MASTER OF COMPUTER APPLICATION

MCA-32

ADVANCED OPERATING SYSTEMS

Directorate of Distance Education
Guru Jambheshwar University of Science & Technology, Hisar – 125001

CONTENTS

Course: MCA-32	Writer: Ritu
Lesson: 1	Vetter:

Introduction to Operating System

Structure

1.0 Learning Objectives

The users of this lesson have already gone through some concepts of operating system in the first year of the course. So the objectives of this lesson are:

(a) To review the basic concepts of operating system i.e. definition, types, and functions performed by them.

(b) To review the process scheduling policies.

1. 1 Introduction

Operating System may be viewed as collection of software consisting of procedures for operating the computer and providing an environment for execution of programs. The main goals of the Operating System are:

(i) To make the computer system convenient to use,

(ii) To make the use of computer hardware in efficient way.

Basically, an Operating System has three main responsibilities:

(a) Perform basic tasks such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices such as disk drives and printers.

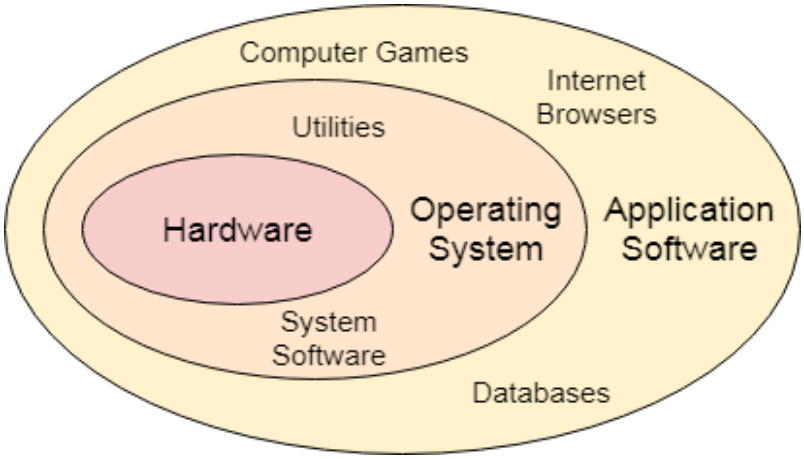
(b) Ensure that different programs and users running at the same time do not interfere with each other.

(c) Provide a software platform on top of which other programs can run.

1.2 Operating System Definition

In the Computer System (comprises of Hardware and software), Hardware can only understand machine code (in the form of 0 and 1) which doesn't make any sense to a naive user. We need a system which can act as an intermediary and manage all the processes and resources present in the system.

Operating System is system software, which may be viewed as collection of software consisting of procedures for operating the computer & providing an environment for execution of programs. It's an interface between user & computer. So an OS makes everything in the computer to work together smoothly & efficiently.



An **Operating System** can be defined as an **interface between user and hardware**. It is responsible for the execution of all the processes, Resource Allocation, CPU management, File Management and many other tasks.

The purpose of an operating system is to provide an environment in which a user can execute programs in convenient and efficient manner.

1.3 Functions of Operation System

An operating system is a program that acts as a user-computer GUI (Graphical user interface). It controls the execution of all types of applications.

The operating system performs the following functions in a device.

1. **Instruction:** The operating system establishes a mutual understanding between the various instructions given by the user.

2. **Input/output Management:** What output will come from the input given by the user, the operating system runs this program. This management involves coordinating various input and output devices. It assigns the functions of those devices where one or more applications are executed.

3. **Memory Management:** The operating system handles the responsibility of storing any data, system programs, and user programs in memory. This function of the operating system is called memory management.

4. **File Management:** The operating system is helpful in making changes in the stored files and in replacing them. It also plays an important role in transferring various files to a device.

5. **Processor Management:** The processor is the execution of a program that accomplishes the specified work in that program. It can be defined as an execution unit where a program runs.

6. **Job Priority:** The work of job priority is creation and promotion. It determines what action should be done first in a computer system.

7. **Special Control Program:** The operating systems make automatic changes to the task through specific control programs. These programs are called Special Control Program.

8. **Scheduling of resources and jobs:** The operating system prepares the list of tasks to be performed for the device of the computer system. The operating system decides which device to use for which

task. This action becomes complicated when multiple tasks are to be performed simultaneously in a computer system. The scheduling programs of the operating system determine the order in which tasks are completed. It performs these tasks based on the priority of performing the tasks given by the user. It makes the tasks available based on the priority of the device.

9. **Security:** Computer security is a very important aspect of any operating system. The reliability of an operating system is determined by how much better security it provides us. Modern operating systems use a firewall for security. A firewall is a security system that monitors every activity happening in the computer and blocks that activity in case of any threat.

10. **Monitoring activities:** The operating system takes care of the activities of the computer system during various processes. This aborts the program if there are errors. The operating system sends instant messages to the user for any unexpected error in the input/output device. It also provides security to the system when the operating system is used in systems operated by multiple users. So that illegal users cannot get data from the system.

11. **Job accounting:** It keeps track of time & resources used by various jobs and users.

1.4 Operating System Characteristics

The Operating systems are different according to the three primary characteristics which are licensing, software compatibility, and complexity.

- **Licensing**

There are basically three kinds of Operating systems. One is Open Source OS, another is Free OS and the third is Commercial OS.

Linux is an **open source** operating system which means that anyone can download and modify it for example Ubuntu etc. A **free OS** doesn't have to be open source. They are free to download and use but cannot modify them. For example, Google owns Chrome OS and makes it free to use.

Commercial operating systems are privately owned by companies that charge money for them. Examples include Microsoft Windows and Apple macOS. These require paying for the right (or license) to use their Operating systems.

- **Software Compatibility**

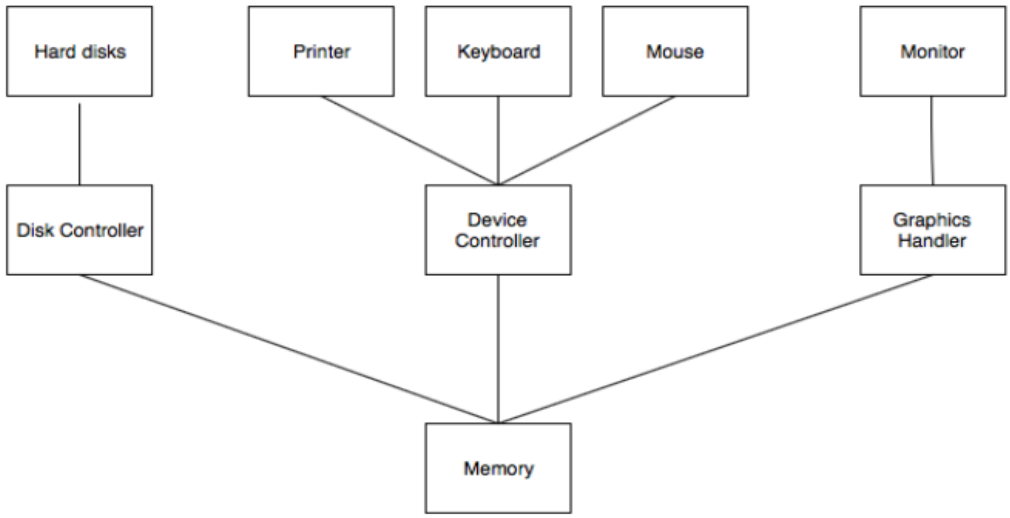
The developers make the software's which may be compatible or incompatible in different versions within the same operating system's type but they can't be compatible with the other OS types. Every OS type have their own software compatibility.

- **Complexity**

Operating systems come in basically two editions one is 32-bit and other is 64-bit editions. The 64-bit edition of an operating system best utilizes random access memory (RAM). A computer with a 64-bit CPU can run either a 32-bit or a 64-bit OS, but a computer with a 32-bit CPU can run only a 32-bit OS.

1.5 Computer System Organization

The computer system is a combination of many parts such as peripheral devices, secondary memory, CPU etc. This can be explained more clearly using a diagram.



The salient points about the above figure displaying Computer System Organization is −

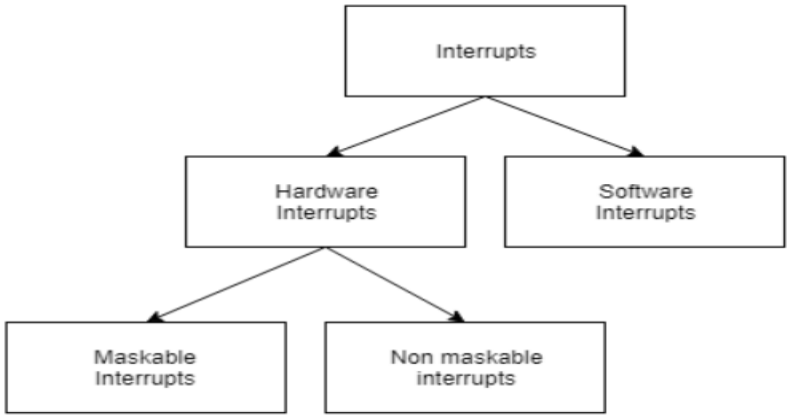
- The I/O devices and the CPU both execute concurrently. Some of the processes are scheduled for the CPU and at the same time, some are undergoing input/output operations.
- There are multiple device controllers, each in charge of a particular device such as keyboard, mouse, printer etc.
- There is buffer available for each of the devices. The input and output data can be stored in these buffers.

- The data is moved from memory to the respective device buffers by the CPU for I/O operations and then this data is moved back from the buffers to memory.
- The device controllers use an interrupt to inform the CPU that I/O operation is completed.

1.5.1 Interrupt Handling

An interrupt is a necessary part of Computer System Organization as it is triggered by hardware and software parts when they need immediate attention.

An interrupt can be generated by a device or a program to inform the operating system to halt its current activities and focus on something else. The types of interrupts are better explained using the following diagram −

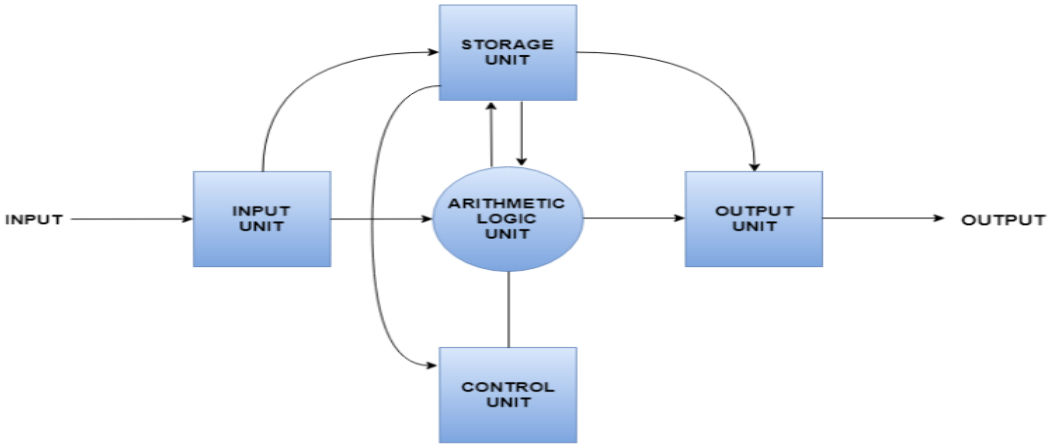


Hardware and software interrupts are two types of interrupts. Hardware interrupts are triggered by hardware peripherals while software interrupts are triggered by software function calls. Hardware interrupts are of further two types. Maskable interrupts can be ignored or disabled by the CPU while this is not possible for non mask able interrupts.

1.6 Computer System Architecture

A computer system is basically a machine that simplifies complicated tasks. It should maximize performance and reduce costs as well as power consumption. The different components in the Computer System Architecture are Input Unit, Output Unit, Storage Unit, Arithmetic Logic Unit, Control Unit etc.

A diagram that shows the flow of data between these units is as follows –



The input data travels from input unit to ALU. Similarly, the computed data travels from ALU to output unit. The data constantly moves from storage unit to ALU and back again. This is because stored data is computed on before being stored again. The control unit controls all the other units as well as their data.

Details about all the computer units are −

- **Input Unit**

  The input unit provides data to the computer system from the outside. So, basically it links the external environment with the computer. It takes data from the input devices, converts it into machine language and then loads it into the computer system. Keyboard, mouse etc. are the most commonly used input devices.

- **Output Unit**

  The output unit provides the results of computer process to the users i.e it links the computer with the external environment. Most of the output data is the form of audio or video. The different output devices are monitors, printers, speakers, headphones etc.

- **Storage Unit**

  Storage unit contains many computer components that are used to store data. It is traditionally divided into primary storage and secondary storage. Primary storage is also known as the main memory and is the memory directly accessible by the CPU. Secondary or external storage is not directly accessible by the CPU. The data from secondary storage needs to be brought into the

primary storage before the CPU can use it. Secondary storage contains a large amount of data permanently.

