex. Thus, the performance of this process can be improved by maintaining a cache on the hard disk. When the file is opened, the directory-by-directory look-up is skipped and the binary address is taken from the cache to view the path name.

Consider a file server that has commands open, read, write and close files. When a file is opened, the client is provided with a file descriptor that uses call to identify the file. When the server receives the request, it uses file descriptor to find out which file is required. File servers can be classified as namely, stateless servers and stateful servers. Stateless servers are those servers, which do not maintain information related to the state of the client. This means that when a client sends a request to the server, the server receives the request, processes it and sends back the response to the client. The server then deletes all the information related to the client from its internal table. Stateful sever is that server, which maintains information about the state of the client even after the request has been processed and response sent. The advantages of stateless servers are as follows:

- OPEN and CLOSE calls are not required in stateless servers. As a result, there is reduction in the number of messages passed between the server and the client.
- Server space is not wasted on tables.
- Multiple files can be opened by clients while using the tables.
- If a client crashes when a file is opened, there will not be any problem with the stateless server.

The advantages of stateful servers are as follows:

- There are shorter request messages and the network bandwidth is also less.
- It gives better performance as the information about the open files are kept in the main memory when the files are closed.
- In stateful servers, file locking is done by a special lock server.

11.4 DIRECTORY STRUCTURES

Directories are considered as symbolic tables of files that store all the related information about the file it holds along with the content. This information includes file attributes, location type and access privileges. They are also known as containers for files. A directory is itself a file that is owned by the distributed operating system.

Millions of files present in the system need to be managed. Directories provide the means to organize files in a structure. Each entry in a directory contains information about a file. Similar to the files, operations, such as insertion, deletion and searching can be performed on it. The following operations can be performed on different entries in a directory:

- **Searching a file**: Whenever a file is referenced, the directory must search for the related entry.

- **Create a file**: An entry for every newly created file needs to be added in the directory.
- **Delete a file**: Whenever a file is deleted, related entry should be removed from the directory.
- **List directory**: List of files in a directory is shown whenever a user requests for it.
- **Rename a file**: The name of the file should be changeable when the use of file changes or its location changes.
- **Update directory**: Whenever a file attribute changes, its corresponding entry needs to be updated.

Based on these entries and its operations, the structure of directories can be organized in different ways. The three most common structures for organizing a directory are as follows:

- Single-level structure
- Two-level structure
- Hierarchical structure

11.4.1 Single-Level Structure

It is the simplest form of directory structures having only one level of directories. The entire files are contained in the same directory. It appears as the list of files or a sequential file having file names serving as the key. The logical structure of single-level directory is shown in Figure 11.3.

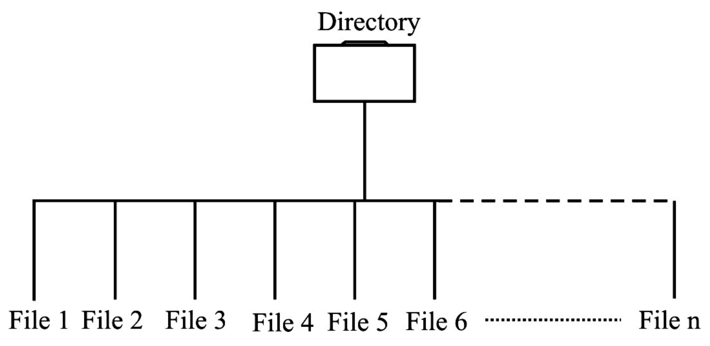


Fig. 11.3 *Single-Level Structure*

This directory structure was implemented in old versions of single-user system. It becomes outdated and inadequate in multiple-user system. Even for single user, it is difficult to keep track of the files if the number of files increases. Moreover, files are of different types, such as graphic files, text files and executable files, and if the user wants to arrange these files in an organized manner, such as group files by type, this structure becomes inconvenient.

The files in a single-level directory should have unique names because they are contained in a single directory. In shared system, unique naming becomes a serious problem. These drawbacks lead us to design another logical structure of directories named as two-level structure.

11.4.2 Two-Level Structure

This structure is divided into two levels of directories, i.e. a master directory and user directories. The user directories are subdirectories of the master directory. A separate directory is provided to each user and all these directories are contained and indexed form in master directory. The user directory represents a list of files of a specific user.

The two-level structure looks like an inverted tree of height 2 cm. The root of this tree is the master directory having user directories as its branches. The files are the leaves of these branches. The logical structure of two-level directories is shown in Figure 11.4.

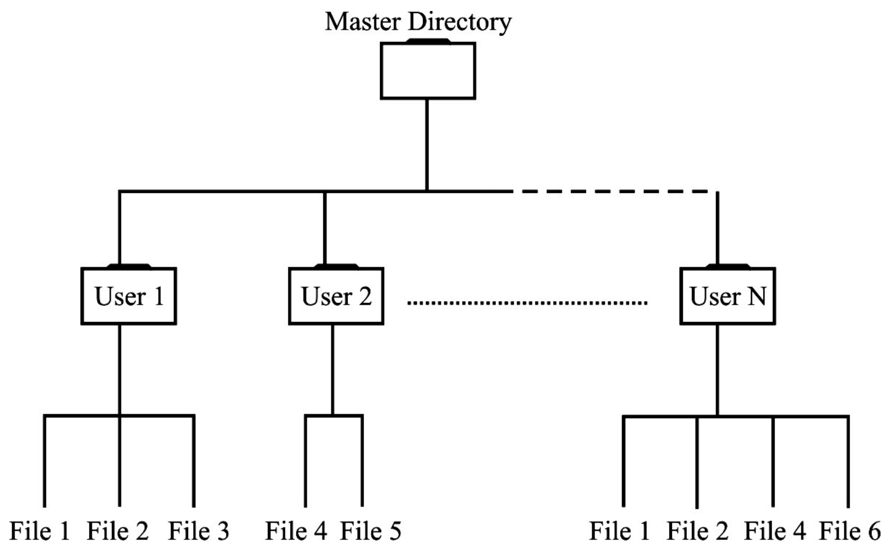


Fig. 11.4 *Two-Level Structure*

The user name and the file name are the pathname for a file. This structure solves the problem of unique names upto a certain extent, i.e. the user can assign duplicate names to the files, provided that the files are present in different directories. The names need to be unique in the user's own directory because two file names cannot be same in a single directory. A user searches a file in his own directory and only then allows different users to have files with same names.

Adistributed operating system uses user's directory to perform the various operations related to the files, such as create and delete a file in two-level structure, because the user directory is used to initiate the commands. The distributed operating system uses system calls to create or delete a user directory. This system program creates or deletes a user directory entry from the master directory.

This two-level structure provides no help in grouping the files of different types. In a shared system, one user wants to access another user's files because the files are shared in a shared system network, which again creates problem of uniqueness in the file names. The user has to give a complete pathname to name a file in other user's directory.

11.4.3 Hierarchical Structure

This is the most powerful and flexible structure and is implemented in almost every operating system. The two-level structure is extended into more advanced hierarchical structures of arbitrary levels. It uses the same concept of two-level structure of master directory having user-directories as subdirectories. In addition, each user-directory in turn has subdirectories and files as its branches and leaves. A typical hierarchical structure of directories and files is shown in Figure 11.5.

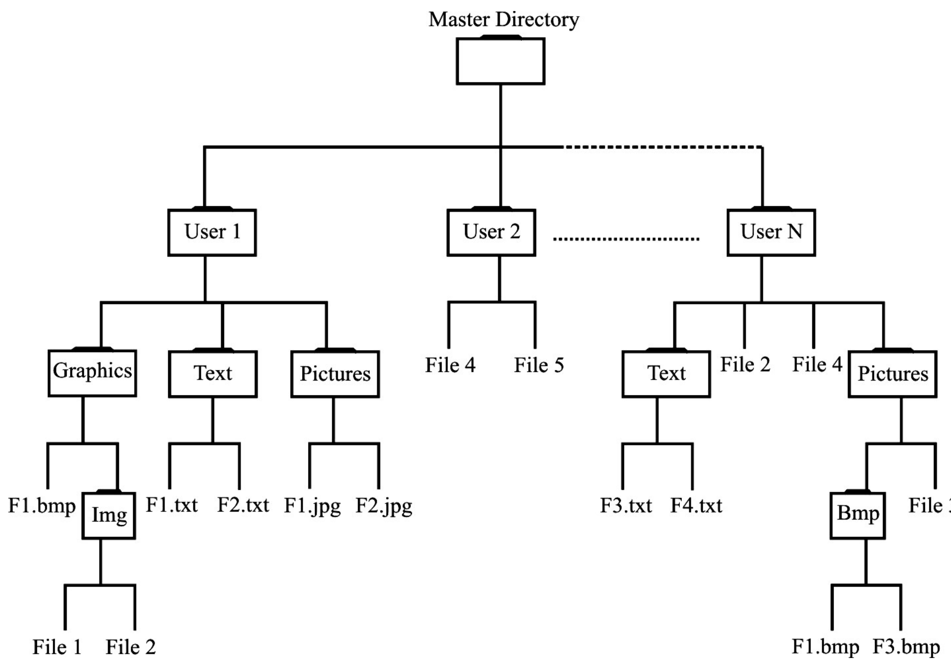


Fig. 11.5 *Hierarchical Structure*

Users can create their own subdirectories to organize the files of different types, such as a separate subdirectory for the graphic files or a separate subdirectory for the text files. System calls are used to create or delete directories. Internal format is the internal structure in which details of directories are stored. Each directory has an entry that stores a special bit representing a subdirectory or a file. The 0 bit represents a file and 1 bit represents a directory.

A user always works on the file in the current directory. The current directory holds all the files, which a user currently requires. The distributed operating system searches the current directory for reference to a file. In the hierarchical structure, a user can access a file, which is not in the current directory by giving the pathname. The user can change the current directory also through a system call.

In the hierarchical structure, a file can be referenced in two ways: absolute pathname and relative pathname. The absolute pathname starts from the root and ends at the required file following a path of directories and subdirectories. The relative pathname starts from the current directory to the file.

A user can access another user's file by giving its pathname. In hierarchical structure, pathname to a file can be longer than the two-level directory. This increases the search time for a file that resides in other user directory. Process scheduling is a technique that ensures efficient multitasking of processes. The process manager assigns the CPU time and other resources to carry out the process of multitasking.

The process scheduling maximizes the utilization of CPU by simultaneously executing multiple processes. The main purpose of process scheduling is to switch CPU among different processes frequently, so that the CPU can process all the programs simultaneously. In order to meet these objectives, the process scheduler selects a process from the available processes that are ready for execution in CPU. The process scheduler examines the information in the process control block that helps in selecting a process for execution. In case of a single-processor system, there is only one process that is to be executed. If there is more than one process in the queue, then the processes have to wait for the CPU to get free for processing.

Check Your Progress

1. What are the two types of naming services in a distributed system?
2. List the three ways to create file replication.
3. Name the two locations where cached data can be stored.
4. Explain the operations that can be performed on different entries in a directory.