- **Arithmetic Logic Unit**

  All the calculations related to the computer system are performed by the arithmetic logic unit. It can perform operations like addition, subtraction, multiplication, division etc. The control unit transfers data from storage unit to arithmetic logic unit when calculations need to be performed. The arithmetic logic unit and the control unit together form the central processing unit.

- **Control Unit**

  This unit controls all the other units of the computer system and so is known as its central nervous system. It transfers data throughout the computer as required including from storage unit to central processing unit and vice versa. The control unit also dictates how the memory, input output devices, arithmetic logic unit etc. should behave.

1.7 Operating System Services

The design of a new operating system is a major task. It is important that the goals of the system be well defined before the design begins. These goals form the basis for choices among various algorithms and strategies. One view focuses on the services that the system provides; another, on the interface that it makes available to users and programmers; a third, on its components and their interconnections. Here, we explore all three aspects of operating systems, showing the viewpoints of users, programmers, and operating-system designers.

1.7.1 Operating System Services

An Operating System supplies different kinds of services to both the users and to the programs as well. It also provides application programs (that run within an Operating system) an environment to execute it freely. It provides users the services run various programs in a convenient manner. These operating-system services are provided for the convenience of the programmer, to make the programming task easier. Figure shows one view of the various operating-system services and how they interrelate.

1.7.2 User Interface

Usually Operating system comes in three forms or types. Depending on the interface their types have been further subdivided. These are:

- Command line interface
- Batch based interface
- Graphical User Interface

Let's get to know in brief about each of them.

The command line interface (CLI) usually deals with using text commands and a technique for entering those commands. The batch interface (BI): commands and directives are used to manage those commands that are entered into files and those files get executed. Another type is the graphical user interface (GUI): which is a window system with a pointing device (like mouse or trackball) to point to the I/O, choose from menus driven interface and to make choices viewing from a number of lists and a keyboard to entry the texts.

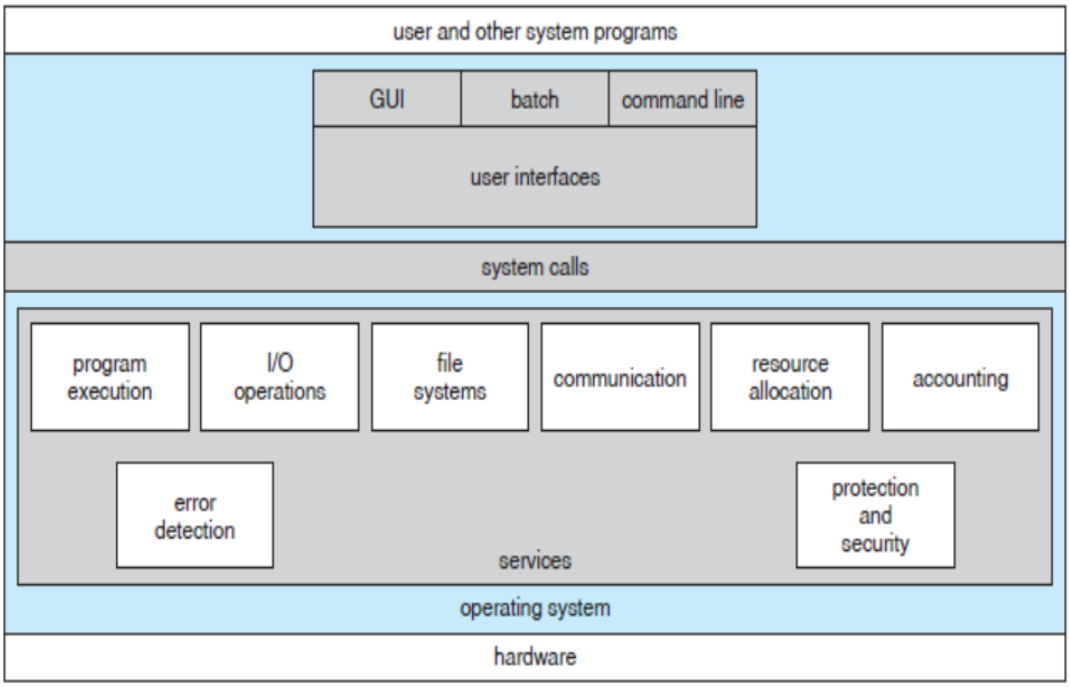


Figure: A view of operating system services

1.7.3 Program execution

Operating system handles many kinds of activities from user programs to system programs like printer spooler, name servers, file server etc. Each of these activities is encapsulated as a process. A process

includes the complete execution context (code to execute, data to manipulate, registers, OS resources in use).Following are the major activities of an operating system with respect to program management.

- Loads a program into memory.
- Executes the program.
- Handles program's execution.
- Provides a mechanism for process synchronization.
- Provides a mechanism for process communication.
- Provides a mechanism for deadlock handling.

1.7.4 I/O Operation

I/O subsystem comprised of I/O devices and their corresponding driver software. Drivers hides the peculiarities of specific hardware devices from the user as the device driver knows the peculiarities of the specific device. Operating System manages the communication between user and device drivers. Following are the major activities of an

- operating system with respect to I/O Operation.
- I/O operation means read or write operation with any file or any specific I/O device.
- Program may require any I/O device while running.
- Operating system provides the access to the required I/O device when required.

1.7.5 File system manipulation

A file represents a collection of related information. Computer can store files on the disk (secondary storage), for long term storage purpose. Few examples of storage media are magnetic tape, magnetic disk and optical disk drives like CD, DVD. Each of these media has its own properties like speed, capacity, data transfer rate and data access methods.

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions. Following are the major activities of an operating system with respect to file management.

- Program needs to read a file or write a file.
- The operating system gives the permission to the program for operation on file.
- Permission varies from read-only, read-write, denied and so on.

- Operating System provides an interface to the user to create/delete files.
- Operating System provides an interface to the user to create/delete directories.
- Operating System provides an interface to create the backup of file system.

1.7.6 Communication

In case of distributed systems which are a collection of processors that do not share memory, peripheral devices, or a clock, operating system manages communications between processes. Multiple processes with one another through communication lines in the network. OS handles routing and connection strategies, and the problems of contention and security. Following are the major

- activities of an operating system with respect to communication.
- Two processes often require data to be transferred between them.
- The both processes can be on the one computer or on different computer but are connected through computer network.
- Communication may be implemented by two methods either by Shared Memory or by Message Passing.

1.7.7 Error handling

Error can occur anytime and anywhere. Error may occur in CPU, in I/O devices or in the memory hardware. Following are the major activities of an operating system with respect to error handling.

- OS constantly remains aware of possible errors.
- OS takes the appropriate action to ensure correct and consistent computing.

1.7.8 Resource Management

In case of multi-user or multi-tasking environment, resources such as main memory, CPU cycles and files storage are to be allocated to each user or job. Following are the major activities of an operating system with respect to resource management.

- OS manages all kind of resources using schedulers.
- CPU scheduling algorithms are used for better utilization of CPU.

1.7.9 Protection

Considering computer systems having multiple users the concurrent execution of multiple processes, then the various processes must be protected from each another's activities.

Protection refers to mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system. Following are the major activities of an operating system with respect to protection.

- OS ensures that all access to system resources is controlled.
- OS ensures that external I/O devices are protected from invalid access attempts.
- OS provides authentication feature for each user by means of a password.

1.7.10 System programs

System programs provide an environment where programs can be developed and executed. In the simplest sense, system programs also provide a bridge between the user interface and system calls. In reality, they are much more complex. For example, a compiler is a complex system program.

The system program serves as a part of the operating system. It traditionally lies between the user interface and the system calls. The user view of the system is actually defined by system programs and not system calls because that is what they interact with and system programs are closer to the user interface. The user view the operating system observed is actually the system programs and not the system calls. System programs can be divided into seven parts. These are given as follows

- **Status Information**: The status information system programs provide required data on the current or past status of the system. This may include the system date, system time, and available memory in system, disk space, logged in users etc.
- **Communications**: These system programs are needed for system communications such as web browsers. Web browsers allow systems to communicate and access information from the network as required
- **File Modification:** System programs that are used for file modification basically change the data in the file or modify it in some other way. Text editors are a big example of file modification system programs.
- **File Manipulation:** These system programs are used to manipulate system files. This can be done using various commands like create, delete, copy, rename, print etc. These commands can create files, delete files, copy the contents of one file into another, rename files, print them etc.

- **Application Programs:** Application programs can perform a wide range of services as per the needs of the users. These include programs for database systems, word processors, plotting tools, spreadsheets, games, scientific applications etc.
- **Programming Language Support:** These system programs provide additional support features for different programming languages. Some examples of these are compilers, debuggers etc. These compile a program and make sure it is error free respectively.

1.8 Operating System Structure

For efficient performance and implementation an OS should be partitioned into separate subsystems, each with carefully defined tasks, inputs, outputs, and performance characteristics. These subsystems can then be arranged in various architectural configurations

1.8.1 Simple Structure: There are several commercial system that don't have a well- defined structure such operating systems begins as small, simple & limited systems and then grow beyond their original scope. MS-DOS is an example of such system. It was not divided into modules carefully. Another example of limited structuring is the UNIX operating system.

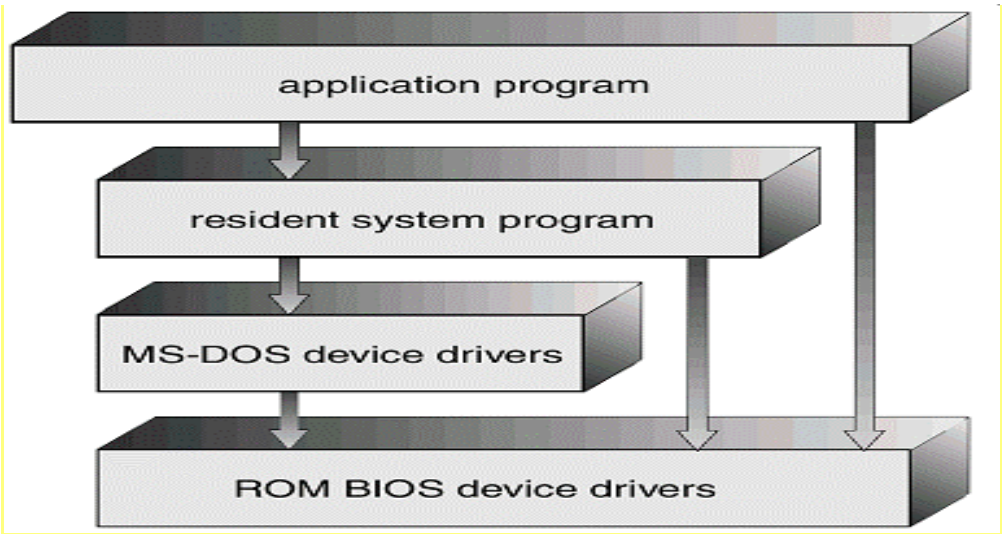


Figure: Layered approach

1.8.2 Layered Approach: Another approach is to break the OS into a number of smaller layers, each of which rests on the layer below it, and relies solely on the services provided by the next lower layer. This approach allows each layer to be developed and debugged independently, with the assumption that all lower layers have already been debugged and are trusted to deliver proper services. The problem is deciding what order in which to place the layers, as no layer can call upon the services of any higher layer, and so many chicken-and-egg situations may arise. Layered approaches can also be less efficient, as a request for service from a higher layer has to filter through all lower layers before it reaches the HW, possibly with significant processing at each step.

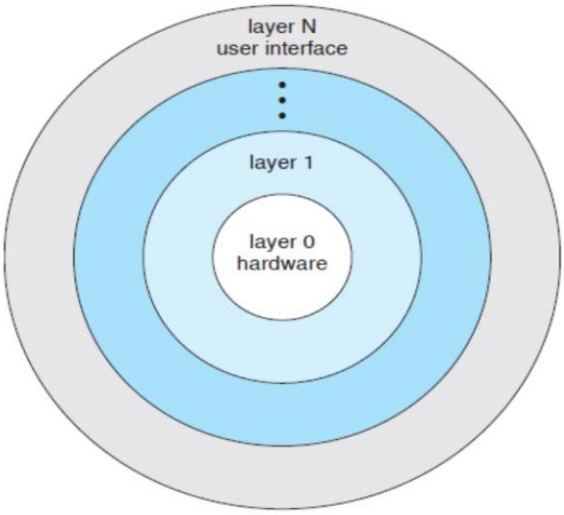


Figure 3: Layered operating system

1.8.3 Micro kernels: The basic idea behind micro kernels is to remove all non-essential services from the kernel, and implement them as system applications instead, thereby making the kernel as small and efficient as possible.

Most microkernels provide basic process and memory management, and message passing between other services, and not much more. Security and protection can be enhanced, as most services are performed in user mode, not kernel mode.

System expansion can also be easier, because it only involves adding more system applications, not rebuilding a new kernel. Mach was the first and most widely known microkernel, and now forms a major component of Mac OSX.

Windows NT was originally microkernel, but suffered from performance problems relative to Windows 95. NT 4.0 improved performance by moving more services into the kernel, and now XP is back to being more monolithic. Another microkernel example is QNX, a real-time OS for embedded systems.