11.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The two types of naming services in a distributed system are as follows:
   A. Multi-level mapping
   B. Multi-valued mapping
2. The three ways to create file replication are stated below:
   A. Explicit File Replication
   B. Lazy File Replication
   C. Group Communication Based Replicas
3. The two places where the cached data can be stored are:
   A. Main memory
   B. Disk

4. The operations that must be performed on different entries in a directory for managing them, are as follows:
    - A. **Searching a file**: Whenever a file is referenced, the directory must search for the related entry.
    - B. **Create a file**: An entry for every newly created file needs to be added in the directory.
    - C. **Delete a file**: Whenever a file is deleted, the related entry should be removed from the directory.
    - D. **List directory**: The list of files in a directory is shown whenever a user requests for it.
    - E. **Rename a file**: The name of the file should be changeable when the use of the file changes or its location changes.
    - F. **Update directory**: Whenever a file attribute changes, its corresponding entry needs to be updated.

11.6 SUMMARY

- A file attribute is required by a file system to manage a file. It is the information about a file that is linked with every file. Some of the attributes, such as file location, name and date and time of creation of the file, are unique for each and every file in the disk.
- User-defined attributes contain those file information, which are specified by a user.
- System-defined attributes contain the file information, which are not specified by a user. An operating system automatically stores some of the information related to the creation and storage location of the files.
- When two or more users share a file at the same time in a network, it is called file sharing.
- Every object, which includes files, has a name associated with it. These names are referred by the end users and are mapped with low-level machine understandable identifiers that are used for identifying a file.
- Naming service refers to the process of mapping user-defined file names with transparency. It is used when the users do not know the location of a file.
- In a distributed system, the files are located on specialized servers. These servers are responsible for holding large disks, such as Redundant Array of Inexpensive Disks (RAID) for storing files. These disks provide an entire file or a part of the file that is requested by a host to perform I/O operation or data processing. This mechanism of accessing files is known as remote file access.

- Cache is a temporary storage area where blocks of files are stored for fast recovery.
- In a distributed file system, file replication is broadly used. File replication improves the availability of files to clients as clients can access the replica of the file stored at the nearest site in the network.
- System structure is the basic arrangement of file servers and directory servers in an organization.
- Directories are considered as symbolic tables of files that store all the related information about the file it holds along with the content.

11.7 KEY WORDS

- **File Server**: It is a machine where the files are stored. It also implements the file services.
- **File System**: It is a part of a distributed operating system that is responsible for controlling the secondary storage space.
- **File Type**: It refers to the type of information stored, i.e., binary file, text file, picture file or program file.
- **File Sharing**: When two or more users share the same file at the same time, it is called file sharing.
- **Naming Service**: It refers to the process of mapping user-defined file names with transparency.
- **Multi-Level Mapping**: The mapping of user-defined names and low-level identifiers with the location of the file in the network is known as multi-level mapping.
- **Static Map**: It is a very simple mechanism, which is used where the location of file or directory does not change.

11.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. Discuss the semantics of file sharing.
2. Discuss the various attributes of a file.
3. Discuss the use of file replication.
4. Does a file system replicate all the files at a single time during file replication? Give an example of a kind of file that is not worth replicating.

Long Answer Questions

1. Explain the concept of remote file access.
2. Explain the concept of naming.
3. Why directory structure is needed for files? Explain the different types of directory structure.

11.9 FURTHER READINGS

Tanenbaum, Andrew S. and Maarten Van Steen. 2006. *Distributed Systems: Principles and Paradigms*. New Jersey: Prentice Hall.

Garg, Vijay K. 2002. *Elements of Distributed Computing*. New Jersey: Wiley-IEEE Press.

Sinha, Pradeep K. 1996. *Distributed Operating Systems: Concepts and Design*. New Delhi: Prentice-Hall of India.

UNIT 12 FAULT TOLERANCE AND TRANSACTION IN DFS

12.0 INTRODUCTION

In this unit, you will learn about the fault tolerance and atomic transactions. A distributed system should exhibit fault tolerance and withstand component failures to be able to provide quality services to its clients. You will learn that to achieve fault tolerance, systems use two phase-based commit protocols.

12.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the concept of fault tolerance with respect to distributed systems
- Discuss the properties of transactions
- Explain the two-phase commit protocol
- Understand how to achieve concurrency

12.2 FAULT TOLERANCE

In a distributed system, it often happens that one or more components of the system undergo failure leaving other unaffected components functioning normally. This quality of the system to continue functioning even when some of its components have failed is termed as fault tolerance.

The ability to function, albeit with reduced capability, in the situation of a failure in itself or in one or more of its associated components is the basic characteristic of a fault-tolerant system.

Such failures might arise due to power failures, disk failures, communication failures, memory failures and general component failures. In the event of any of these happenings, a fault tolerant system will resist a complete failure and would continue to operate in a reduced functionality mode. This attribute of a fault-tolerant system is known as graceful degradation. The decrease in the operational quality is proportional to the failure.

Fault-tolerant systems are deployed in mission critical systems, life-support systems, aviation, hazard-oriented systems, financial systems, banking, stock markets, etc. Common examples include air traffic, medical apparatuses used in intensive care units, bank ATMS, electronic transaction equipment at merchant establishments, power backup systems in data centre.

Fault tolerance is a holistic attribute of system; a system being composed of smaller constituent participating entities. It is a cooperative characteristic of the system to recover and continue functioning in the event of a failure.

Fault Tolerance in Network Systems

As an example, connection-oriented network protocols exhibit fault tolerance. In the event of a link failure at a network node, the receiver/sender employs an acknowledgement policy. If the sender does not receive an acknowledgement from the receiver within a specific time period the packet is retransmitted.

The retransmitted packet is then routed through a network in which all the intermediate links are functional. The TCP network layer is an example of such a fault-tolerant system.

An interesting quality of a fault-tolerant system is that it allows for correction of the faulty component without having to shutdown the entire system. For example, in the event of a network failure due to link breakdown, the router (a component in the distributed network system) will send packets over an alternate link till the fault is recovered.

Fault tolerance is often achieved by employing redundancy in systems. In certain cases, a fail-safe operation is duplicated across multiple systems and if a failure happens on one of the systems, a fall-back system takes over the operation and continues from the point where the original system left off.

Systems that are fault tolerant in nature have a well-defined recovery path. Two widely used recovery techniques are roll-back and roll-forward. In the case of the former, in the event of a failure, the system reverts back it state to the last non-erroneous point of operation and resumes processing; in the case of the latter, the system on detecting an error, corrects itself and continues with the processing.

Attributes of Fault-Tolerant Systems

For a distributed system to be able to resist complete failures, or in other words to be fault tolerant, it should be dependable. A dependable system should satisfy the requirements of **availability**, **reliability**, **safety** and **maintainability**.

Availability is the metrics of a system which tells us the probability that the system is functioning properly at any given instance in time. If a system fails and consequently recovers within a very small amount of time—the system to a user will be unavailable only for that very short duration. Thus, the probability of the system being available would be high.

On the other hand, **reliability** is the metrics of a system which indicates the ability of a system to operate without failures over a period of time. In the context of reliability, it is necessary for a system to function without failures. A highly available system does not necessary imply a highly reliable system. A system which stops functioning every now and then, but for a very small duration of time, is a highly available system—as the probability of finding it up and running is high; but since the system does not function without interruptions over an extended period of time, it cannot be said to be reliable.

A system is said to exhibit **safety** if a failure in it does not lead to disaster. A typical example would be the flight control systems found in passenger airplanes. They are designed taking into account that a failure in one of its component does not affect the functioning of the system.

In the context of dependable systems, **maintainability** implies that in the event of a failure, the system can be repaired within a reasonable timeframe and without much effort. A maintainable system can in some cases be designed to automatically detect and correct errors. These self-recovery mechanisms are often present in high availability systems.

Errors, Failures and Faults

A distributed system provides a set of services to its users. When one or more of these services become unavailable and does not perform as expected, the system is said to have failed. A failure in such systems is typically due to the erroneous state the system reaches as a result of a fault in one or more of its components. In order to build fault-tolerant distributed systems, it is very important to study how failures occur.

Failures are the effect of error in the system; the error manifests itself in the form of component failure in the distributed system, and occurs due a fault. As can be noticed, this cause-effect sequence, i.e., the relationship between fault, error and failure is present in all cases of system malfunction. Analysis of each of these components, help us design effective fault-tolerant systems.

Fault Categories

Faults can be categorized into the following types:

Permanent: A permanent fault is one that remains until the defective part is changed. As an example, a failure in a database server due to a hard disk fault remains uncorrected until the bad disk is replaced with a good one followed by restoration of data.

Transient: A transient fault happens once and then does not reappear; for example, a cellular phone fails to detect the mobile network in places such as tunnels and underground railway systems, but once the person comes out in the open, the system re-establishes connection with the nearest base station. Another example of a transient fault would be the disruption in the wireless transmission due to solar flares. Solar flares are explosions in the sun's outer atmosphere which results in the huge release of electromagnetic radiation into space. These radiations sometimes disrupt telecommunication which automatically disappears once the effect of the solar flare subsides.

Sporadic: As the term suggests, these types of faults occur on and off. For example, due to a malfunction in the cooling system, a processor shuts down to prevent its circuit burn-out, thereby bringing the system to a halt; once the temperature drops, the processor starts functioning again. Sporadic failures are the most difficult to detect and isolate. This makes them the most difficult to correct and accounts for most of the system failures.

Failure Models

In accordance with a specific failure classification scheme, failures can be categorized by their types into crash failures, omission failures, timing failures, response failures and arbitrary failures.

Crash Failure: This occurs when a server stops due to a system malfunction. Before the crash happened, the server was functioning correctly. A server program termination by the operating system due to an illegal memory address access would be a typical example of a crash. To recover from such situations, the server program needs to be restarted.

Omission Failure: Such failure takes places when a server does not send its response to a request. It might very well be the case that the server never received the request to start with. It might also be due to a transient failure in the communication media, which resulted in the total loss of network traffic. An omission failure can also occur in a situation when the server has failed to transmit the response after processing the request.

Timing Failures: These are noticed when a system is not able to respond to the requestor within a predefined time period. As an example, if a http request takes a long time to complete by a Web server, a timing failure happens—the result of which is the familiar '408 Request Timeout' response displayed by the browser. A timing failure might also happen if the server responds later than it is expected to. Timing failures might arise due to high server loads when the request was issued, low communication bandwidth, etc. Timing failures also arise due to response sent by the server too quickly, when the client was not expecting it.

Response Failures: These are more severe forms of failures. When a system responds incorrectly to a request, a response failure is said to have taken place. Response failures are of two distinct types, value failure—in which the response to the request is incorrect and state transition failure. A state transition failure occurs when the system receives a request that it is not designed to handle. The system might process the request incorrectly and generate an incorrect response.

Arbitrary Failures: These are the most serious forms of failures. These failures are also known as **Byzantine failures**. Byzantine refers to the Byzantine General's problem, in which a number of generals separated by distances need to decide upon whether to attack the enemy or retreat. Each general communicates the decision to the nearest general. To aggravate the problem, it might be that some of these generals are traitors and might maliciously alter the message. An arbitrary failure happens when the system responds incorrectly because of the presence of faulty components within it. For example, if a server interacts with several other servers for processing a request and if some of these servers are faulty, then the system fails to produce the correct output.

The manner in which a server system exhibits a crash failure can differ from system to system. Some systems might halt altogether due to a fault. Exigency handling in such systems might broadcast that it is about to stop. Such failures are known as fail-stop. Fail-silent systems are those which do no intimate the interested parties that it is about to halt. In these systems, observer's processes monitor the servers and initiate appropriate action, such as to restart the failed system, inform interested parties, and so on. The other class of systems is referred to as fail-safe. These systems, in the event of a failure will respond to requests in such a manner that other processes would understand that all is not well with the server.

Fault Tolerant Design Techniques

Redundancy

Redundancy is often used to build fault-tolerant systems. Take the case of a passenger aeroplane which typically has more than one engine. In the event of an engine failure, the other takes over. In other words, a redundant engine is installed to be used in the event of primary engine failure. Similarly, enterprise databases are replicated and hosted on different systems. In the event of a disk crash on any of the system, the other systems still continue to function normally. In yet another example, multiple copies of a critical process are run in a server. If any of the processes fail, the others still continue to function, thus providing uninterrupted service. This type of redundancy, where more than one instances of a critical system is deployed in known as **physical redundancy**. Patterns exist that are employed to address fault tolerance using physical redundancy. Triple Modular Redundancy (TMR) is a technique where two instances are deployed for each of the components in a system. Each of these systems at a particular stage receives input at all the three components of the previous stage, and so on. A component in a particular stage accepts an input if it was provided by two or more of the components of the previous stage. If three different inputs are received, the system state is undefined.

Temporal Redundancy, on the other hand, involves recording a series of events that happen in a system, and playing them back in case of a failure. This technique is heavily used in database transactions. A transaction can be started, a series of operations can be made, and then the transaction can be committed. If a system failure occurs, the transactions that have been logged can be replayed to bring the system to the state where it was before the crash happened. Temporal redundancy is used in scenarios where transient faults or sporadic faults occur.

Another technique used in the design of fault-tolerant systems is **information redundancy**. In this case, extra data is added to the transmission which helps in detecting and correcting errors at the receiver's end. An example of such an algorithm is the Hamming code. Extra parity bits are added to the transmitted data in such a way, that the receiver on inspecting the data will be able to ascertain whether the data is in error, and in certain cases, could also correct the data.

12.3 ATOMIC TRANSACTIONS AND DESIGN PRINCIPLES

Atomic transactions provide synchronization at a higher level of abstraction. The user does not have to bother about how the algorithm and the processes work together. The first atomic transaction model was inspired by the business world, where if two parties are bound by a contract, they need to complete a signed transaction. However, if they are not legally bound, any party can revert any time. Similarly, in computer, a process announces its wish to initiate a transaction with other processes. They can create and delete objects and perform other operations, such as read on the processes. Now, the initiator announces that it wants all the other processes to commit the changes they have done so far. If all the processes agree to the changes, the results of change are made permanent. However, even if one process disagrees, the changes are rejected and the transaction is reverted to the original state.

Transaction Model

A transaction model consists of some processes capable of failing at random. The communication is unreliable in the sense that the messages can be lost. However, lower levels can use a timeout and retransmission protocol to recover lost messages. Thus, it is assumed that the software transparently handles the communication errors. Since transactions make use of databases to store relevant information, some kind of storage is required. Some primitives are also required to establish proper communication between the communicating processes.

Stable storage

In the transaction model, a stable storage is required that can survive any crashes except for calamities, such as floods and earthquakes. A pair of ordinary disks can be used to implement a stable storage. There are two drives, drive 1 and drive 2. Each block on drive 1 is copied to the corresponding block on drive 2. A block to be updated is first updated on drive 1 and verified and then copied to drive 2. Now, suppose a system crashes after updating drive 1 but not drive 2. When the system recovers, the disk can be compared block by block and when the corresponding blocks in the two disks differ, the block of drive 1 is copied to drive 2.

Another problem can be the spontaneous decay of a block from exposure to dust particles or may be general wear and tear with time. Thus, in such cases, a previously valid block may suddenly result in a checksum error. The bad block can be regenerated from the corresponding block on the other drive. Due to its high tolerance to failures, stable storage is well suited for applications, such as atomic transactions.

Transaction primitives

Special primitives are required for programming transactions. These primitives must either be supplied by the operating system or by the language run-time system. Some of the examples of primitives are as follows:

- **BEGIN_TRANSACTION**: It marks the start of a transaction.
- **END_TRANSACTION**: It terminates a transaction and tries to commit it.
- **ABORT_TRANSACTION**: It is used to kill a transaction and restore the previous values.
- **READ**: It is used to read data from a file or object.
- **WRITE**: It is used to write data from a file or object.

The BEGIN_TRANSACTION and END_TRANSACTION primitives delimit the scope of a transaction, whereas the primitives and operations inside them form the body of a transaction. Either all of them execute or none of them is executed.

Properties of transactions

Transactions are characterized by four essential properties that are collectively known as ACID properties. These properties are enumerated below:

- **Atomic**: It ensures that each transaction either occurs completely or not at all. Also, if at all it happens, it should occur in a single indivisible action. This means that other processes must not be able to view the intermediate states of the transaction.
- **Consistent**: It says that a system should be consistent before and after a transaction. This means that if some invariants apply to a system before transaction, they must hold true even after the transaction is complete: for example, in a banking system, the law of conservation of money must always hold true, i.e. money should not be lost during a transaction.
- **Isolated**: This means that transactions should seem to be occuring serially independent of each other. If two transactions are running simultaneously to each other and to the other proceses, the final result should seem as if the transactions occurred sequentially.
- **Durable**: It specifies that if a transaction has been committed, no matter what, the transaction proceeds forward and its results become permanent. Thus, the results cannot be undone, once a transaction has been committed.

Nested transactions

When a transaction consists of sub-transactions, it is called a nested transaction. The top-level process or the parent process may give rise to child processes that may run parallel to each other on different processors to gain performance. Each of the child processes may also fork further into sub-transactions. However, sub-transactions result in a problem. Consider a situation where a parent transaction has several sub-transactions running in parallel. Now, suppose one of the sub-transactions commits itself and makes its results visible to the parent process. After a while, the parent aborts itself, restoring the entire system to its initial state. Now, the results of the committed sub-transaction must be reverted. Thus, even after a sub-transaction commits itself, the changes may not be permanent.

Since nesting can be done up to any level, it requires a considerable amount of administration. Therefore, as a general semantics whenever a transaction or sub-transaction starts, it is given a separate copy of all the objects in the entire system to manipulate according to its needs. Therefore, if the process aborts, its private copy vanishes as if it never existed and if it commits, its private copy replaces the original copy.

Implementation of Atomic Transactions

A clean way is required to implement atomic transactions so as to maintain their ACID properties. The methods commonly used for implementation are private workspace and writeahead log.

Private workspace

Under this method, a separate (private) workspace is given to the process when it starts a transaction and all the changes and operations it performs are done on its private copy until it commits the changes. However, the cost of copying all the objects to a private workspace is quite high. It can be restricted with certain optimizations.

The first optimization is based on the fact that when a process reads a file but does not modify it, there is no need for a separate private copy. As a result, when a process begins a transaction, initially the private workspace created consists only of a pointer that points back to the parent workspace. Therefore, when a transaction is at the top-most level, its private workspace is the real file system. If a process opens a file for reading, the pointers are followed backward till the file is located in the parent's workspace. When the file is opened for writing, it is located in the same manner as for reading; however, this time it is copied to a private workspace first.

There is another optimization that removes most of the copying processes. Rather than copying the entire file, only its index is copied to the private workspace. The file index specifies where the file's disk blocks reside. Figure 12.1 depicts the implementation of private workspace through the file index.

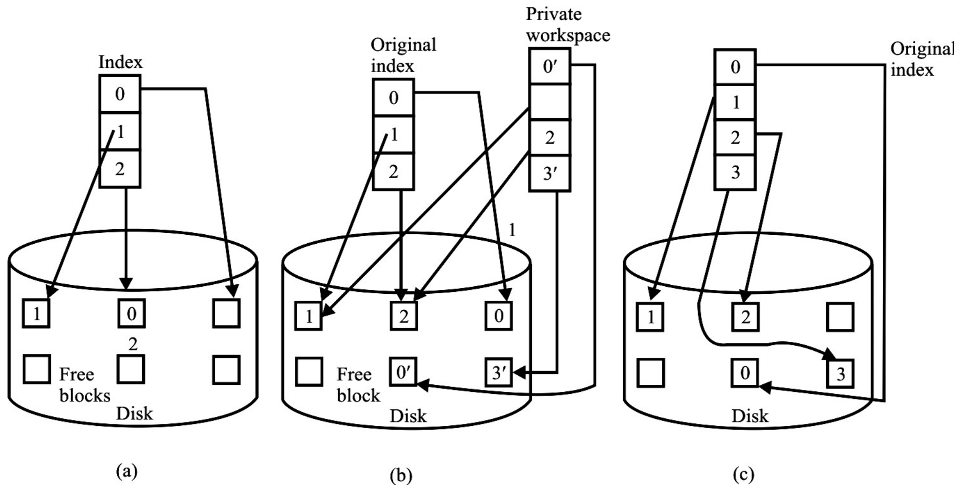
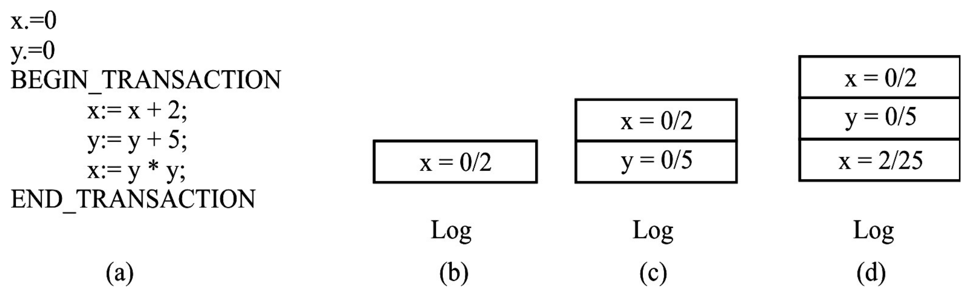


Fig. 12.1 Implementation of Private Workspace

Figure 12.1(a) shows the file indices and disk blocks for a three-block file. Using the index, it is possible to read a file from the original location itself as the index contains the disk addresses for the original copy. However, when a file block is modified, the block is copied and its address is inserted into the index. Figure 12.1(b) shows that the process running the transaction sees the modified version of files, i.e. 0' and 3', whereas the other processes continue to see the original file. When a transaction aborts, the private workspace is simply deleted. However, if a transaction is committed, the private indices are moved to the parent workspace as shown in Figure 12.1(c).