1.8.4 Modules: Modules are similar to layers in that each subsystem has clearly defined tasks and interfaces, but any module is free to contact any other module, eliminating the problems of going through multiple intermediary layers, as well as the chicken-and-egg problems. The kernel is relatively small in this architecture, similar to microkernels, but the kernel does not have to implement message passing since modules are free to contact each other directly.

For example, the Solaris operating system structure is organized around a core kernel with seven. Types of loadable kernel modules:

- Scheduling classes
- File systems
- Loadable system calls
- Executable formats
- STREAMS modules
- Miscellaneous
- Device and bus drivers

1.9 Virtual Machines

Virtual Machine abstracts the hardware of our personal computer such as CPU, disk drives, memory, NIC (Network Interface Card) etc, into many different execution environments as per our requirements, hence giving us a feel that each execution environment is a single computer. For example, Virtual Box.

When we run different processes on an operating system, it creates an illusion that each process is running on a different processor having its own virtual memory, with the help of CPU scheduling and virtual-memory techniques. There are additional features of a process that cannot be provided by the hardware alone like system calls and a file system. The virtual machine approach does not provide these additional functionalities but it only provides an interface that is same as basic hardware. Each process is provided with a virtual copy of the underlying computer system.

We can create a virtual machine for several reasons, all of which are fundamentally related to the ability to share the same basic hardware yet can also support different execution environments, i.e., different operating systems simultaneously.

The main drawback with the virtual-machine approach involves disk systems. Let us suppose that the physical machine has only three disk drives but wants to support seven virtual machines. Obviously, it cannot allocate a disk drive to each virtual machine, because virtual-machine software itself will need substantial disk space to provide virtual memory and spooling. The solution is to provide virtual disks. Users are thus given their own virtual machines. After which they can run any of the operating systems or software packages that are available on the underlying machine.

The virtual-machine software is concerned with multi-programming multiple virtual machines onto a physical machine, but it does not need to consider any user-support software. This arrangement can provide a useful way to divide the problem of designing a multi-user interactive system, into two smaller pieces.

Advantages:

1. There are no protection problems because each virtual machine is completely isolated from all other virtual machines.
2. Virtual machine can provide an instruction set architecture that differs from real computers.
3. Easy maintenance, availability and convenient recovery.

Disadvantages:

1. When multiple virtual machines are simultaneously running on a host computer, one virtual machine can be affected by other running virtual machines, depending on the workload.
2. Virtual machines are not as efficient as a real one when accessing the hardware.

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of the main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program-generated addresses are translated automatically to the corresponding machine addresses.

The size of virtual storage is limited by the addressing scheme of the computer system and the amount of secondary memory is available not by the actual number of the main storage locations.

It is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.

1. All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of the main memory such that it occupies different places in the main memory at different times during the course of execution.
2. A process may be broken into a number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time address translation and use of page or segment table permits this.

If these characteristics are present then, it is not necessary that all the pages or segments are present in the main memory during execution. This means that the required pages need to be loaded into memory whenever required. Virtual memory is implemented using Demand Paging or Demand Segmentation.

1.10 Protection and Security

Protection and security requires that computer resources such as CPU, software's, memory etc. are protected. This extends to the operating system as well as the data in the system.

This can be done by ensuring integrity, confidentiality and availability in the operating system. The system must be protecting against unauthorized access, viruses, worms etc.

1.10.1 Threats to Protection and Security

A threat is a program that is malicious in nature and leads to harmful effects for the system. Some of the common threats that occur in a system are −

Virus

Viruses are generally small snippets of code embedded in a system. They are very dangerous and can corrupt files, destroy data, crash systems etc. They can also spread further by replicating themselves as required.

Trojan horse

A trojan horse can secretly access the login details of a system. Then a malicious user can use these to enter the system as a harmless being and wreak havoc.

Trap Door

A trap door is a security breach that may be present in a system without the knowledge of the users. It can be exploited to harm the data or files in a system by malicious people.

Worm

A worm can destroy a system by using its resources to extreme levels. It can generate multiple copies which claim all the resources and don't allow any other processes to access them. A worm can shut down a whole network in this way.

Denial of Service

These types of attacks do not allow the legitimate users to access a system. It overwhelms the system with requests so it is overwhelmed and cannot work properly for other user.

1.10.2 Protection and Security Methods

The different methods that may provide protect and securities for different computer systems are:

Authentication

This deals with identifying each user in the system and making sure they are who they claim to be. The operating system makes sure that all the users are authenticated before they access the system. The different ways to make sure that the users are authentic are:

- **Username/ Password**

  Each user has a distinct username and password combination and they need to enter it correctly before they can access the system.

- **User Key/ User Card**

  The users need to punch a card into the card slot or use they individual key on a keypad to access the system.

- **User Attribute Identification**

Different user attribute identifications that can be used are fingerprint, eye retina etc. These are unique for each user and are compared with the existing samples in the database. The user can only access the system if there is a match.

One Time Password

These passwords provide a lot of security for authentication purposes. A onetime password can be generated exclusively for a login every time a user wants to enter the system. It cannot be used more than once. The various ways a onetime password can be implemented are −

- **Random Numbers**

  The system can ask for numbers that correspond to alphabets that are pre arranged. This combination can be changed each time a login is required.

- **Secret Key**

  A hardware device can create a secret key related to the user id for login. This key can change each time

1.11 Check your progress:

1. What is operating system?

   A. collection of programs that manages hardware resources

   B. system service provider to the application programs

   C. link to interface the hardware and application programs

   D. all of the mentioned

2. To access the services of operating system, the interface is provided by the:

   A. system calls

   B. API

   C. library

   D. assembly instructions

3. Which one of the following is not true?

   A. kernel is the program that constitutes the central core of the operating system

B. kernel is the first part of operating system to load into memory during booting

C. kernel is made of various modules which cannot be loaded in running operating system

D. kernel remains in the memory during the entire computer session

4. Which one of the following error will be handling by the operating system?

A. power failure

B. lack of paper in printer

C. connection failure in the network

D. all of the mentioned

5. The main function of the command interpreter is:

A. to get and execute the next user-specified command

B. to provide the interface between the API and application program

C. to handle the files in operating system

D. none of the mentioned

6. By operating system, the resource management can be done via:

A. time division multiplexing

B. space division multiplexing

C. both (a) and (b)

D. none of the mentioned

1.12 Summary

The prime responsibility of operating system is to manage the resources of the computer system. In addition to these, Operating System provides an interface between the user and the bare machine. On the basis of their attributes and design objectives, different types of operating systems were defined and characterized with respect to scheduling and management of memory, devices, and files. The primary concerns of a time-sharing system are equitable sharing of resources and responsiveness to interactive requests. Real-time operating systems are mostly concerned with responsive handling of external events generated by the controlled system.

Distributed operating systems provide facilities for global naming and accessing of resources, for resource migration, and for distribution of computation. Process scheduling is a very important function of an operating system. Three different schedulers may coexist and interact in a complex operating system: long-term scheduler, medium-term scheduler, and short-term scheduler. Of the presented scheduling disciplines, FCFS scheduling is the easiest to implement but is a poor performer. SRTN scheduling is optimal but unrealizable. RR scheduling is most popular in time- sharing environments, and event-driven and earliest-deadline scheduling are dominant in real-time and other systems with time-critical requirements. Multiple- level queue scheduling, and its adaptive variant with feedback, is the most general scheduling discipline suitable for complex environments that serve amixture of processes with different characteristics.

1.13 Keywords

SPOOL: Simultaneous Peripheral Operations on Line

Task: An instance of a program in execution is called a process or a task.

Multitasking: The ability to execute more than one task at the same time is called as multitasking.

Real time: These systems are characterized by very quick processing of databecause the output influences immediate decisions.

Multiprogramming: It is characterized by many programs simultaneously resident in memory, and execution switches between programs.

Long-term scheduling: the decisions to introduce new processes for execution,or re-execution.

Medium-term scheduling: the decision to add to (grow) the processes that are fully or partially in memory.

Short-term scheduling: It decides which (Ready) process to execute next.

Non-preemptive scheduling: In it, process will continue to execute until itterminates, or makes an I/O request which would block the process, or makes anoperating system call.

Preemptive scheduling: In it, the process may be pre-empted by the operating system when a

new process arrives (perhaps at a higher priority), or an interrupt or signal occurs, or a (frequent) clock interrupt occurs.

1.14 Self-Assessment Questions (SAQ)

1. What are the objectives of an operating system? Discuss.
2. Differentiate between multiprogramming, multitasking, and multiprocessing.
3. What are the major functions performed by an operating system? Explain.
4. Discuss various process scheduling policies with their cons and pros.
5. Define process. What are the different states of a process? Explain using aprocess state transition diagram.
6. What are the objectives of a good scheduling algorithm?
7. Explain the term context switch; how does the cost of a context switch affect the choice of scheduling algorithm.
8. From the point of view of scheduling, briefly explain the different requirementsimposed by the following types of system: (a) batch, (b) interactive, (c) real- time.
9. Explain the need to compromise quantum used in round-robin.
10. Why round-robin scheduling is not suitable in batch operated computer system?

1.15 Answer to check your progress

1. all of the mentioned
2. system calls
3. kernel is made of various modules which cannot be loaded in running operating system
4. all of the mentioned
5. to get and execute the next user-specified command
6. both (a) and (b)

1.16 Reference/ Suggested Readings

1. Operating System Concepts, 5th Edition, Silberschatz A., Galvin P.B.,John Wiley and Sons.

2. Systems Programming and Operating Systems, 2nd Revised Edition,Dhamdhere D.M., Tata McGraw Hill Publishing Company Ltd., New Delhi.

3. Operating Systems, Madnick S.E., Donovan J.T., Tata McGraw HillPublishing Company Ltd., New Delhi.

4. Operating Systems-A Modern Perspective, Gary Nutt, Pearson EducationAsia, 2000.

5. Operating Systems, Harris J.A., Tata McGraw Hill Publishing CompanyLtd., New Delhi, 2002.

6. https://www.javatpoint.com

7. https://www.tutorialspoint.com

Course: MCA-32	Writer: Ritu
Lesson: 2	Vetter:

System Call and Types of Operating System

Structure

2.4 Multiprocessing Operating System

2.4.1 Types of multiprocessing systems

2.4.1.1 Symmetrical multiprocessing operating system

2.4.1.2 Asymmetric multiprocessing operating system

2.5 Network Operating System

2.5.1 Types of Network Operating System

2.5.1.1 Peer to Peer System

2.5.1.2 Client-Server System

2.6 Real Time Operating System (RTOS)

2.6.1 Applications of Real-time operating system (RTOS)

2.6.2 Types of Real-time operating system

2.6.2.1 **Hard Real-Time operating system**

2.6.2.2 Soft Real-Time operating system

2.6.2.3 Firm Real-Time operating system

2.7 Mobile Operating System

2.7.1 Popular platforms of the Mobile OS

2.8 Cloud Operating System

2.8.1 Advantages and disadvantages of Cloud Operating System

2.8.2 Top Cloud Operating Systems

2.8.2.1 Netvibes

2.8.2.2 CloudMe

2.8.2.3 Amoeba OS

2.8.2.4 EyeOS

2.8.2.5 Ghost OS

2.8.2.6 OSv

2.0 Learning Objectives

The users of this lesson have already gone through some concepts of operating system in the first year of the course. So the objectives of this lesson are:

(a) To review the basic concepts of operating system i.e. definition, types, and functions performed by them.

(b) To review the process scheduling policies.

2. 1 Introduction

Operating System may be viewed as collection of software consisting of procedures for operating the computer and providing an environment for execution of programs. The main goals of the Operating System are:

(iii)To make the computer system convenient to use,

(iv)To make the use of computer hardware in efficient way.

Basically, an Operating System has three main responsibilities:

(d) Perform basic tasks such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices such as disk drives and printers.

(e) Ensure that different programs and users running at the same time do not interfere with each other.

(f) Provide a software platform on top of which other programs can run.