Writeahead log

Writeahead log or intentions list is another method of implementing atomic transactions. Under this method, the files are modified in their original location. However, before a block is modified, a record is written to the writeahead log on a stable storage. The writeahead log specifies which transaction is making the change, to which file and block is the change being made and what are the old and new values. The file is changed only when the log has been successfully written. Figure 12.2 shows the working of the writeahead log.



```
x.=0
y.=0
BEGIN_TRANSACTION
        x:= x + 2;
        y:= y + 5;
        x:= y * y;
END_TRANSACTION
```

Fig. 12.2 *Working of Writeahead Log*

Figure 12.2(a) shows a simple transaction using two shared variables, x and y, initialized to 0. A log record will be written for each of the three statements inside the transaction before they are executed. The old and new values are separated by a slash. Figures 12.2(b), (c) and (d) depict the state of the log after each of the three statements are written to it.

When a transaction is successful and commits itself, a commit record is written to the log, without any need to change the data structures, as they are already updated. However, if the transaction aborts, the log is used to restore the system to its previous state. The log is read from the end towards beginning and the described change is undone. This process is called rollback.

You can also use the log to recover from crashes. Consider a situation where the process writing the log fails at the time of writing the last log, as shown in Figure 12.2(d). As a result of failure, it is not able to write the last x value to the log. Therefore, when the machine reboots after the crash, the log is checked to see if any transactions were in process at the time of crash. When the value of x is found to be 2, the system knows that the crash occurred before the update and x is changed to the current value of 25.

Two-Phase Commit Protocol

A transaction must be committed atomically as a single indivisible unit. It may require multiple processes on different machines to communicate with each other in a distributed system. This is because each of these processes holds variables, files, databases and other objects that are used and modified by the transaction. The two-phase commit protocol is used to achieve atomicity of commit process in a transaction. Figure 12.3 shows the two-phase commit protocol.

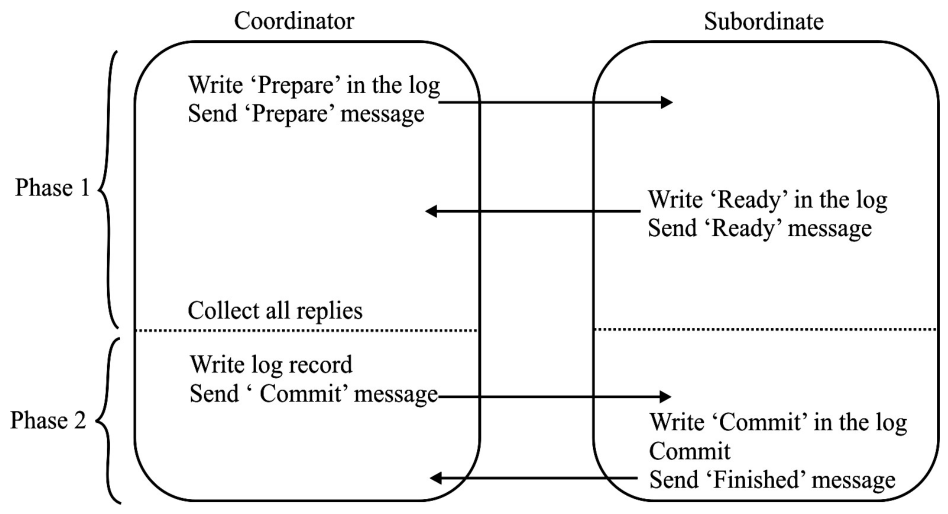


Fig. 12.3 *Two-Phase Commit Protocol*

The process executing the transaction acts as the coordinator. The protocol begins when the coordinator writes a log entry stating that it is beginning the commit protocol. The coordinator then sends a message to all the subordinate processes informing them to prepare for commit. Upon receiving the message, a subordinate checks if it is ready to commit. If yes, it makes a log entry and sends the decision back to the coordinator. When the coordinator receives responses from all the subordinates, it knows whether to commit or abort. If all the subordinate processes are ready to commit, the transaction commits itself; otherwise, it is aborted. In both the cases, a log entry is written by the coordinator and a message is sent to each of the processes informing them about its decision.

The protocol is highly flexible in the face of crashes because the log is stored on a stable storage. In case the coordinator crashes after writing the initial log record, after recovering, it can continue from where it left. If it crashes after writing the result to the log, after recovering, it can inform all the subordinates about the result. Also, if a subordinate crashes before replying to the first message, the coordinator keeps sending messages until it gives up and if the subordinate crashes later, it can view the log and continue from where it left.

Concurrency Control

When multiple transactions occur simultaneously in different processors or processes, a method is required to keep them from interfering with each other. This method is called concurrency control algorithm. The most commonly used concurrency control algorithm is locking.

Whenever a process needs to write to a file involved in a transaction, the process must first lock the file. You can lock a file by using either a centralized lock manager or with a local lock manager on each machine. In both the cases, a list of locked files is maintained by the lock manager and all attempts to lock an already locked file is rejected. Since setting up a lock keeps a process from interfering with another process, it ensures that the file will not change during the lifetime of the transaction.

The simple implementation of locks is overly restrictive and it can be greatly improved by distinguishing between read and write locks. When a read lock is set on a file, other read locks are permitted on that file, as they do not modify the contents of a file. However, if a write lock has been set on a file, no other lock can be permitted on that file. Therefore, it can be said that read locks are shared but write locks are exclusive.

Till now it is assumed that an entire file can be locked. However, a lock may be set on a smaller unit, such as a single record or a page or may be on a larger unit than a file, such as an entire database. The size of an item to be locked is called granularity of locking. A lock can be more precise if the granularity is finer and more parallelism can be achieved with finer granularity.

Two-phase locking is used by most transactions to acquire or release locks. Under this system, a process first acquires all the locks needed by it during its growing phase. It then releases the locks acquired during the shrinking phase. Figure 12.4 shows the two-phase locking system.

NOTES

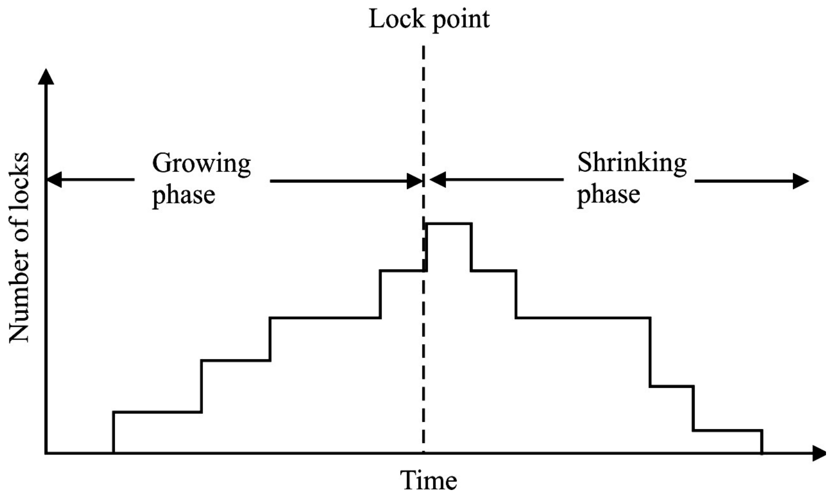


Fig. 12.4 *Two-Phase Locking*

However, in many systems there is no shrinking phase until the transaction has completed and has either committed or aborted. This is called strict two-phase locking and it has the following two advantages:

- Since a value written by a committed transaction is always read by other transaction, there is no need to abort a transaction because its calculations were based on a file it should not have seen.
- The acquisition and release of locks can be handled in a way transparent to the transaction. This is because locks are acquired when a file is to be accessed and locks are released when a transaction has been finished.

Two-phase locking can lead to a deadlock if two processes try to acquire a same pair of locks in an opposite order. This problem is resolved by acquiring all the locks in some canonical order so as to prevent hold-and-wait cycles. Deadlock detection is also possible if an explicit graph of processes and locks is maintained. The graph should specify which process has which locks and which locks are required by it. Then, this graph can be checked for cycles. Now, when it is known in advance that a lock cannot be held longer than t seconds, a timeout scheme can be used in which if a lock remains under the same ownership for a long time, then there must be a deadlock.

Check Your Progress

1. Define the term fault tolerance.
2. What are the requirements of to be satisfied by a dependable system?
3. What are the four essential properties of transactions?

12.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Fault tolerance is a holistic attribute of system; a system being composed of smaller constituent participating entities. It is a cooperative characteristic of the system to recover and continue functioning in the event of a failure.
2. A dependable system should satisfy the requirements of **availability**, **reliability**, **safety** and **maintainability**.
3. The four essential properties of transactions are:
   (i) Atomic
   (ii) Consistent
   (iii) Isolated
   (iv) Durable

12.5 SUMMARY

- Fault-tolerant systems are deployed in mission critical systems, life-support systems, aviation, hazard-oriented systems, financial systems, banking, stock markets, etc. Common examples include air traffic, medical apparatuses used in intensive care units, bank ATMS, electronic transaction equipment at merchant establishments, power backup systems in data centre.
- Fault tolerance is a holistic attribute of system; a system being composed of smaller constituent participating entities. It is a cooperative characteristic of the system to recover and continue functioning in the event of a failure.
- The retransmitted packet is then routed through a network in which all the intermediate links are functional. The TCP network layer is an example of such a fault-tolerant system.
- An interesting quality of a fault-tolerant system is that it allows for correction of the faulty component without having to shut down the entire system.
- Fault tolerance is often achieved by employing redundancy in systems. In certain cases, a fail-safe operation is duplicated across multiple systems and if a failure happens on one of the systems, a fall-back system takes over the operation and continues from the point where the original system left off.
- A dependable system should satisfy the requirements of **availability**, **reliability**, **safety** and **maintainability**.

- Atomic transactions provide synchronization at a higher level of abstraction. The user does not have to bother about how the algorithm and the processes work together. The first atomic transaction model was inspired by the business world, where if two parties are bound by a contract, they need to complete a signed transaction.
- Transactions are characterized by four essential properties that are collectively known as ACID properties.
- A transaction must be committed atomically as a single indivisible unit. It may require multiple processes on different machines to communicate with each other in a distributed system. This is because each of these processes holds variables, files, databases and other objects that are used and modified by the transaction. The two-phase commit protocol is used to achieve atomicity of commit process in a transaction.
- When multiple transactions occur simultaneously in different processors or processes, a method is required to keep them from interfering with each other. This method is called concurrency control algorithm.

12.6 KEY WORDS

- **Fault tolerance:** The quality of a system to continue functioning even when some of its components have failed.
- **Atomicity**: It is a property of atomic transaction that ensures that each transaction either occurs completely or not at all.
- **Consistency**: This means that if some invariants apply to a system before a transaction, they must hold true even after the transaction is complete.
- **Isolation**: This means that transactions should seem to be serialized. If two transactions are running simultaneously to each of them and to the other processes, the final result should seem as if the transactions occurred sequentially.
- **Durability**: It specifies that if a transaction has been committed, no matter what, the transaction proceeds forward and its results become permanent.