2.2 System call

A system call is programming interface for an application to request service from the operating system. A system call is a **method** used by application programs **to communicate with the system core**. In modern operating systems, this method is used if a user application or process needs to pass information onto the hardware, other processes or the kernel itself, or if it needs to read information from these sources. This makes these calls a **link between user mode and kernel mode,** the two key access and security modes for processing CPU commands in computer systems. As soon as the action required by a system call is carried out, the kernel gives control back again, and the program code is continued from the point it had reached before the system call was started. The need for system calls is closely tied to the modern operating system model with user mode and kernel mode, which was implemented as a response to the rising number of processes being carried out simultaneously in computers' **main memory**

The more privileged kernel mode is the pivotal control system here because – as mentioned already – not only are all services and processes in the system itself run there, but also **system-critical actions by application programs** that are blocked in user mode. One requirement is the right system call through the respective program, which in most cases is simply for access to processing power or memory structures. If an application needs more **computing power or storage space**, for example, or an **application-external file** is required, system calls are essential. After the entire file is copied, the program may close both files, write a message to the console or window, and finally terminate normally. As we simple programs may make heavy use of the operating system. Frequently, systems execute thousands of system calls per second. This system call sequence is shown in Figure

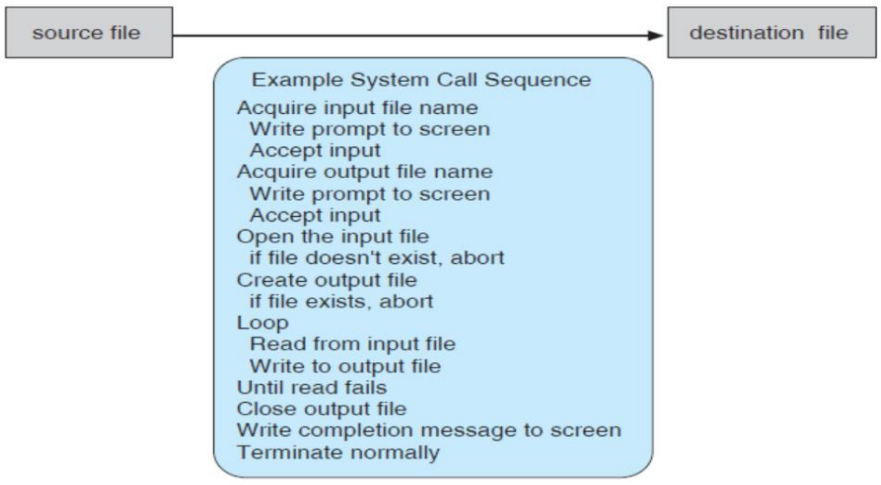


Figure 4: Example of how system calls are used

2.2.1 Types of System Calls

The system call provides an interface to the operating system services. Application developers often do not have direct access to the system calls, but can access them through an application programming interface (API). The functions that are included in the API invoke the actual system calls. By using the API, certain benefits can be gained: Portability: as long a system supports an API, any program using that API can compile and run. Ease of Use: using the API can be significantly easier than using the actual system call.

Three general methods exist for passing parameters to the OS: Parameters can be passed in registers. When there are more parameters than registers, parameters can be stored in a block and the block address can be passed as a parameter to a register. Parameters can also be pushed on or popped off the stack by the operating system. There are 5 different categories of system calls: process control, file manipulation, device manipulation, information maintenance, and communication.

2.2.2 Process Control

A process or job executing one program may want to load and execute another program. This feature allows the command interpreter to execute a program as directed by, for example, a user command, the click of a mouse, or a batch command. An interesting question is where to return control when the loaded program terminates. This question is related to the problem of whether the existing program is lost, saved, or allowed to continue execution concurrently with the new program. A running program needs to be able to stop execution either normally or abnormally. When execution is stopped abnormally, often a dump of memory is taken and can be examined with a debugger.

- **End, abort:** A running program needs to be able to has its execution either normally (end) or abnormally (abort).
- **Load, execute:** A process or job executing one program may want to load and executes another program.
- **Create Process, terminate process:** There is a system call specifying for the purpose of creating a new process or job (create process or submit job). We may want to terminate a job or process that we created (terminates process, if we find that it is incorrect or no longer needed).

- **Get process attributes, set process attributes:** If we create a new job or process we should able to control its execution. This control requires the ability to determine & reset the attributes of a job or processes (get process attributes, set process attributes).
- **Wait time:** After creating new jobs or processes, we may need to wait for them to finish their execution (wait time).
- **Wait event, signal event:** We may wait for a specific event to occur (wait event). The jobs or processes then signal when that event has occurred (signal event).

2.2.3 File Management

We first need to be able to create and delete files. Either system call requires the name of the file and perhaps some of the file's attributes. Once the file is created, we need to open it and to use it. We may also read, write, or reposition (rewinding or skipping to the end of the file, for example). Finally, we need to close the file, indicating that we are no longer using it. There is a need to determine the file attributes – *get* and *set* file attribute. Many times the OS provides an API to make these system calls. These are as follow

- **Create file, delete file:** We first need to be able to create & delete files. Both the system calls require the name of the file & some of its attributes.
- **Open file, close file:** Once the file is created, we need to open it & use it. We close the file when we are no longer using it.
- **Read, write, reposition file:** After opening, we may also read, write or reposition the file (rewind or skip to the end of the file).
- **Get file attributes, set file attributes:** For either files or directories, we need to be able to determine the values of various attributes & reset them if necessary. Two system calls get file attribute & set file attributes are required for their purpose.

2.2.4 Device Management

A process may need several resources to execute—main memory, disk drives, access to files, and so on. These resources are also thought of as devices. Some are physical, such as a video card, and others are abstract, such as a file. User programs *request* the device, and when finished they *release* the device. Once the device has been requested and allocated to us, we can read, write, and (possibly) reposition the device, just as we can with files.

- **Request device, release device:** If there are multiple users of the system, we first request the device. After we finished with the device, we must release it.
- **Read, write, reposition:** Once the device has been requested & allocated to us, we can read, write & reposition the device.

2.2.5 Information Maintenance

The operating system keeps information about all its processes, and system calls are used to access this information. Generally, calls are also used to reset the process information (get process attributes and set process attributes). Some system calls exist purely for transferring information between the user program and the operating system. An example of this is *time,* or *date*. The OS also keeps information about all its processes and provides system calls to report this information.

- **Get time or date, set time or date:** Most systems have a system call to return the current date & time or set the current date & time.
- **Get system data, set system data:** Other system calls may return information about the system like number of current users, version number of OS, amount of free memory etc.
- **Get process attributes, set process attributes:** The OS keeps information about all its processes & there are system calls to access this information.

2.2.6 Communication

There are two modes of communication such as:

- **Message passing model:** Information is exchanged through an inter process

  Communication facility provided by operating system. Each computer in a network has a name by which it is known. Similarly, each process has a process name which is translated to an equivalent identifier by which the OS can refer to it. The get host id and get processed systems calls to do this translation. These identifiers are then passed to the general purpose open & close calls provided by the file system or to specific open connection system call. The recipient process must give its permission for communication to take place with an accept connection call. The source of the communication known as client & receiver known as server exchange messages by read message & write message system calls. The close connection call terminates the connection.

- **Shared memory model:** processes use map memory system calls to access regions of memory owned by other processes. They exchange information by reading & writing data in the shared areas. The processes ensure that they are not writing to the same location simultaneously.

2.2.7 Protection

Protection provides a mechanism for controlling access to the resources provided by a computer system. Historically, protection was a concern only on multi programmed computer systems with several users. However, with the advent of networking and the Internet, all computer systems, from servers to PDAs, must be concerned with protection. Typically, system calls providing protection include

- **Set permission and get permission**, which manipulate the permission settings of resources such as files and disks.
- **Allow user and deny user** system calls specify whether particular users can—or cannot—be allowed access to certain resources.

2.3 TYPES OF OPERATING SYSTEMS

Operating System can be classified into various categories on the basis of several criteria, viz. number of simultaneously active programs, number of users working simultaneously, number of processors in the computer system, etc.

2.3.1 Batch Operating System

Batch processing requires the program, data, and appropriate system commands to be submitted together in the form of a job. Batch operating systems usually allow little interaction between users and executing programs. Batch processing has a greater potential for resource utilization than simple serial processing in computer systems serving multiple users. Due to turnaround delays and offline debugging, batch is not very convenient for program development. Programs that do not require interaction and need long execution times may be served well by a batch operating system. Memory management and scheduling in batch is very simple. Jobs are typically processed in first-come first-served fashion. Memory is usually divided into two areas. The resident portion of the Operating System permanently occupies one of them, and the other is used to load transient programs for execution. When a transient program terminates, a new program is loaded into the same area of memory. Since at most one program is in execution at

any time, batch systems do not require any time-critical device management. Batch systems often provide simple forms of file management because of serial access to files. In it little protection and no concurrency control of file access is required.

2.3.2 Multiprogramming Operating System

A multiprogramming system permits multiple programs to be loaded into memory and execute the programs concurrently. Concurrent execution of programs results into improved system throughput and resource utilization. This potential is realized by a class of operating systems that multiplex resources of a computer system among a multitude of active programs. Such operating systems usually have the prefix multi in their names, such as multitasking or multiprogramming.

2.3.3 Multitasking Operating System

A multitasking Operating System is distinguished by its ability to support concurrent execution of two or more active processes. An instance of a program in execution is called a process or a task. Multitasking is usually implemented by maintaining code and data of several processes in memory simultaneously, and by multiplexing processor and I/O devices among them. Multitasking is often coupled with hardware and software support for memory protection in order to prevent erroneous processes from corrupting address spaces and behavior of other resident processes. The terms multitasking and multiprocessing are often used interchangeably, although multiprocessing sometimes implies that more than one CPU is involved. In multitasking, only one CPU is involved, but it switches from one program to another so quickly that it gives the appearance of executing all of the programs at the same time. There are two basic types of multitasking: preemptive and cooperative. In preemptive multitasking, the Operating System parcels out CPU time slices to each program. In cooperative multitasking, each program can control the CPU for as long as it needs it. If a program is not using the CPU, however, it can allow another program to use it temporarily.

2.3.4 Multi-user Operating System

Multiprogramming operating systems usually support multiple users, in which case they are also called multi-user systems. Multi-user operating systems provide facilities for maintenance of individual user environments and therefore require user accounting. In general,

multiprogramming implies multitasking, but multitasking does not imply multi-programming. In effect, multitasking operation is one of the mechanisms that a multiprogramming Operating System employs in managing the totality of computer-system resources, including processor, memory, and I/O devices. Multitasking operation without multi-user support can be found in operating systems of some advanced personal computers and in real-time systems. Multi-access operating systems allow simultaneous access to a computer system through two or more terminals. In general, multi-access operation does not necessarily imply multiprogramming. An example is provided by some dedicated transaction-processing systems, such as airline ticket reservation systems, that support hundreds of active terminals under control of a single program.

In general, the multiprocessing or multiprocessor operating systems manage the operation of computer systems that incorporate multiple processors. Multiprocessor operating systems are multitasking operating systems by definition because they support simultaneous execution of multiple tasks (processes) on different processors. Depending on implementation, multitasking may or may not be allowed on individual processors. Except for management and scheduling of multiple processors, multiprocessor operating systems provide the usual complement of other system services that may qualify them as time- sharing, real-time, or a combination operating system.

2.3.5 Multithreading

Multithreading allows different parts of a single program to run concurrently. The programmer must carefully design the program in such a way that all the threads can run at the same time without interfering with each other.

2.3.6 Time-sharing system

Time-sharing is a popular representative of multi-programmed, multi-user systems. One of the primary objectives of multi-user and time-sharing is good terminal response time. Time-sharing systems often attempt to provide equitable sharing of common resources, giving the illusion to each user of having a machine to oneself. Most time-sharing systems use time-slicing scheduling, in which programs are executed with rotating priority that increases during waiting and drops after the service is granted. In order to prevent programs from monopolizing the

processor, a program executing longer than the system-defined time slice is interrupted by the Operating System and placed at the end of the queue of waiting programs. Memory management in time-sharing systems provides for isolation and protection of co-resident programs. Some forms of controlled sharing are sometimes provided to conserve memory and to exchange data between programs. Being executed on behalf of different users, programs in time-sharing systems generally do not have much need to communicate with each other. Allocation and de-allocation of devices must be done in a manner that preserves system integrity and provides for good performance.

2.3.7 Real-time systems

Real time systems are used in time critical environments where data must be processed extremely quickly because the output influences immediate decisions. A real time system must be responsive in time which is measured in fractions of seconds. In real time systems the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time at which the results is produced. Real-time operating systems are used in environments where a large number of events, mostly external to the computer system, must be accepted and processed within certain deadlines.

A primary objective of real-time systems is to provide quick event-response times. User convenience and resource utilization are of secondary concern. It is not uncommon for a real-time system to be expected to process bursts of thousands of interrupts per second without missing a single event.

The Multitasking operation is accomplished by scheduling processes for execution independently of each other. Each process is assigned a certain level of priority that corresponds to the relative importance of the event that it services. The processor is normally allocated to the highest-priority process. Higher-priority processes usually preempt execution of the lower-priority processes. The process population in real-time systems is fairly static, and there is comparatively little moving of programs between primary and secondary storage. Processes in real-time systems tend to cooperate closely, thus necessitating support for both separation and sharing of memory. Time-critical device management is one of the main characteristics of real-time systems. In addition to providing sophisticated forms of interrupt management and I/O

buffering, real-time operating systems often provide system calls to allow user processes to connect themselves to interrupt vectors and to service events directly. File management is found only in larger installations of real-time systems. In fact, some embedded real-time systems may not even have any secondary storage.

2.3.8 Combination of operating systems

Different types of Operating System are optimized to serve the needs of specific environments. In practice, however, a given environment may not exactly fit any of the described molds. For instance, both interactive program development and lengthy simulations are often encountered in university computing centers. For this reason, some commercial operating systems provide a combination of described services. For example, a time-sharing system may support interactive users and also incorporate a full-fledged batch monitor. This allows computationally intensive non-interactive programs to be run concurrently with interactive programs. The common practice is to assign low priority to batch jobs and thus execute batched programs only when the processor would otherwise be idle. In other words, batch may be used as a filler to improve processor utilization while accomplishing a useful service of its own. Similarly, some time-critical events, such as receipt and transmission of network data packets, may be handled in real-time fashion on systems that otherwise provide time-sharing services to their terminal users.

2.3.9 Distributed Operating Systems

A distributed computer system is a collection of autonomous computer systems capable of communication and cooperation via their hardware and software interconnections. Distributed computer systems evolved from computer networks in which a number of largely independent hosts are connected by communication links and protocols. A distributed Operating System governs the operation of a distributed computer system and provides a virtual machine abstraction to its users. In distributed Operating System component and resource distribution should be hidden from users and application programs unless they explicitly demand. Distributed operating systems usually provide the means for system- wide sharing of resources, such as computational capacity, files, and I/O devices. In addition to typical operating-system services, a distributed Operating System may facilitate access to remote resources,

communication with remote processes, and distribution of computations.

2.4 MULTIPROCESSING OPERATING SYSTEM

In operating systems, to improve the performance of more than one CPU can be used within one computer system called Multiprocessor operating system.

Multiple CPUs are interconnected so that a job can be divided among them for faster execution. When a job finishes, results from all CPUs are collected and compiled to give the final output. Jobs needed to share main memory and they may also share other system resources among themselves. Multiple CPUs can also be used to run multiple jobs simultaneously.

For Example: UNIX Operating system is one of the most widely used multiprocessing systems.

To employ a multiprocessing operating system effectively, the computer system must have the following things:

- A motherboard is capable of handling multiple processors in a multiprocessing operating system.
- Processors are also capable of being used in a multiprocessing system.

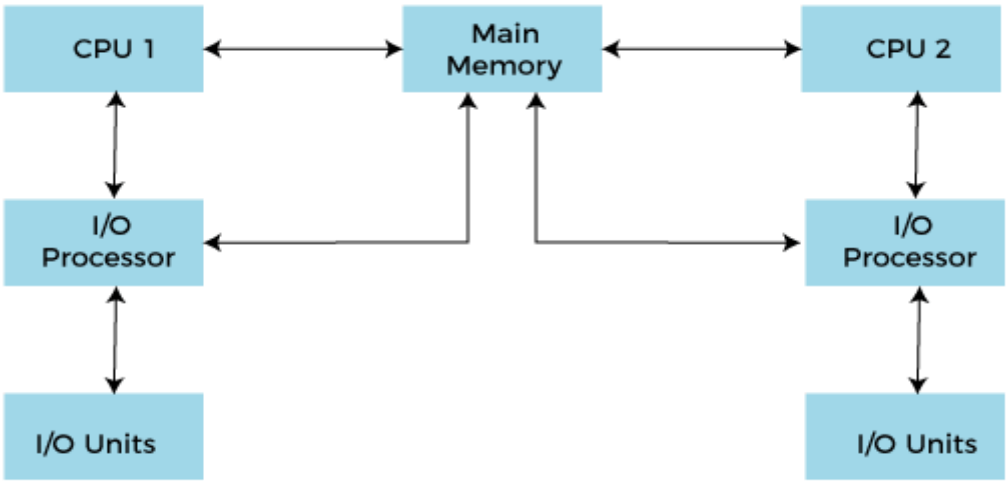
Advantages of multiprocessing operating system are:

- **Increased reliability:** Due to the multiprocessing system, processing tasks can be distributed among several processors. This increases reliability as if one processor fails; the task can be given to another processor for completion.
- **Increased throughout:** As several processors increase, more work can be done in less
- **The economy of Scale:** As multiprocessors systems share peripherals, secondary storage devices, and power supplies, they are relatively cheaper than single-processor systems.

Disadvantages of Multiprocessing operating System

- Operating system of multiprocessing is more complex and sophisticated as it takes care of multiple CPUs at the same time.

The basic organization of a typical multiprocessing system is shown in the given figure.



Working of Multiprocessor System

2.4.1 Types of multiprocessing systems

1. Symmetrical multiprocessing operating system
2. Asymmetric multiprocessing operating system

2.4.1.1 Symmetrical multiprocessing operating system:

In a Symmetrical multiprocessing system, each processor executes the same copy of the operating system, takes its own decisions, and cooperates with other processes to smooth the entire functioning of the system. The CPU scheduling policies are very simple. Any new job submitted by a user can be assigned to any processor that is least burdened. It also results in a system in which all processors are equally burdened at any time.

The symmetric multiprocessing operating system is also known as a "shared every-thing" system, because the processors share memory and the Input output bus or data path. In this system processors do not usually exceed more than 16.

Characteristics of Symmetrical multiprocessing operating system:

- In this system, any processor can run any job or process.
- In this, any processor initiates an Input and Output operation.

Advantages of Symmetrical multiprocessing operating system:

- These systems are fault-tolerant. Failure of a few processors does not bring the entire system to a halt.

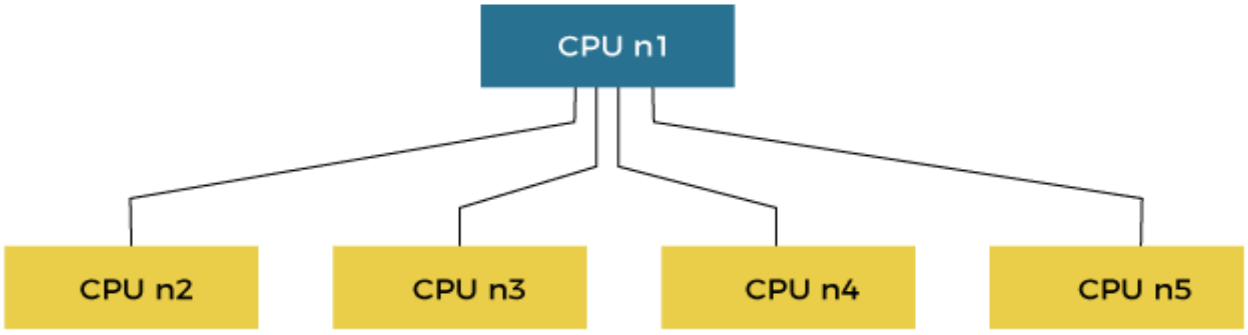
Disadvantages of Symmetrical multiprocessing operating system:

- It is very difficult to balance the workload among processors rationally.
- Specialized synchronization schemes are necessary for managing multiple processors.

2.4.1.2 Asymmetric multiprocessing operating system

In an asymmetric multiprocessing system, there is a master slave relationship between the processors.

Further, one processor may act as a master processor or supervisor processor while others are treated as shown below.



Asymmetric Multiprocessor System

In the above figure, the asymmetric processing system shows that CPU n1 acts as a supervisor whose function controls other following processors.

In this type of system, each processor is assigned a specific task, and there is a designated master processor that controls the activities of other processors.

For example, we have a math co-processor that can handle mathematical jobs better than the main CPU. Similarly, we have an MMX processor that is built to handle multimedia-related jobs. Similarly, we have a graphics processor to handle the graphics-related job better than the main processor. When a user submits a new job, the OS has to decide which processor can perform it better, and then that processor is assigned that newly arrived job. This processor acts as the master and controls the system.

All other processors look for masters for instructions or have predefined tasks. It is the responsibility of the master to allocate work to other processors.

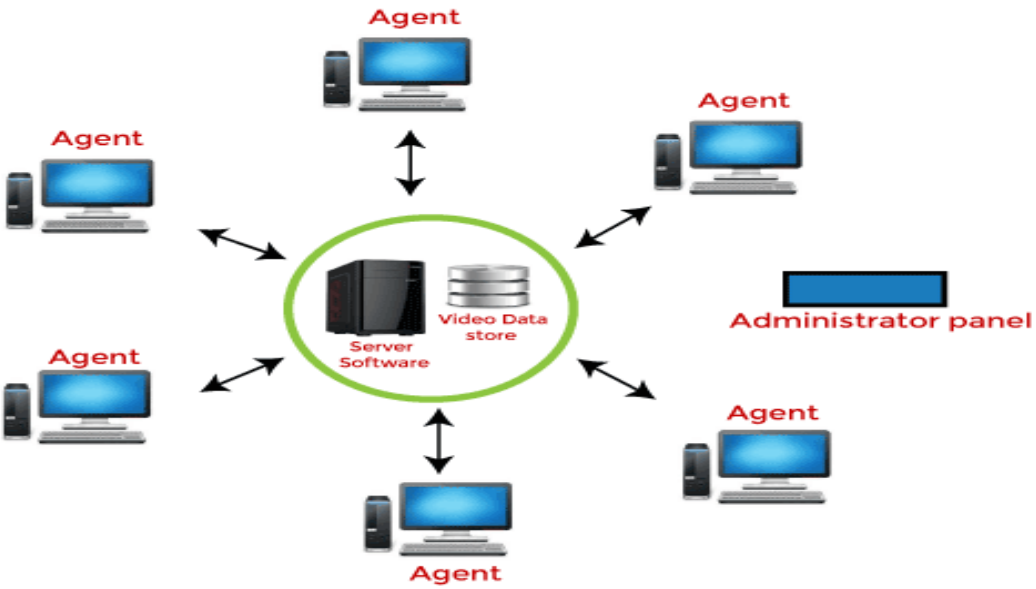
Advantages of Asymmetric multiprocessing operating system:

- In this type of system execution of Input and Output operation or an application program may be faster in some situations because many processors may be available for a single job.

Disadvantages of Asymmetric multiprocessing operating system:

- In this type of multiprocessing operating system the processors are unequally burdened. One processor may be having a long job queue, while another one may be sitting idle.
- In this system, if the process handling a specific work fails, the entire system will go down.

2.5 NETWORK OPERATING SYSTEM



An Operating system, which includes software and associated protocols to communicate with other autonomous computers via a network conveniently and cost-effectively, is called Network Operating System. It allows devices like a disk, printers, etc., shared between computers. The individual machines that are part of the Network have their operating system, and the Network Operating System resides on the top of the individual machines. Since individual machines have their Operating System to access

resources from other computers, they have to log into another machine using the correct password. This feature also results in no process migration, and processes running at different machines cannot communicate. The transmission control protocol is the common network protocol.

The various Features of the Network Operating System are given below.

- Network Operating System presents a few protection functions inclusive of login regulations via way of means of
- This kind of Operating System presents numerous net offerings and backup offerings.
- It presents numerous functions inclusive of guide for processors, computerized hardware detection, and guide multiprocessing of numerous
- It helps diverse auditing equipment with graphical interfaces.

2.5.1 Types of Network Operating System

Network operating systems can be specialized serve as:

1. Peer To Peer System
2. Client-Server System

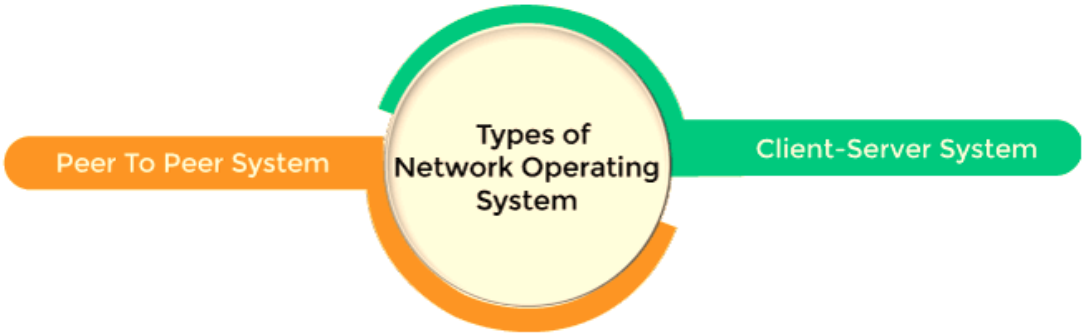
2.5.1.1 Peer To Peer Network Operating System

Peer To Peer networks are the network resources in which each system has the same capabilities and responsibilities, i.e., none of the systems in this architecture is superior to the others in terms of functionality.

There is no master-slave relationship among the systems, i.e., every node is equal in a Peer Peer Network Operating System. All the nodes at the Network have an equal relationship with others and have a similar type of software that helps the sharing of resources.

A Peer to Peer Network Operating System allows two or more computers to share their resources, along with printers, scanners, CD-ROM, etc., to be accessible from each computer. These networks are best suitable for smaller environments with 25 or fewer workstations.

To establish a Peer Peer Network, you need network adapter cards, properly installed network cabling to connect them and a network hub or switch to interconnect the computers.



Peer to Peer Networks is organized, simply a group of computers that can share resources. Each computer in a workstation keeps track of its user accounts and security settings, so no single computer is in charge of the workgroup. Workgroups have little security, and there is no central login process. Any user can use any shared resources once he logs into a peer on the Network. As there is no central security, sharing resources can be controlled by a password, or the user may stop the accessibility of certain files or folders by making them not shared.