12.7 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. Write a short note on fault tolerance.
2. What are the attributes of fault tolerant systems?

3. Discuss the fault tolerant design techniques.
4. What are the ACID properties of transaction?

Long Answer Questions

1. Explain the various categories of fault.
2. What are the various failure models? Explain.
3. Explain the implementation of atomic transactions.
4. Explain the two phase commit protocol.

12.8 FURTHER READINGS

Tanenbaum, Andrew S. and Maarten Van Steen. 2006. *Distributed Systems: Principles and Paradigms*. New Jersey: Prentice Hall.

Garg, Vijay K. 2002. *Elements of Distributed Computing*. New Jersey: Wiley-IEEE Press.

Sinha, Pradeep K. 1996. *Distributed Operating Systems: Concepts and Design*. New Delhi: Prentice-Hall of India.

BLOCK - V

SECURITY

NOTES

UNIT 13 INTRODUCTION TO SECURITY

13.0 INTRODUCTION

In this unit, you will learn about the various concepts related to security. Computer security is a branch of information security as applied to computers. Computer security involves the security of computer programs and data. The objectives of computer security include:

- Protection of data and programs from theft and corruption,
- The prevention of program subversion, and
- The preservation of availability

Security is an important issue for all computer users, particularly for developers. The main reason behind the insecurity of computer programs is the flexibility in the programming of hardware instructions to build programs. That is, there are many ways to build a computer program to meet a specific requirement. This is actually a desirable feature that makes computer programming easier. However, this flexibility in programming makes it very difficult to ensure the correct behaviour of computer programs.

13.1 OBJECTIVES

After going through this unit, you will be able to:

- Identify security threats and attacks from inside and outside a system
- Discuss attacks employing Trojan horses, Spoofing, Logic bombs and Trap doors

- Discuss attacks from Viruses and Worms
- Discuss the fundamentals of cryptography in computing
- Discuss symmetric-key encryption, public-key encryption and digital signature
- Describe various techniques of user authentication

13.2 SECURITY ATTACKS

Attacks on a system may be from insiders who have logged into the system or outsiders who prompt an innocent legitimate user to download free of cost a malicious program such as Trojan horses or Viruses or Worms masquerading as an exciting game or music. These intruders may be thrill-seeking adolescent hackers or professional hackers stealing vital information for sale or disgruntled employees who want to damage the system for revenge or profit. A secure system administrator must monitor physically and by employing appropriate auditing of the usage of resources the following activities:

- Unauthorized attempt to use computers by any insiders
- Browsing through system libraries and files, and modification and destruction of information by users
- Programming denial of service attack such as creating large number of processes by a user
- Users trying to access security auditing functions and files without permission or authorization

Outsider attacks are mainly from computers connected through Internet. Intelligence departments all over the world collect information exploiting the security flaws in the Internet and by attracting users through irresistible web page contents. Any computer system connected to the insecure Internet is vulnerable to attack, and Internet is a comfortable place for attackers to engineer attacks and remain anonymous. Easier-to- use automated tools are available for attackers to plant attacks that cause huge destructions in minutes. There are different categories of external attackers, such as freelance information brokers, foreign or domestic competitors, terrorist organizations and crime syndicates.

A significant threat of recent importance is the use of Internet for Information warfare. Information warfare weapons are changing the nature of war from the traditional gunpowder and high-cost high-tech nuclear weapons. Information warfare weapons are low cost and affordable even by small countries or groups. The operations of telecommunications, power, transportation and financial systems are now increasingly linked to the Internet, and they are the targets of attacks from abroad exploiting the vulnerabilities of Internet connected computers. Even a small terrorist organization can attack computers of defence department of a highly developed nation for destroying and stealing sensitive information which may amount to weakening the security of military communication and consequent failures in operations.

Attacks from Inside the System

An insider is a person who has logged into a computer using legitimate username and password. In a system with long and special symbol based passwords, breaking them to login may be difficult. However, week passwords may be broken easily and the intruder can cause damages to the user information or steal them. A person who has logged in can also exploit the system vulnerabilities (bugs or loop holes) to gain entry into other users' area including administrators and work in the system with their privileges. These attackers are also called crackers, as they break the password system to gain unauthorized entries into other users' area. Attacks by insiders include:

- Trojan Horses
- Login Spoofing
- Logic Bombs
- Trap Doors
- Buffer Overflow

In this section, we will discuss the attacks engineered by attackers or crackers.

Trojan Horses

A Trojan Horse is a program that appears legitimate and innocent but performs illicit activity when it is run, such as stealing passwords, making the system more vulnerable to future entry or simply destroying programs or data on the hard disk. This program may even steal valuable information and send it to the site of the attacker. The Trojan has to be placed in some directory of the user and get executed from there without the knowledge of the user. An easy technique is to place the program on the Internet site as freely downloadable attractive programs like games or other useful applications. When users download and execute the program, the Trojan is executed do all nasty things like removing files or reading passwords or send information to the attacker's site.

There are various other ways to get the Trojan executed many times without the knowledge of the user. In UNIX, when we type a program for execution the system searches the program in the directories given under the PATH environment variable. Suppose the attacker is able to put the Trojan in any of those directories, for example to bin directory, with a name (*ld*) that represent common typing errors for a very often executed command *ls* (list directory). Now, if the user mistakenly types *ld* instead of *ls*, the Trojan gets executed, and it does all disastrous things for which the Trojan is programmed to do by the attacker. If the user now reports to the supervisor of the system about the problem like loss of some files, the supervisor may try to execute the *ls* command in the user directory with super user privilege. If the super user, by chance, types *ld* out of the same mistake in typing *ls,* the Trojan gets executed with super user privileges, acquiring the power to access and destroy the entire system. The Trojan may now replace the genuine version of *ls* with one containing a Trojan horse inside so that the Trojan will get executed when ever any user type *ls* command.

Login Spoofing

This is a technique for collecting usernames and passwords of users of the system by an attacker who is an ordinary user of the system. The attacker or cracker logs in to the system and executes a program which displays a login window exactly looking like that of normal login window of the system. He then leaves the seat and may work on some other terminal in the multiuser system. When a new user sits down and enters the username and password, the hacker program collects the same and stores in a file, and then sends a signal to kill its shell. This logs out the attacker and triggers the execution of the real login program which displays the real login window. The user, unaware of the fact that his username and password are stolen, now again enters his username and password to login to the system and works as usual. The cracker may do the trick on other terminals many times to collect the username and password pairs of all users of the system.

If the users start the login sequence by pressing a key combination that the user program cannot catch, this spoof attack can be bypassed or prevented. Windows system uses *control-alt-del* keys combination for this purpose.

Logic Bombs

This is a piece of code that programmers (who are current employees) of a company secretly inserted into the companies production operation system or companies applications. The logic in the inserted code will be to bring down the whole system when the programmer gets fired, so that the company will be forced to rehire him to set the things right. For example, as long as the programmer logs in everyday or alternate days, the system functions normally. If the programmer did not login for, say two continuous days, the logic bomb goes off leading to things like erasing files at random and malfunctioning of the whole system.

Trap Doors

Trap Door is another security hole caused by the programmer. This is done by secretly inserting some code to the operating system (or application) code that bypass some normal check. For example, a programmer may add code to the login program to login using a name 'SAHARA' whatever be the password string. So, the programmer can login to computers of any company that loads this operating system or application.

NOTES

Buffer Overflow

Another major source of attack is the buffer overflow vulnerability of some programming languages like C and C++. These programming languages do not provide any built-in protection against accessing or overwriting data in any part of memory. There is no array bounds check done while writing data to arrays. Thus any attempt to store data beyond the boundaries of fixed-length buffer overwrites adjacent memory locations which may be part of other buffers, variables, return address of functions, etc. This may lead to erratic program behaviour, incorrect results or program crash. Attackers exploit the buffer overflow to inject code and get it executed to gain illegal entry to the system even with supervisor privileges.

Example: Suppose a C++ program has the following data structures defined:

```
int A = 35; char B[10]; int C=33; int D=9;
```

Assume that integers take two bytes each, and characters take 1 byte each. B is initially empty; means B contain all null characters (zero value in all elements of B). The memory allocated for A, B, C and D with the values in the memory locations will be as given below:

A		B										C		D	
00	35	00	00	00	00	00	00	00	00	00	00	00	33	00	09

Suppose the program during execution copies the string “BufferOverRun” to the character array B. This copies 13 characters of the string and the null character (ASCII value 00) to the array B. The result of the copy operation will be as given below:

A		B										C		D	
00	35	‘B’	‘u’	‘f’	‘f’	‘e’	‘r’	‘O’	‘v’	‘e’	‘r’	‘R’	‘u’	‘n’	00

The contents of variables C and D are overwritten and their values become different. If these variables are allocated on stack, then it will lead to stack buffer overflow and may cause overwriting other variables and return address of functions. If the buffer is sufficiently large, an intruder can even copy the code of a program to the buffer and the return address on the stack may be overwritten with the start address of this intruder program. Thus, the intruder program gets the privilege of the program which is broken.

There are other techniques to exploit buffer overflow like heap overflow and NOP sled. Interested readers may refer to *Wikipedia* for details on the topic.

Attacks from Outside the System

Major security threats of outsider attacks are through viruses and worms. These can easily enter in to the system and spread to other systems through the Internet.

Viruses

A **virus** is a program fragment that is attached to legitimate popular programs like games or other utilities with the intention of infecting other programs. Normally, the virus code segment will be attached to the beginning part of the executable file so that, when anybody executes it, the virus code will b executed first causing damages to the system. The virus segment also includes the code to search for other executable files in the system and add itself to the beginning part of those files. In this way, after a few hours of infecting one of the files in the system, most of the executable files in the system might get the virus infection leading to wastage of CPU time, and other resources. As a consequence, the system response will become unacceptably low. A virus writer can also include in it the code to cause damages to data and delete files. For a virus to spread to another computer, one must first place a virus affected file in that computer manually or through emails.

The viruses are given names based on the method of attack or the media attacked or the location they reside. Following are some of the known viruses:

- Companion virus
- Executable program virus
- Memory resident virus
- Boot sector virus
- Device driver virus
- Macro virus
- Sources code virus

A detailed description of these can be found in the book by Andrew S Tanenbaum for further reading.

Worms

A **worm** is also like virus, but it can automatically spread to other computers through the Internet. A worm has two parts, the *bootstrap* code and the *main worm code*. The main worm code does all damages to the system under attack. The bootstrap code may be inserted to the machine exploiting the bugs in the system. When the bootstrap code gets executed on the system under attack, it gets connected to the attacker's machine from which it came. The attacker's machine uploads the main worm to the system under attack and executes it there causing damages to the system. The worm then hides its presence and then spreads the worm to other machines by sending the bootstrap code to all other machines connected to the attacked machine. The process of attack again continues from its new location. Thus, the whole networked machines can be brought down in a few hours.

13.3 CRYPTOGRAPHY AND ENCRYPTION

The huge growth of the Internet has changed the way businesses are conducted. Buying and selling of goods have changed from the traditional ways to Internet-based techniques. Sellers need not display items in a physical showroom to lure buyers. Sellers display the items on web pages which can be viewed by buyers all over the world sitting in front of their Internet connected computers. Orders and payments can be made through computers (using credit card numbers or bank accounts), and the seller can dispatch the item to the homes of buyers quickly. However, the security of sensitive information like credit card numbers and pins sent over the Internet is a serious concern for both the sellers and buyers. Other sensitive information we may send over internet may include social security number, private correspondence, personal details and company information. These informations are to be secured during transit over Internet for the security of the buyers and sellers.

There are a number of techniques by which information is secured during transactions and during communications between parties over the Internet. Encoding the sensitive information before transmission over the Internet using some key called the *Encryption* is the widely used technique in today's secure transactions and communications. The received information may be decrypted using decoding key to view and use the information in the original form. The process of encryption and decryption falls under the traditional topic of *cryptography*.

Cryptography is the process of representing information using secret codes for providing security and confidentiality of information in a system. Cryptography is used to encrypt a plain text like ASCII strings into the ciphertext (coded form of plaintext) in such a way that only the authorized people know how to convert it back to the plaintext.

In the rest of this section, we will learn about encryption-decryption techniques using public-key system and symmetric-key system.

Encryption

Encryption is the process of transforming information (referred to as plaintext) using an algorithm (called cipher) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key. The result of the process is the encrypted information (in cryptography, referred to as ciphertext). That is, the encryption is actually the coding process of cryptography. However, in many contexts, the word *encryption* also implicitly refers to the reverse process, decryption**.** For example, software for encryption can typically also perform *decryption*, to make the encrypted information back to readable form. In a loose sense, encryption and cryptography actually mean the same thing in the context of computer security.

Encryption has long been used by militaries and governments to facilitate secret communication. Encryption is now used in protecting information within many kinds of civilian systems, such as computers, storage devices, networks, mobile telephones, wireless microphones, wireless intercom systems, Bluetooth devices and bank automatic teller machines. Encryption is also used in digital rights management to prevent unauthorized use or reproduction of copyrighted material and in software also to protect against reverse engineering.

Encryption can protect the confidentiality of messages, but other techniques like digital signatures are still needed to protect the integrity and authenticity of messages.

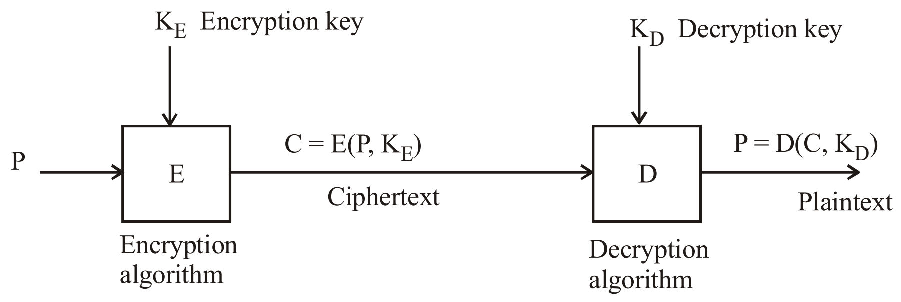


Fig. 13.1 *Encryption-Decryption Process and Relationship between Plaintext and Ciphertext*

Let us look at the process of encryption and decryption in greater detail. Figure 8.1 depicts the encryption and decryption processes and the relationships between plaintext and ciphertext. The encryption algorithm, E takes the plaintext (information in the normal readable form), P and a key, called encryption key, K_E to produce the ciphertext (coded form- not in a readable form for ordinary users), $C= E(P, K_E)$. Information in the form of ciphertext is not useful for normal users. However, the ciphertext can be converted to the original plaintext using the decryption algorithm, D and the decryption key K_D. That is, the plaintext $P= D(C, K_D)$. The secrecy of the information depends on the keys used. Depending up on the number of keys used, we have two types of encryption techniques as given below:

- Symmetric-key encryption or Secret-key encryption
- Public-key encryption

Each of these techniques is described below.

Symmetric-key Encryption

In the symmetric-key encryption technique, both the parties involved in the communication uses the same key to send information between them. Each computer has a secret key (code) that it can use to encrypt a packet of information before it is sent over the network to another computer. Symmetric-key requires that you know which computers will be talking to each other so you can install the key on each one. Symmetric-key encryption is essentially the same as a secret code that each of the two computers must know in order to decode the information. The code provides the key to decoding the message.

Let us look at an example to illustrate the process of symmetric-key encryption. A coded message (ciphertext) is create by substituting each letter in the message by a letter two down in the alphabet. That is, replace 'A' by 'C', 'B' by 'D', 'C' by 'E' and so on. So, the secret code is 'shift by 2'. The secret key should be told to the person (receiver) to whom you want to send the coded message so that he can decode the message and read it correctly. Anybody else receiving the message cannot make a sense out of it and hence cannot read it. However, it is possible to extract such simple secret keys just by analysis of the ciphertext using statistical properties of natural languages. Computers can easily break the ciphertext produced using simple keys. So, keys must be much longer to use with computers.

The first major symmetric algorithm developed for computers in the United States was the Data Encryption Standard (DES), approved for use in the 1970s. The DES uses a 56-bit key. Because computers have become increasingly faster since the 1970s, security experts no longer consider DES secure, although a 56-bit key offers more than 70 quadrillion possible combinations (70,000,000,000,000,000), an attack of brute force (simply trying every possible combination in order to find the right key) could easily decipher encrypted data in a short while. DES has since been replaced by the Advanced Encryption Standard (AES), which uses 128-, 192- or 256-bit keys. Most people believe that AES will be a sufficient encryption standard for a long time coming: A 128-bit key, for instance, can have more than 300 billion-trillion-trillion key combinations. However, the problem with symmetric-key (secret-key) cryptography is the difficulty to send the secret key to the receiver of the information.

Public-key Encryption

Public-key encryption is also known as ***asymmetric-key*** encryption. It uses a pair of keys—a private key and a public key so that one is used for encryption and the other is used for decryption. The private key is known only to your computer, while the public key is given by your computer to any computer that wants to communicate securely with it. As the keys are chosen based on the prime long numbers, it is very difficult to guess the private key from the known public key. This makes the system extremely secure, because there is essentially an infinite number of prime numbers available, that is, there are nearly infinite possibilities for keys. To send a secret message, the message is first encrypted by the receiver's public key and then sends to the receiver. The receiver uses its own private key to decode the message. No other parties can decode the message without the private key of the receiver. So, the receiver keeps its private key secret, the secrecy or confidentiality of the message will not be lost. ***Pretty Good Privacy (PGP)* is an example of public-key encryption system.**

Large scale implementation of public-key encryption requires support from independent ***certificate authority*** to issue digital certificates for trusting web servers/ users involved in the communication. A digital certificate is basically a unique piece of code or a large number to certify that a web server/ user is trusted. The certificate authority acts as a middle man for authenticating the computers involved in the communication. The certificate authority provides its public key to each of the parties (computers) wishing to communicate. For sending messages between two computers, the certificate authority provides the digitally signed name and public key of each of the computers to the other.

SSL and TLS

A brief understanding of SSL (Secure Socket Layer) and TLS (Transport Layer Security) is essential for people who use online banking and other transactions. SSL is a popular implementation of public-key encryption. SSL was originally developed by Netscape as an Internet security protocol used by Internet browsers and web servers to transmit sensitive information. SSL has now become part of the TLS. TLS is known as the overall security protocol for Internet communication.

In the browser, we can make out when we are using a secure protocol, such as TLS by two ways. We can notice that the 'http' in the address line is replaced with 'https,' (*https* stands for secure *http*) and we can see a small padlock in the status bar at the bottom of the browser window. When we are accessing sensitive information, such as an online bank account or a payment transfer service like PayPal or Google Checkout, we can see this type of format change which tells us that the information will be passed securely along the communication path.

TLS and its predecessor SSL make significant use of *certificate authorities*. Once the browser requests a secure page and adds the "s" onto "http", the browser sends out the public key and the certificate, checking three things: 1) that the certificate comes from a trusted party; 2) that the certificate is currently valid; and 3) that the certificate has a relationship with the site from which it is coming.

The browser then uses the public key to encrypt a randomly selected symmetric key. Public-key encryption takes a lot of computing, so most systems use a combination of public-key and symmetric key encryption. When two computers initiate a secure session, one computer creates a symmetric key and sends it to the other computer using public-key encryption. The two computers can then communicate using symmetric-key encryption. Once the session (communication session) is finished, each computer discards the symmetric key used for that session. Any additional sessions require that a new symmetric key be created, and the process is repeated.

Digital Signature

In the paperless office, there is no way to use the normal signature to keep as proof of having issued some orders or agreed up on a contract or done some other commitments by people involved. After having done something like issuing an order or instruction, the denial of the same at a later occasion is called repudiation. So, it is necessary to sign documents digitally to prevent repudiation of orders issued or commitments made through emails or other messages in the paperless office system.

13.4 AUTHENTICATION

The security of a system involves two parts: one, the task of authenticating a subject (process or people) who attempts to access a secure entity, and two, the task of determining if the subject is authorized to have access to each specific secure entity. When a user logs on to a computer, the operating system wants to determine who the user is. This process is called user ***authentication***.

More formally, *authentication* is the task of ensuring that a subject who attempts to access the secure entity is actually the subject that it claims to be.

Authorization is that task after authentication to ensure whether the subject has the right to access a secure entity in the system. That is, a person is an authenticated user of the system, but has he got (authorized) the right, for example, to print on the laser printer?

Most applications today need to access information over networks such as a LAN (local area network) or WAN (wide area network). The information may be secure inside the computer hardware, but the networks are open to anybody to peep in. So, another important aspect of security and protection is to ensure that the information is not copied or confidentiality is not compromised while in transit. The modern technique to ensure the security of information in transit is through cryptography. Cryptography can also protect information in persistent storage state.

Most authentication techniques are based on the general principle of identifying what he (the subject accessing the resource) knows or what he has or what he is. This leads to various authentication schemes with varying complexities and security properties. Some of the major schemes are:

- Authentication using password
- Authentication using physical object
- Authentication using biometrics
- Authentication using digital signatures

We will briefly discuss each of these schemes below.

Authentication using password

The use of a user name and password provides the most common form of authentication. You enter your name and password when prompted by the computer. It checks the pair against a secure file to confirm. If either the name or the password does not match, then you are not allowed to proceed further.

NOTES

Authentication using physical object

These physical objects are made in the form of plastic cards with magnetic strips for recording information. User identification information is recorded in these cards. The card has to be inserted into a scanning machine for it to read and authenticate the user like the credit cards. These pass-cards can be even a sophisticated one like a smart card with more memory to carry detailed information and having embedded computer chip inside.