Advantages of Peer To Peer Network Operating System

- This type of system is less expensive to set up and maintain.
- In this, dedicated hardware is not required.
- It does not require a dedicated network administrator to set up some network policies.
- It is very easy to set up as a simple cabling scheme is used, usually a twisted pair cable.

Disadvantages of Peer To Peer Network Operating System

- Peer to Peer networks are usually less secure because they commonly use share-level security.
- This failure of any node in a system affects the whole system.
- Its performance degrades as the Network grows.
- Peer to Peer networks cannot differentiate among network users who are accessing a resource.
- In Peer to Peer Network, each shared resource you wish to control must have its password. These multiple passwords may be difficult to remember.
- Lack of central control over the Network.

2.5.1.2 Client-Server Network Operating System

In Client-Server systems, there are two broad categories of systems:

- The server is called the backend.
- A client called as frontend.

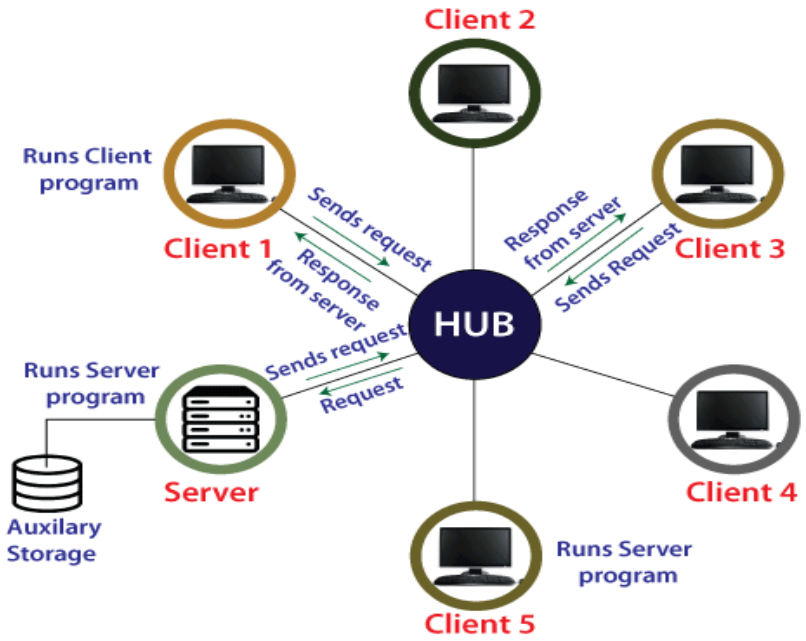
Client-Server Network Operating System is a server-based Network in which storage and processing workload is shared among clients and servers.

The client requests offerings which include printing and document storage, and servers satisfy their requests. Normally all community offerings like digital mail, printing are routed through the server.

Server computers systems are commonly greater effective than client computer systems. This association calls for software programs for the customers and servers. The software program walking at the server is known as the Network Operating System, which offers a community of surroundings for server and client.

Client-Server Network was developed to deal with the environment when many PC printers and servers are connected via a network. The fundamental concept changed to outline a specialized server with unique functionality.

For Example: Number of customers are related or connected to a file server that stores the files of client machines. Another system might be special as a Print Server to satisfy the printing request with the aid of using the diverse customers. Web servers or email servers are different specialized servers that may be utilized in a Client-Server system.



Client Server Network

A common application of Client-Server application involves a database that many computers on a network can access. The database is stored on the server, and the database queries are sent from clients

and processed by the server. The result of queries is then sent across the Network back to the clients. One server may provide too many clients at a time.

Advantages of Client-Server Network Operating System

- This Network is more secure than the Peer Peer Network system due to centralized data security.
- Network traffic reduces due to the division of work among clients and the server.
- The area covered is quite large, so it is valuable to large and modern organizations because it distributes storage and processing.
- The server can be accessed remotely and across multiple platforms in the Client-Server Network system.

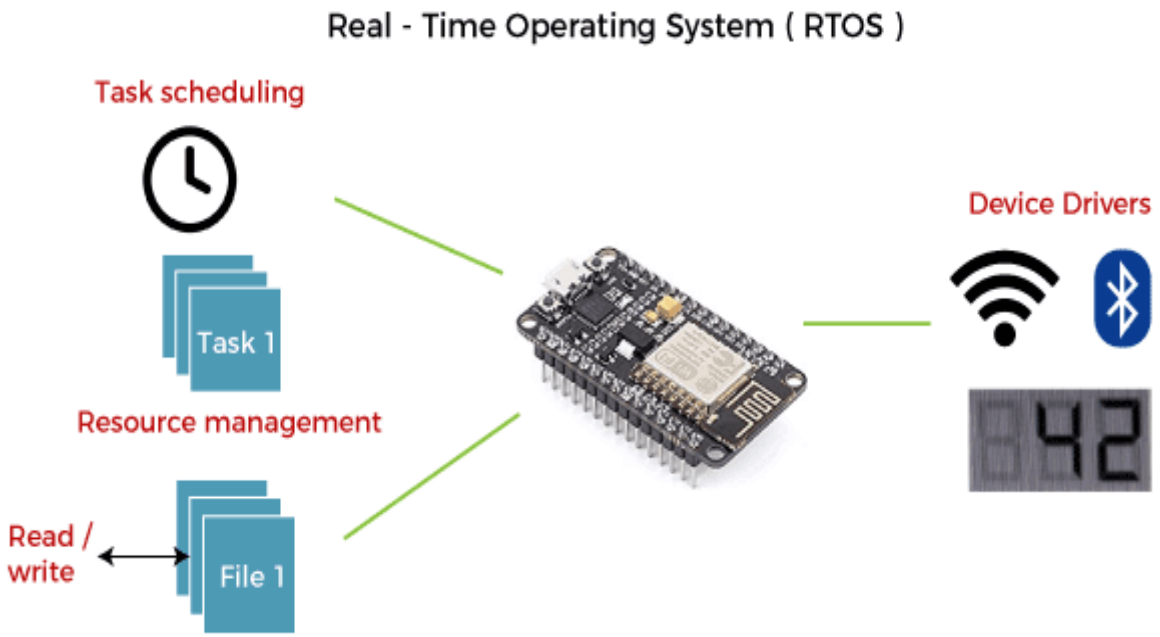
Disadvantages of Client-Server Network Operating Systems

- In Client-Server Networks, security and performance are important issues. So trained network administrators are required for network administration.
- Implementing the Client-Server Network can be a costly issue depending upon the security, resources, and connectivity.

2.6 Real Time Operating System (RTOS)

A real-time operating system (RTOS) is a special-purpose operating system used in computers that has strict time constraints for any job to be performed. It is employed mostly in those systems in which the results of the computations are used to influence a process while it is executing. Whenever an event external to the computer occurs, it is communicated to the computer with the help of some sensor used to monitor the event.

The sensor produces the signal that is interpreted by the operating system as an interrupt. On receiving an interrupt, the operating system invokes a specific process or a set of processes to serve the interrupt.



This process is completely uninterrupted unless a higher priority interrupt occurs during its execution. Therefore, there must be a strict hierarchy of priority among the interrupts. The interrupt with the highest priority must be allowed to initiate the process, while lower priority interrupts should be kept in a buffer that will be handled later. Interrupt management is important in such an operating system. Real-time operating systems employ special-purpose operating systems because conventional operating systems do not provide such performance.

The various examples of Real-time operating systems are:

- MTS
- Lynx
- QNX
- VxWorks etc.

2.6.1 Applications of Real-time operating system (RTOS):

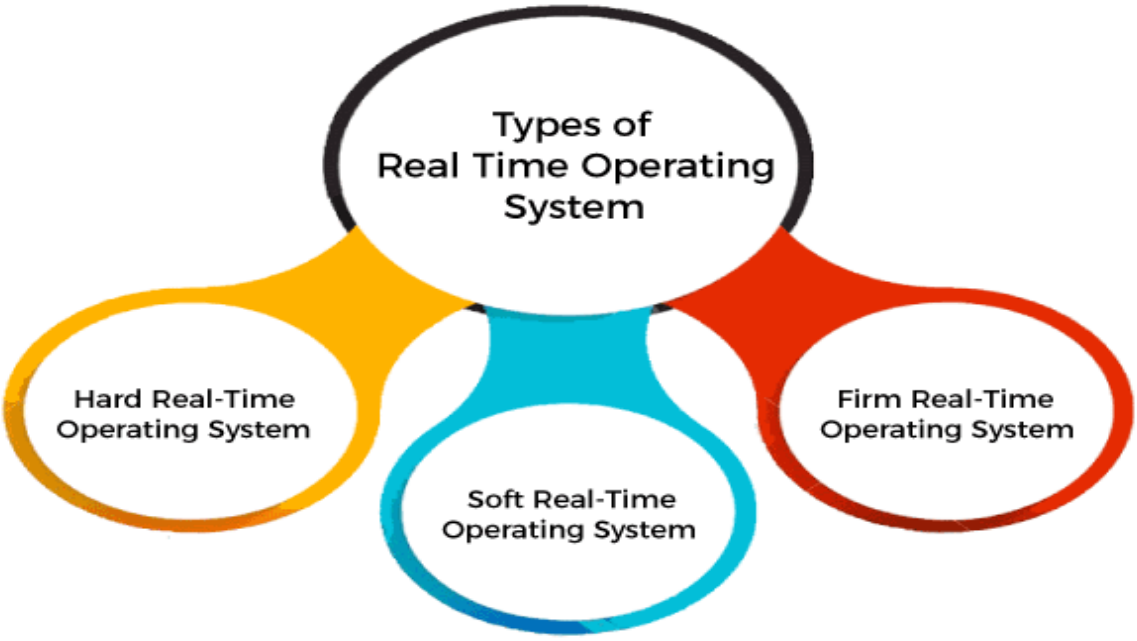
RTOS is used in real-time applications that must work within specific deadlines. Following are the common areas of applications of Real-time operating systems are given below.

- Real-time running structures are used inside the Radar gadget.
- Real-time running structures are utilized in Missile guidance.

- Real-time running structures are utilized in on line inventory trading.
- Real-time running structures are used inside the cell phone switching gadget.
- Real-time running structures are utilized by Air site visitors to manipulate structures.
- Real-time running structures are used in Medical Imaging Systems.
- Real-time running structures are used inside the Fuel injection gadget.
- Real-time running structures are used inside the Traffic manipulate gadget.
- Real-time running structures are utilized in Autopilot travel simulators.

2.6.2 Types of Real-time operating system

There are the three types of RTOS systems are:



2.6.2.1 Hard Real-Time operating system:

In Hard RTOS, all critical tasks must be completed within the specified time duration, i.e., within the given deadline. Not meeting the deadline would result in critical failures such as damage to equipment or even loss of human life.

Let's take an example of airbags provided by carmakers along with a handle in the driver's seat. When the driver applies brakes at a particular instance, the airbags grow and prevent the driver's head from

hitting the handle. Had there been some delay even of milliseconds, then it would have resulted in an accident.

Similarly, consider on-stock trading software. If someone wants to sell a particular share, the system must ensure that command is performed within a given critical time. Otherwise, if the market falls abruptly, it may cause a huge loss to the trader.

2.6.2.2 Soft Real-Time operating system:

Soft RTOS accepts a few delays via the means of the Operating system. In this kind of RTOS, there may be a closing date assigned for a particular job, but a delay for a small amount of time is acceptable. So, cut off dates are treated softly via means of this kind of RTOS. This type of system is used in Online Transaction systems and Livestock price quotation Systems.

2.6.2.3 Firm Real-Time operating system:

In Firm RTOS additionally want to observe the deadlines. However, lacking a closing date might not have a massive effect, however may want to purposely undesired effects, like a massive discount within the fine of a product.

For Example, this system is used in various forms of Multimedia applications.

Advantages of Real-time operating system:

The benefits of real-time operating system are as follows-:

- Easy to layout, develop and execute real-time applications under the real-time operating system.
- The real-time working structures are extra compact, so those structures require much less memory space.
- In a Real-time operating system, the maximum utilization of devices and systems.
- Focus on running applications and less importance to applications that are in the queue.
- Since the size of programs is small, RTOS can also be embedded systems like in transport and others.
- These types of systems are error-free.
- Memory allocation is best managed in these types of systems.

Disadvantages of Real-time operating system:

The disadvantages of real-time operating systems are as follows-

- Real-time operating systems have complicated layout principles and are very costly to develop.
- Real-time operating systems are very complex and can consume critical CPU cycles.

2.7 MOBILE OPERATING SYSTEM

A mobile operating system is an operating system that helps to run other application software on mobile devices. It is the same kind of software as the famous computer operating systems like Linux and Windows, but now they are light and simple to some extent.

The operating systems found on smart phones include Symbian OS, iPhone OS, RIM's BlackBerry, Windows Mobile, Palm WebOS, Android, and Maemo. Android, WebOS, and Maemo are all derived from Linux. The iPhone OS originated from BSD and NeXTSTEP, which are related to Unix.n It combines the beauty of computer and hand use devices. It typically contains a cellular built-in modem and SIM tray for telephony and internet connections. If you buy a mobile, the manufacturer company chooses the OS for that specific device.