Authentication using biometrics

Biometric techniques are used recently in home and office computers for effective authentication. Biometric authentication is based upon physical or behavioural attributes of individuals. There are many biometric features that can be used, depending on the sophistication required in the authentication process. Biometric authentication techniques include:

Fingerprint scan

Retina scan

Face scan

Voice identification

Keystroke dynamics based identification, and

Palm-print based identification

Authentication using digital signature

A digital signature is basically a way to ensure the authenticity of electronic documents such as e-mail, spreadsheet, text file and Java Applets. We have ***digital signature standard*** based on a combination of public-key and private-key encryption techniques and employing a ***digital signature algorithm***. *Digital signature standard* is the format for digital signatures that has been approved by many countries. The *digital signature algorithm* consists of a private key, known only by the originator of the document (the signer), a public key (known to others) and a hash algorithm. The digital signature is attached as a *signature block* to the document before sending it to the destination. Any change made to the document during the transit (by a snooper or eavesdropper or intruder) causes a change in the digital signature when re-computed at the receiving end. The snooper will not be able to change the signature block appropriately to hide the change in the document as it requires the private key of the sender. Therefore, by comparing the signature in the signature block of the document and the re-computed signature enable the receiver of the document to determine any integrity violation of the document made during transit. We will discuss this topic further in the section on cryptography.

NOTES

Check Your Progress

1. What is login spoofing?
2. What is a logic bomb?
3. What is cryptography?
4. How many keys are used in symmetric encryption?
5. What is a digital certificate?
6. What is user authentication?

13.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Login spoofing is a technique for collecting usernames and passwords of users of the system by an attacker who is an ordinary user of the system.
2. Logic bomb is a piece of code that programmer (who is a current employee) of a company secretly inserted into the companies production operation system to bring down the entire system when the programmer is fired (when he losses his job).
3. Cryptography is the process of representing information using secret codes for providing security and confidentiality of information in a system.
4. Only one key called the secret-key is used in symmetric encryption.
5. A digital certificate is basically a unique piece of code or a large number to certify that a web server or user is trusted.
6. When a user logs on to a computer, the operating system wants to determine who the user is. This process is called user A*uthentication.*

13.6 SUMMARY

- Attacks on a system may be from insiders who have logged into the system or outsiders who prompt an innocent legitimate user to download free of cost a malicious program such as Trojan horses or Viruses or Worms masquerading as an exciting game or music.
- An insider is a person who has logged into a computer using legitimate username and password.
- A Trojan Horse is a program that appears legitimate and innocent but performs illicit activity when it is run, such as stealing passwords, making the system more vulnerable to future entry or simply destroying programs or data on the hard disk.

- Login spoofing is a technique for collecting usernames and passwords of users of the system by an attacker who is an ordinary user of the system. The attacker or cracker logs in to the system and executes a program which displays a login window exactly looking like that of normal login window of the system.
- A **virus** is a program fragment that is attached to legitimate popular programs like games or other utilities with the intention of infecting other programs.
- ***Cryptography*** is the process of representing information using secret codes for providing security and confidentiality of information in a system.
- ***Encryption*** is the process of transforming information (referred to as plaintext) using an algorithm (called cipher) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key.
- In the symmetric-key encryption technique, both the parties involved in the communication uses the same key to send information between them.
- Public-key encryption is also known as ***asymmetric-key*** encryption. It uses a pair of keys—a private key and a public key so that one is used for encryption and the other is used for decryption.
- The security of a system involves two parts: one, the task of authenticating a subject (process or people) who attempts to access a secure entity, and two, the task of determining if the subject is authorized to have access to each specific secure entity.

13.7 KEY WORDS

- **Cryptography**: The process of representing information using secret codes for providing security and confidentiality of information in a system.
- **Encryption**: The process of transforming information (referred to as plaintext) using an algorithm (called cipher) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key.
- **User Authentication:** The process in which if a user logs onto a computer, the operating system wants to determine who the user is.
- **Authorization:** The task after authentication to ensure whether the subject has the right to access a secure entity in the system.

13.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. List three objectives of computer security.
2. What are the important measures for safeguarding system security?

3. What are the major biometric techniques for authentication?
4. What is the process of authentication using digital signature?
5. What are the popular techniques of insider attacks?
6. What are Trojan horses? How do they enter into the system and start attacks?
7. How does Login spoofing collect usernames and passwords?
8. Explain the working of logic bombs.
9. What are virus programs? How do they spread into other files?
10. What is a worm? Explain the intrusion techniques of worms.
11. What are the applications of encryption/cryptography?
12. What is a digital signature? How can you digitally sign a document?
13. What is symmetric-key encryption?
14. What is public-key encryption?
15. How can you enforce non-repudiation using digital signatures in a paperless office?
16. What are the advantages of public-key encryption over symmetric-key encryption?

Long Answer Questions

1. What are the four major classes of techniques for user authentication? Explain each.
2. What are the popular techniques of insider attacks? Explain each of them.
3. Describe the encryption and decryption processes in cryptography.

13.9 FURTHER READINGS

Tanenbaum, Andrew S. and Maarten Van Steen. 2006. *Distributed Systems: Principles and Paradigms*. New Jersey: Prentice Hall.

Garg, Vijay K. 2002. *Elements of Distributed Computing*. New Jersey: Wiley-IEEE Press.

Sinha, Pradeep K. 1996. *Distributed Operating Systems: Concepts and Design*. New Delhi: Prentice-Hall of India.

UNIT 14 ACCESS CONTROL AND DESIGN PRINCIPLES

NOTES

14.0 INTRODUCTION

Nowadays, most of the organizations serving domains such as banking, education, finance and telecommunication rely on the use of computers for their day-to-day activities. These organizations store huge amount of data in computers. Since the data is highly valuable, it is important to protect it from unauthorized access. In addition to data, the protection of computer resources, such as memory, I/O devices, etc., is also necessary. In this unit, you will learn about the protection mechanisms, digital signature and design principles of security.

14.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain the two protection mechanisms i.e. protection domain and access control list
- Discuss how to create digital signature
- Discuss the design principles of security

14.2 PROTECTION MECHANISM AND ACCESS CONTROL

One of the aspect that operating system provides is protection, which deals with protecting user's data and programs from other user's interference. Implementing protection requires policies and mechanisms. Policy decides which data should be protected from whom and mechanism specifies how this policy is to be enforced. In the discussion of protection, we focus on mechanism rather than on policy because policy may change from application to application.

There are many protection mechanisms used in a system, each having some advantages and disadvantages. However, the kind of protection mechanism used depends on the need and size of the organization. In this section, we will discuss two protection mechanisms, namely, *protection domain* and *access control list*.

Protection Domain

A computer system consists of a set of objects that may be accessed by the processes. An **object** can be either a hardware object (such as, CPU, memory segment and printer) or software object (such as, file, database, program and semaphore). Each object is referred to by a unique name and is accessible by the processes using some pre-defined operations. For example, a process can perform `wait()` and `signal()` operations on a semaphore object.

Since an object is the basic component of a computer system, a mechanism is required to ensure that a process accesses only those objects for which it has got permission from the operating system. Moreover, it must be ensured that a process performs only those operations on the object that it currently requires to complete its task. To facilitate this, the concept of protection domain is used which specifies the objects that a process may access.

A **domain** is a collection of access rights where each access right is a pair of `<object-name,rights_set>`. The `object_name` is the name of the object and `rights_set` is the set of operations that a process is permitted to perform on the `object_name`. For example, a domain `D` with access right `<A,[R, W]>` specifies that any process in domain `D` can perform read and write operation on the object `A`.

A system can specify a number of domains. These domains may be disjoint or share access rights with each other. To understand this, consider Figure 14.1, which shows three domains `D1`, `D2` and `D3` with five objects `A`, `B`, `C`, `D` and `E`. The domain `D1` is disjoint while domains `D2` and `D3` share the access right `<C,[Print]>` and thus, overlap.

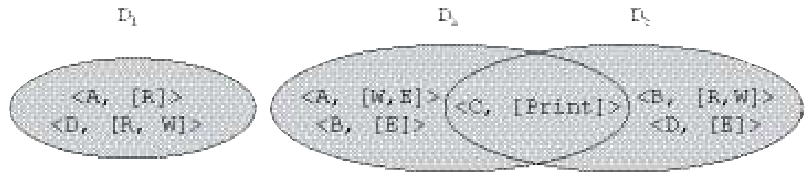


Fig. 14.1 *Protection Domain*

From this figure, it is clear that a process in domain `D1` can only read the object `A`; however, a process in domain `D2` can write as well as execute the object `A`. In addition, a process executing in either of the domain `D2` and `D3` can print the object `C`.

Each process, at a given time, executes in some protection domain with access to objects specified in domain and the specified set of rights on those objects. The association between a process and domain may be either **static** or

dynamic. In the former case, a process is allowed to access only a fixed set of objects and rights during its lifetime, while in the latter case, the process may switch from one domain to another during its execution (termed as **domain switching**).

14.2.1 Access Control List (ACL)

Access control list is an alternative method of recording access rights in a computer system. It is often employed in file systems. In this method, a list is associated with each object such as file that stores user names (or processes) which can access the object and the type of access allowed to each user (or process). This list is known as **Access Control List** (**ACL**). When a user (or process) tries to access an object, the ACL is searched for that particular object. If that user is listed for the requested access, the access is allowed. Otherwise, the user is denied access to the file. Figure 14.2 shows a sample access control list for five files and four users. It is clear from the figure that user `A` has access to `File 1`, `File 2` and `File 5`, user `B` has access to `File 1`, `File 2` and `File 3` and user `C` has access to `File 2`, `File 3` and `File 4`.

FIle name		Protection info
File 1	→	A:R; B:RWE
File 2	→	A:RW; B:R; C:RE
File 3	→	B:RWE; C:R; D:R
File 4	→	C:RW; D:RWE
File 5	→	A:RWE; D:RW

Fig. 14.2 *A Sample Access Control List*

This system of access control is effective but, in case if all users want to read an object, say the file `F`, the ACL for this file should list all users with read permission. The main drawback of this system is that, making such a list would be a tedious job when number of users is not known. Moreover, the list needs to be dynamic in nature as the number of users will keep on changing, thus, resulting in complicated space management.

To resolve the problems associated with ACL, a restricted version of the access control list can be used in which the length of the access control list is shortened by classifying the users of the system into the following three categories.

- **Owner:** The user who created the file.
- **Group:** A set of users who need similar access permission for sharing the file is a group, or work group.
- **Universe:** All the other users in the system form the universe.

Based on the category of a user, access permissions are assigned. The owner of the file has full access to a file; and can perform all file operations (read, write and execute) whereas, a group user can read and write a file but cannot execute or delete a file. However, the member of the universe group can only read a file and is not allowed to perform any other operations on a file.

The above method of classifying users in groups will not work, when one user wants to access file of other user (for performing a specific file operation). For example, say, a user `comp` wants to access the file `abc` of other user `comp1`, for reading its content. To provide file-specific permissions to a user, in addition to the user groups, an access control list is attached to a file. This list stores the user names and permissions in a specific format.

The UNIX operating system uses this method of access control, where the users are divided into three groups, and access permissions for each object is set with the help of three fields. Each field is a collection of bits where, three bits are used for setting protection information and an additional bit is kept for a file owner, for the file's group and for all other users. The bits are set as `–rwx` where `r` controls read access, `w` controls write access and `x` controls execution. When all three bits are set to `–rwx`, it means a user has full permission on a file whereas, if only `–r-- f`ield is set, it means a user can only read from a file and when `–rw-` bits are set, it means user can read and write but cannot execute a file. The scheme requires total nine bits, to store the protection information. The permissions for a file can be set either by an administrator or a file owner.

14.3 DIGITAL SIGNATURES

Digital signatures can be generated by simply taking the mathematical summary of the message, which will give a fixed size message known as hash code. The hash code is used for identifying the message and if even small changes take place in the original message then this will dramatically change the hash code. The next step is to sign this hash code with the private key of the sender. This signed message is now appended with the original message and sent.

The receiver of this message can verify the signature using the sender's public key. The encrypted hash code is decrypted first and the new hash code is generated using the appended original message. Now, the received hash code and the new computed hash code at the receiver end is compared. If the hash codes are same, then the receiver has verified that the message has not been altered. This also ensures authentication because only that sender has the private key which was used for signing. Digital signatures are used for providing authentication as well as integrity of the data.

Hash algorithms, such as MD5 and SHA, are used for generating the hash code. For these algorithms, an arbitrary length message is the input and fixed length message is the output. These are one-way functions. It is computationally infeasible to find two different messages which produce the same hash value.

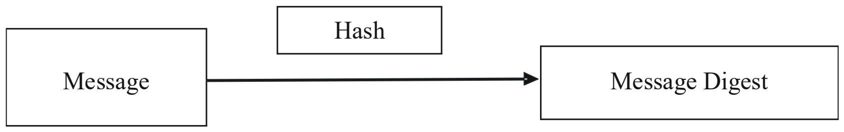
There are many other digital signature algorithms, such as RSA, DSA, ElGamal signature scheme, SHA with RSA, ECDSA, Rabin Signature algorithm, and so on.

For example,

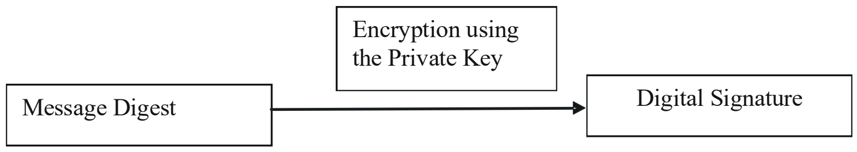
Alice has two keys, one is the public key and the other is a private key. Anyone can access Alice's public key and the private key is only known to Alice. These keys are used to encrypt the information. Any one of the keys can be used for encryption and the other one for decryption.

Any of Alice's co-workers can encrypt a message using her public key and any person who knows the public key may access that encrypted message. But without knowing the private key it is worthless, as no one can decrypt it and know the contents of the original message.

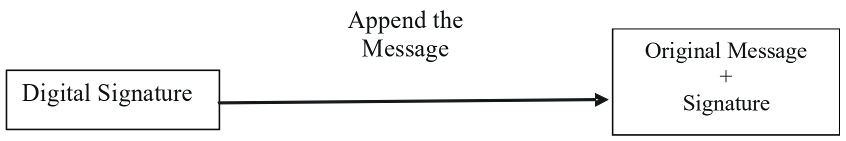
With the private key, Alice can append the digital signature with the original data which is unique and difficult to forge.



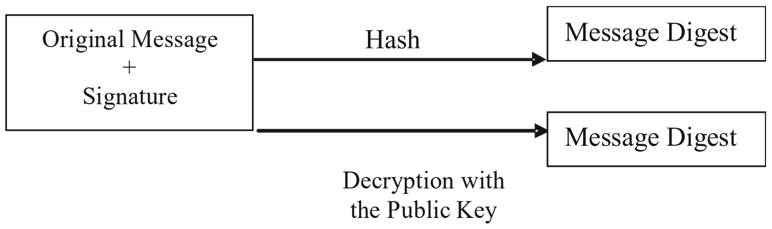
Alice will first generate the small fixed size message by applying the hashing. This fixed size message is called message digest.



Alice will then encrypt the message digest with the private key which will give the digital signature.



Now digital signature is appended with the original message.



At the receiver side, Bob decrypts the signature using Alice's public key which will produce the message digest. If it works then it proves that Alice is the sender because only he has the private key. Now, Bob generates the hash code of the message, i.e., message digest and if this message digest is the same as the message digest created by decrypting the signature then Bob confirms that the message has not been changed.

The following are some of the applications of digital signatures:

(a) Authentication

Messages may sometimes include the information about the sender of the message but that information may not be correct. Digital signatures, therefore, can be used to authenticate the sender of message. Valid signature shows that the message was sent by that sender only who is claiming to be a sender because there is only one unique owner of a specific digital signature. Digital signatures are most often used in the context of financial matters.

(b) Integrity

Digital signatures may be used in such cases when the sender and receiver of a message need assurance that the message is not altered by anyone during transmission. Message is transmitted in the encrypted form which conceals the matter of the message, but it is possible to change the message without the knowledge of it. Non-malleable encryption algorithms prevent this but not others. So if digital signatures are used then any change in the contents of the message will invalidate the signature. It is not feasible to modify a message and its signature which can validate the signatures at the receiver end.

(c) Private Key Storage on a Smart Card

Security of public and private key cryptosystems depends mainly on the secrecy of the private key. This private key may be stored on an end-user's computer and can be protected by that local machine's password. It has some disadvantages as the sender can digitally sign the document on that computer and the security of the key depends on the security of that local machine.

One way to provide more security is to save the private key on the smart card. Smart cards are designed so that the data stored in it cannot be altered but some of these designs have already been broken. In the implementation of the digital signature the hash code is sent to the smart card, CPU will now encrypt this hash code using the private key and returns the encrypted hash code. A user can activate the smart card using his personal identification number, i.e., a PIN code which provides the two factor authentication. If somehow the smart card is stolen then the attacker will also need the PIN number to generate the digital signature. So access of any one of the PIN code or private key will not work. Loss of the smart card may de detected very easily and the owner will revoke the corresponding certificate immediately.

A numeric keypad is required for entering a PIN code to activate the smart card. Some card readers hold their individual numeric keypad which is safe in comparison to using a keyboard integrated to a system. The main difference between a digital signature and written signature is that in the case of digital signature a user cannot see the sign, while in written, he can see it.

One of the main differences between a digital signature and written signature is that the user does not 'see' what he signs. The user application presents a hash code to be encrypted by the digital signing algorithm using the private key. An attacker who gains control of the user's PC can possibly replace the user application with a foreign substitute, in effect replacing the user's own communications with those of the attacker. This could allow a malicious application to trick a user into signing any document by displaying the user's original on screen, but presenting the attacker's own documents to the signing application. A digital signature is applied to a string of bits. Humans and applications believe that digital signature is the semantic interpretation of those bits which will be meaningful sentences. WYSIWYS (What You See Is What You Sign) is the property which is required for digital signatures which says what you see is what you sign. It means that the signed message does not contain any type of hidden information that the signer does not know about. But it is difficult to guarantee that this property will hold because the complexity of the computer systems is increasing very fast.

14.4 DESIGN PRINCIPLES FOR SECURITY

Designing a secure operating system is a crucial task. While designing the operating system, the major concern of designers is on the internal security mechanisms that lay the foundation for implementing security policies. Researchers have identified certain principles that can be followed to design a secure system. Some design principles presented by Saltzer and Schroeder (1975) are as follows:

- **Least Privilege:** This principle states that a process should be allowed the minimal privileges that are required to accomplish its task.
- **Fail-Safe Default:** This principle states that access rights should be provided to a process on its explicit requests only and the default should be no access.
- **Complete Mediation:** This principle states that each access request for every object should be checked by an efficient checking mechanism in order to verify the legality of access.
- **User Acceptability:** This principle states that the mechanism used for protection should be acceptable to the users and should be easy to use. Otherwise, the users may feel a burden in following the protection mechanism.
- **Economy of Mechanism:** This principle states that the protection mechanism should be kept simple as it helps in verification and correct implementations.

- **Least Common Mechanism:** This principle states that the amount of mechanism common to and depended upon by multiple users should be kept as minimum as possible.
- **Open Design:** This principle states that the design of the security mechanism should be open to all and should not depend on ignorance of intruders. This entails to the use of cryptographic systems where the algorithms are made public while the keys are kept secret.
- **Separation of Privileges:** This principle states that the access to an object should not depend only on fulfilling a single condition; rather more than one condition should be fulfilled before granting an access to the object.

Check Your Progress

1. What is a domain?
2. What is the use of access control list?

14.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A domain is a collection of access rights where each access right is a pair of `<object-name,rights_set>`.
2. An access control list, for each object, stores the user names (or processes) and the type of access allowed to each user. When a user (or process) tries to access an object, the ACL is searched for that particular object. If that user is listed for the requested access, the access is allowed. Otherwise, the user is denied access to the file.

14.6 SUMMARY

- One of the aspect that operating system provides is protection, which deals with protecting user's data and programs from other user's interference.
- There are many protection mechanisms used in a system, each having some advantages and disadvantages. However, the kind of protection mechanism used depends on the need and size of the organization.
- A **domain** is a collection of access rights where each access right is a pair of `<object-name,rights_set>`. The `object_name` is the name of the object and `rights_set` is the set of operations that a process is permitted to perform on the `object_name`.

- Access control list is an alternative method of recording access rights in a computer system. It is often employed in file systems. In this method, a list is associated with each object such as file that stores user names (or processes) which can access the object and the type of access allowed to each user (or process).
- Digital signatures can be generated by simply taking the mathematical summary of the message, which will give a fixed size message known as hash code. The hash code is used for identifying the message and if even small changes take place in the original message then this will dramatically change the hash code. The next step is to sign this hash code with the private key of the sender. This signed message is now appended with the original message and sent.
- Designing a secure operating system is a crucial task. While designing the operating system, the major concern of designers is on the internal security mechanisms that lay the foundation for implementing security policies.

14.7 KEY WORDS

- **Domain:** A collection of access rights where each access right is a pair of `<object-name,rights_set>`.
- **Access Control List (ACL):** A method of recording access rights in a computer system that is often employed in file system.

14.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer Questions

1. What do you mean by domain switching?
2. Access control list is an alternative method of recording access rights in a computer system. Justify this statement.
3. What is the use of digital signature? How does it work?

Long Answer Questions

1. Explain protection mechanism illustrating use of protection domain and access control list.
2. What is the importance of design principles for security? Explain some of these principles.

NOTES

14.9 FURTHER READINGS

Tanenbaum, Andrew S. and Maarten Van Steen. 2006. *Distributed Systems: Principles and Paradigms*. New Jersey: Prentice Hall.

Garg, Vijay K. 2002. *Elements of Distributed Computing*. New Jersey: Wiley-IEEE Press.

Sinha, Pradeep K. 1996. *Distributed Operating Systems: Concepts and Design*. New Delhi: Prentice-Hall of India.