2.7.1 Popular platforms of the Mobile OS

1. **Android OS:** The Android operating system is the most popular operating system today. It is a mobile OS based on the **Linux Kernel** and **open-source software.** The android operating system was developed by **Google**. The first Android device was launched in **2008**.

2. **Bada (Samsung Electronics):**Bada is a Samsung mobile operating system that was launched in 2010. The Samsung wave was the first mobile to use the bada operating system. The bada operating system offers many mobile features, such as 3-D graphics, application installation, and multipoint-touch.

3. **BlackBerry OS:** The BlackBerry operating system is a mobile operating system developed **by Research In Motion** (RIM). This operating system was designed specifically for BlackBerry handheld devices. This operating system is beneficial for the corporate users because it provides synchronization with Microsoft Exchange, Novell GroupWise email, Lotus Domino, and other business software when used with the BlackBerry Enterprise Server.

4. **iPhone OS / iOS:** The iOS was developed by the Apple inc for the use on its device. The iOS operating system is the most popular operating system today. It is a very secure operating system. The iOS operating system is not available for any other mobiles.

5. **Symbian OS:** Symbian operating system is a mobile operating system that provides a high-level of integration with communication. The Symbian operating system is based on the java language. It combines middleware of wireless communications and personal information management (PIM) functionality. The Symbian operating system was developed by **SymbianLtd** in **1998** for the use of mobile phones. **Nokia** was the first company to release Symbian OS on its mobile phone at that time.

6. **Windows Mobile OS:** The window mobile OS is a mobile operating system that was developed by **Microsoft**. It was designed for the pocket PCs and smart mobiles.

7. **Harmony OS:** The harmony operating system is the latest mobile operating system that was developed by Huawei for the use of its devices. It is designed primarily for IoT devices.

8. **Palm OS:** The palm operating system is a mobile operating system that was developed by **Palm Ltd** for use on personal digital assistants (PADs). It was introduced in **1996**. Palm OS is also known as the **Garnet OS**.

9. **WebOS (Palm/HP):** The WebOS is a mobile operating system that was developed by **Palm**. It based on the **Linux Kernel**. The HP uses this operating system in its mobile and touchpads.

2.8 CLOUD OPERATING SYSTEM

The cloud OS is a browser-based operating system that works for real-time support. It is developed for visualization in cloud computing. It provides a virtualized environment that runs on cloud computing. It helps to manage the machines, processes of virtual servers, execution, and infrastructure. It also manages software and back-end hardware resources. Although, the features of the cloud OS differs depending on the virtual environment and cloud services used. It is a lightweight OS that stores data and connects to a remote server to access web-based apps. Cloud OS include **Google Chrome** operating system and **Microsoft Windows Azure.**

As a cloud, users may manage the tasks from their mobile internet devices and tablets in the same way that they may manage them from their systems because the internet-based users may work on the go with a cloud operating system. A cloud operating system improves productivity by allowing greater

utilization of resources such as storage, processing, and a network of data centers. At the same time, it boosts employee productivity by delivering applications faster and more effectively across all clouds, including public, private, and hybrid. Additionally, it improves security and compliance.

2.8.1 Advantages and disadvantages of Cloud Operating System

There are various advantages and disadvantages of the cloud operating system. Some advantages and disadvantages of the cloud operating system are as follows:

Advantages

1. **Cost-Effective:** The Cloud OS is highly cost-effective. You must pay only once to get the service. On the other hand, when you use a regular operating system, there are many additional costs, such as an antivirus application, storage costs, and a monthly operating system membership price. Furthermore, people require some utility software to keep the system updated. There is no need for such a large investment in a cloud-based operating system.
2. **No risk of viruses:** Cloud antivirus management is a concern for cloud service providers. Because it does not directly contact with your computer, there is less difficulty and risk of the virus. It has decreased your pain, increased productivity, and provided you with a stress-free life. There is a virus risk with traditional OSs, but there is no virus risk if you use an online-based OS.
3. **Easy Software upgrades:** When you use a cloud-based OS, you don't have to worry about updating your OS. There is no requirement to consume extra money, time, or effort on upgrading. Traditional systems appear to take a long time to upgrade the operating system, but you always get the most recent version of the operating system on the online operating system.
4. **High Speed:** There are numerous causes for OS performance problems, such as open apps, insufficient memory, computer viruses, etc. In the case of cloud OS, there is no matter for compromise in speed. It is a very fast OS because it's not tied to any system hardware. So, it doesn't matter if you're utilizing little RAM and running numerous programs at the same time using cloud OS.

Disadvantages

1. It provides limited features.

2. Cloud operating system hardware failure may cause a loss in data.
3. It needs an internet connection continuously.
4. It doesn't perform on a low-speed connection.

2.8.2 Top Cloud Operating Systems

The booting of a cloud operating system takes only a few seconds due to its simplicity. It may be used as a stand-alone operating system or immediately boot into the main operating system to the preference. Here, you will learn the top cloud operating system one by one.

2.8.2.1 Netvibes

Netvibes is a cloud operating system that may be used as an ad-free cloud computing solution by running within a browser. Furthermore, the option to customize it takes it a step further because the characteristics are distinctive and fit in with the environment. Everything that happens is recorded on the dashboard: favorite websites, news, apps, social media, and smart gadgets.

Features of Netvibes

There are various features of Netvibes. Some features of Netvibes are as follows:

1. It includes a word editor, compatible environment for groupware, application for presentation, SMTP client for email, etc.
2. It is a free operating system that may download and install as a plug-in in any common browser. It primarily focuses on platform and device independence and collaboration with browsers and operating systems.
3. If you upgrade to the premium version, you'll be able to collaborate with your personal data, evaluate business matrices, and create drag-and-drop charts, among other things.
4. It provides support of more than 100 mobile devices, and you may change files for sharing when needed.
5. You may automate operations by defining triggers from the dashboard, and you may make data-driven decisions by looking at the visualization on the monitor.
6. Keeping data online enables the user to operate with both local and cloud applications while maintaining optimal security.

2.8.2.2 CloudMe

CloudMe was formerly known as **iCloud.** It is a European-based Cloud system that allows you to access your data from anywhere and file sharing, music streaming, and photo storage. CloudMe is free for personal users, but it also has commercial possibilities. It offers a GUI and may be easily handled by collaborating with others. It is suitable for freelancers, small businesses, and organizations with similar interests. It is a cloud-based desktop with information sharing and task management capabilities.

Features of CloudMe

There are various features of CloudMe. Some features of CloudMe are as follows:

1. It provides a list of the latest Hive activity and lets users send messages to users and groups.
2. Users may upload or import photographs from Flickr, messaging, and videos from YouTube and Vimeo.
3. It is simple to navigate between different accounts, engage with other members, and manage tasks.
4. It includes various facilities, such as bookmarking, online notes, tweeting, and a chat room.
5. It offers modular app architecture for plugging in the tools needed to plan and complete the job.

2.8.2.3 Amoeba OS: It is an advanced and important cloud operating system in this list. It is a general-purpose distributed OS. It was designed and developed by Andrew S. Tanenbaum and some others developers. Its primary purpose is to work as a single integrated system. It didn't launch any new version for the last 23 years.

It is a microkernel-based OS that provides multithreaded programs. It works like the remote procedure call mechanism to interact between the threats. It is written in the Python Programming Language. It has four fundamental designs: performance, parallelism, transparency, and distribution. It uses the high-performance FLIP network protocol for LAN connection. It includes various supported platforms like Sun 3/50 and 3/60, Motorola 68030, MIPS, NS 32016, i386/i486, SPARC, and VAX.

Features of Amoeba OS

There are various features of Amoeba OS. Some features of Amoeba OS are as follows:

1. It is designed so that it may operate in both distributed and parallel modes, depending on the project's need.
2. You don't have to allow different remote login instructions for different processors when using Amoeba. Similarly, there is no requirement for distinct remote mount actions for distant files.
3. The machine may be distributed across a building via a LAN even without a machine.
4. User-level programs determine the application's policy. As a result, the users don't need to know where the files are located or the position of the processors.

2.8.2.4 EyeOS

EyeOS is a cloud computing web desktop that is looking for ways to facilitate collaboration and communication. It's a web-based desktop application that runs on a private cloud. It is referred to as a cloud desktop due to its distinct user interface. It provides various features, including file management systems, client applications, collaborative tools, information tools, etc. This free cloud operating system is built on JavaScript, XML, and PHP.

Features of EyeOS

There are various features of EyeOS. Some features of EyeOS are as follows:

1. It offers a desktop interface with 67 programs and system benefits.
2. It offers free internet access points in public library networks, academia, and other web venues, allowing users to work with network administrators and communicate with them.
3. It supports the dynamic content. It is also equipped with a plethora of applications and a remote storage feature. Simultaneously, it supports independent browsers and platforms with sophisticated testing capabilities.
4. It takes only one web browser to provide access to the EyeOS and its apps.

2.8.2.5 Ghost OS

Ghost OS stands for the Global Hosted Operating System. It is one of the most famous cloud operating systems. It is an open-source OS and compatible with the x86 platforms. It provides storage services with security. It may also be used as a distributed system with various website channels. It might be used for both individual and business uses.

Features of Ghost OS

There are various features of Ghost OS. Some features of Ghost OS are as follows:

1. It is very easy to use. It also comes with quick files and folders managing facilities.
2. Any type of data may be uploaded in this OS from any device and may be viewed and edited with any browser.
3. Files may be moved very easily from a hard disk to the cloud or from the cloud to a hard disk with online editing capabilities.
4. You may share or transfer the file quickly by sending those links and mounting them as a Windows drive, just like a USB flash drive.

2.8.2.6 OSv is an open-source OS. It is designed and developed for better performance and effortless management. An overhead hypervisor may be minimized because it may provide extreme optimizations. It provides a straightforward API for managing any virtual machine component and qualifies it for virtual machine images. It provides the ability to execute Linux applications, but it is not Linux itself.

Features of OSv

There are various features of OSv. Some features of OSv are as follows:

1. It is useful for the critical management interface and serves as a mechanism for passing configuration data.
2. Users may download and execute the interface on a cloud server under the BSD license from any available resource.
3. It has its own app for image systems called Caspstan.
4. It also contains services for directly controlling JAVA apps and making the process straightforward.

2.9 Check your progress:

1. What is operating system?

   A. collection of programs that manages hardware resources

   B. system service provider to the application programs

C. link to interface the hardware and application programs

D. all of the mentioned

2. To access the services of operating system, the interface is provided by the:

A. system calls

B. API

C. library

D. assembly instructions

3. Which one of the following is not true?

A. kernel is the program that constitutes the central core of the operating system

B. kernel is the first part of operating system to load into memory during booting

C. kernel is made of various modules which cannot be loaded in running operating system

D. kernel remains in the memory during the entire computer session

4. Which one of the following error will be handling by the operating system?

A. power failure

B. lack of paper in printer

C. connection failure in the network

D. all of the mentioned

5. The main function of the command interpreter is:

A. to get and execute the next user-specified command

B. to provide the interface between the API and application program

C. to handle the files in operating system

D. none of the mentioned

6. By operating system, the resource management can be done via:

A. time division multiplexing

B. space division multiplexing

C. both (a) and (b)

D. none of the mentioned

2.10 Summary

The prime responsibility of operating system is to manage the resources of the computer system. In addition to these, Operating System provides an interface between the user and the bare machine. On the basis of their attributes and design objectives, different types of operating systems were defined and characterized with respect to scheduling and management of memory, devices, and files. The primary concerns of a time-sharing system are equitable sharing of resources and responsiveness to interactive requests. Real-time operating systems are mostly concerned with responsive handling of external events generated by the controlled system. Distributed operating systems provide facilities for global naming and accessing of resources, for resource migration, and for distribution of computation. Process scheduling is a very important function of an operating system. Three different schedulers may coexist and interact in a complex operating system: long-term scheduler, medium-term scheduler, and short-term scheduler. Of the presented scheduling disciplines, FCFS scheduling is the easiest to implement but is a poor performer. SRTN scheduling is optimal but unrealizable. RR scheduling is most popular in time- sharing environments, and event-driven and earliest-deadline scheduling are dominant in real-time and other systems with time-critical requirements. Multiple- level queue scheduling, and its adaptive variant with feedback, is the most general scheduling discipline suitable for complex environments that serve a mixture of processes with different characteristics.

2.11 Keywords

SPOOL: Simultaneous Peripheral Operations on Line

Task: An instance of a program in execution is called a process or a task.

Multitasking: The ability to execute more than one task at the same time is called as multitasking.

Real time: These systems are characterized by very quick processing of data because the output influences immediate decisions.

Multiprogramming: It is characterized by many programs simultaneously resident in memory,

and execution switches between programs.

Long-term scheduling: the decisions to introduce new processes for execution, or re-execution.

Medium-term scheduling: the decision to add to (grow) the processes that are fully or partially in memory.

Short-term scheduling: It decides which (Ready) process to execute next.

Non-preemptive scheduling: In it, process will continue to execute until it terminates, or makes an I/O request which would block the process, or makes an operating system call.

Preemptive scheduling: In it, the process may be pre-empted by the operating system when a new process arrives (perhaps at a higher priority), or an interrupt or signal occurs, or a (frequent) clock interrupt occurs.

2.12 Self-Assessment Questions (SAQ)

11. What are the objectives of an operating system? Discuss.
12. Differentiate between multiprogramming, multitasking, and multiprocessing.
13. What are the major functions performed by an operating system? Explain.
14. Discuss various process scheduling policies with their cons and pros.
15. Define process. What are the different states of a process? Explain using a process state transition diagram.
16. What are the objectives of a good scheduling algorithm?
17. Explain the term context switch; how does the cost of a context switch affect the choice of scheduling algorithm.
18. From the point of view of scheduling, briefly explain the different requirements imposed by the following types of system: (a) batch, (b) interactive, (c) real- time.
19. Explain the need to compromise quantum used in round-robin.
20. Why round-robin scheduling is not suitable in batch operated computer system?

2.13 Answer to check your progress

7. all of the mentioned
8. system calls
9. kernel is made of various modules which cannot be loaded in running operating system
10. all of the mentioned
11. to get and execute the next user-specified command
12. both (a) and (b)

1.14 Reference/ Suggested Readings

8. Operating System Concepts, 5th Edition, Silberschatz A., Galvin P.B., John Wiley and Sons.
9. Systems Programming and Operating Systems, 2nd Revised Edition, Dhamdhere D.M., Tata McGraw Hill Publishing Company Ltd., New Delhi.
10. Operating Systems, Madnick S.E., Donovan J.T., Tata McGraw Hill Publishing Company Ltd., New Delhi.
11. Operating Systems-A Modern Perspective, Gary Nutt, Pearson Education Asia, 2000.
12. Operating Systems, Harris J.A., Tata McGraw Hill Publishing Company Ltd., New Delhi, 2002.
13. https://www.javatpoint.com
14. https://www.tutorialspoint.com

Course: MCA-32	Writer: Ritu
Lesson: 3	Vetter:

Process and scheduling algorithms

Structure

3.0 Learning Objectives

The objective of this lesson is to make the students familiar with the process and scheduling algorithm. After studying this lesson, they will be familiar with:

1. Process states and transitions.
2. Different types of scheduler
3. Scheduling criteria
4. Scheduling algorithms
5. Inter process communication

3.1 Introduction

In nearly every computer, the most often requested resource is processor. Many computers have only one processor, so this processor must be shared via time-multiplexing among all the programs that need to execute on the computer. So processor management is an important function carried out by the operating system. Here we need to make an important distinction between a program and an executing program.

One of the most fundamental concepts of modern operating systems is the distinction between a program and the activity of executing a program. The former is merely a static set of directions; the latter is a dynamic activity whose properties change as time progresses. This activity is knows as a process. A process encompasses the current status of the activity, called the process state. This state includes the current position in the program being executed (the value of the program counter) as well as the values in the other CPU registers and the associated memory cells. Roughly speaking, the process state is a snapshot of the machine at that time. At different times during the execution of a program (at different times in a process) different snapshots (different process states) will be observed.

The operating system is responsible for managing all the processes that are running on a computer. It allocates each process a certain amount of time to use the processor. In addition, the operating system also allocates various other resources that processes will need such as computer memory or disks. To keep track of the state of all the processes, the operating system maintains a table known as the process table. Inside this table, every process is listed along with the resources the processes are using and the current state of the process. Processes can be in one of three states: running, ready, or waiting (blocked). The running state means that the process has all the resources it need for execution and it has been given

permission by the operating system to use the processor. Only one process can be in the running state at any given time. The remaining processes are either in a waiting state (i.e., waiting for some external event to occur such as user input or a disk access) or a ready state (i.e., waiting for permission to use the processor). In a real operating system, the waiting and ready states are implemented as queues, which hold the processes in these states. The assignment of physical processors to processes allows processors to accomplish work. The problem of determining when processors should be assigned and to which processes, is called processor scheduling or CPU scheduling.

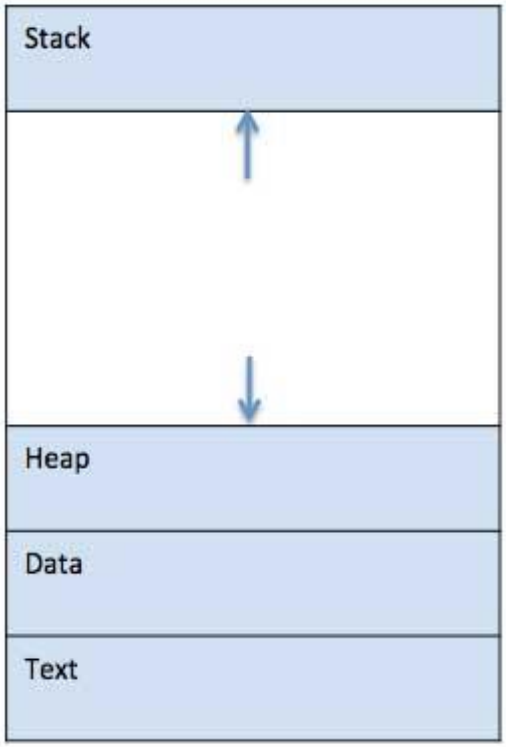
When more than one process is unable, the operating system must decide which one first. The part of the operating system concerned with this decision is called the scheduler, and algorithm it uses is called the scheduling algorithm. In operating system literature, the term "scheduling" refers to a set of policies and mechanisms built into the operating system that govern the order in which the work to be done by a computer system is completed. A scheduler is an Operating System module that selects the next job to be admitted into the system and the next process to run. The primary objective of scheduling is to optimize system performance in accordance with the criteria deemed most important by the system designers.

3.2 Process

A process is basically a program in execution. The execution of a process must progress in a sequential fashion. A process is defined as an entity which represents the basic unit of work to be implemented in the system.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory –



S.N.	Component & Description
1	**Stack** The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2	**Heap** This is dynamically allocated memory to a process during its run time.
3	**Text** This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4	**Data** This section contains the global and static variables.

3.2.1 Program

A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. For example, here is a simple program written in C programming language −

```
#include<stdio.h>


intmain()
{
printf("Hello, World! \n");
return0;
}
```

A computer program is a collection of instructions that performs a specific task when executed by a computer. When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program.

A part of a computer program that performs a well-defined task is known as an algorithm. A collection of computer programs, libraries and related data are referred to as software.

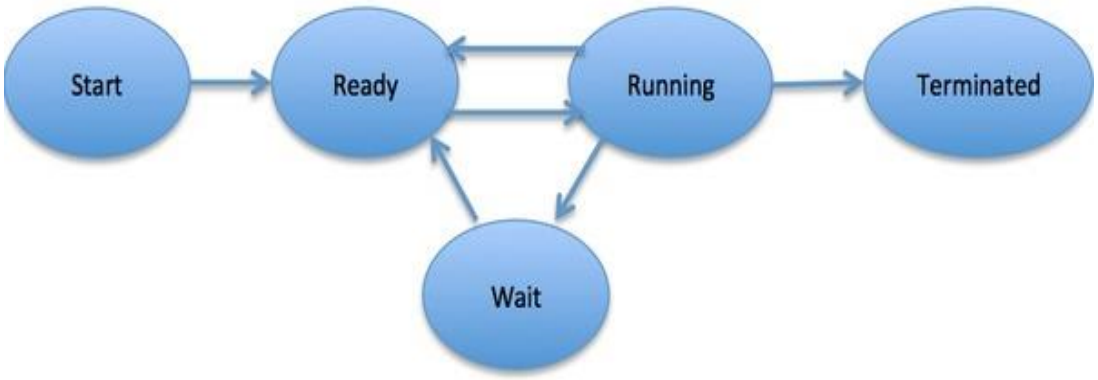
3.2.2 Process Life Cycle

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

In general, a process can have one of the following five states at a time.

S.N.	State & Description
1	**Start** This is the initial state when a process is first started/created.

2	**Ready** The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process.
3	**Running** Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.
4	**Waiting** Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
5	**Terminated or Exit** Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.



Five states at a time.

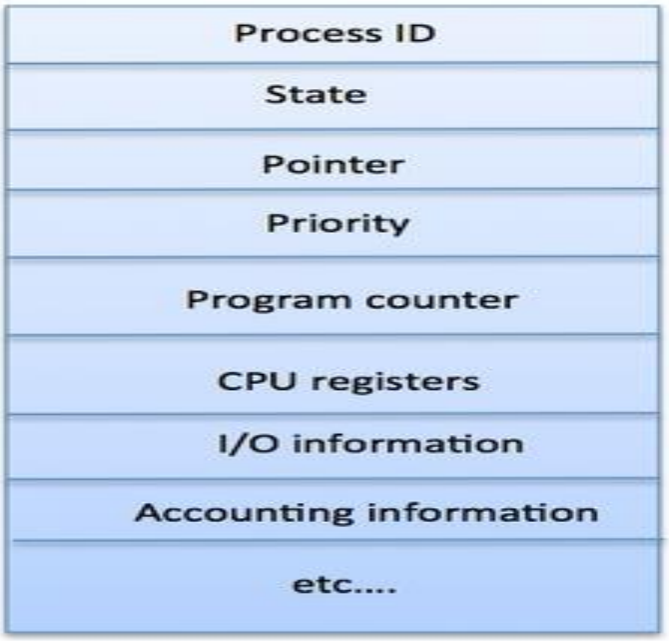
3.3 Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table –

S.N.	Information & Description
1	**Process State** The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2	**Process privileges** This is required to allow/disallow access to system resources.
3	**Process ID** Unique identification for each of the process in the operating system.
4	**Pointer** A pointer to parent process.
5	**Program Counter** Program Counter is a pointer to the address of the next instruction to be executed for this process.
6	**CPU registers** Various CPU registers where process need to be stored for execution for running state.
7	**CPU Scheduling Information** Process priority and other scheduling information which is required to schedule the process.

8	**Memory management information** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9	**Accounting information** This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	**IO status information** This includes a list of I/O devices allocated to the process.

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB −



The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

3.3.1 The process scheduling

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Process scheduling is an essential part of Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Categories of Scheduling

There are two categories of scheduling:

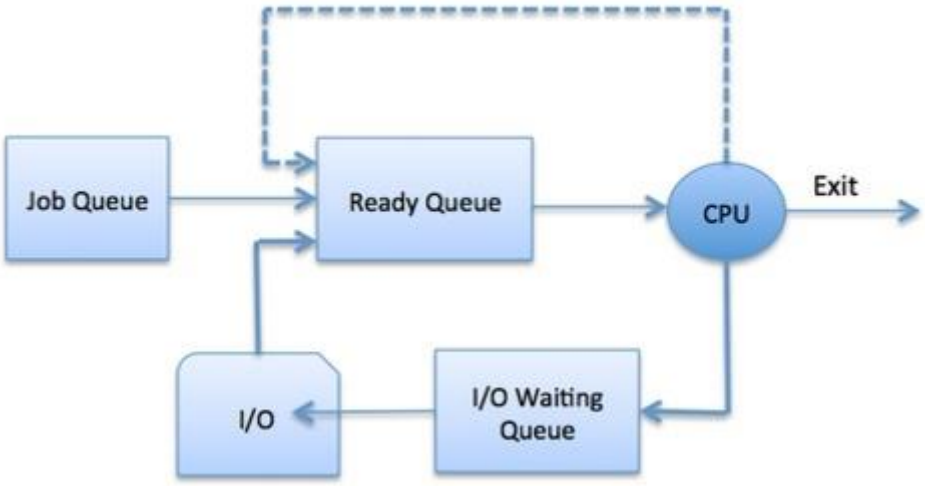
1. **Non-preemptive:** Here the resource can't be taken from a process until the process completes execution. The switching of resources occurs when the running process terminates and moves to a waiting state.
2. **Preemptive:** Here the OS allocates the resources to a process for a fixed amount of time. During resource allocation, the process switches from running state to ready state or from waiting state to ready state. This switching occurs as the CPU may give priority to other processes and replace the process with higher priority with the running process.

3.3.2 Process Scheduling Queues

The OS maintains all Process Control Blocks (PCBs) in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

- **Job queue** − this queue keeps all the processes in the system.
- **Ready queue** − this queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** − the processes which are blocked due to unavailability of an I/O device constitute this queue.



The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

7.3.3 Two-State Process Model

Two-state process model refers to running and non-running states which are described below −

S.N.	State & Description
1	**Running** When a new process is created, it enters into the system as in the running state.
2	**Not Running** Processes that are not running are kept in queue, waiting for their turn to execute. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list. Use of dispatcher is as follows. When a process is interrupted, that process is transferred in the waiting queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then selects a process from the queue to execute.

3.4 Schedulers

Schedulers are special system software which handles process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

3.4.1 Long Term Scheduler

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

3.4.2 Short Term Scheduler

It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

3.4.3 Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

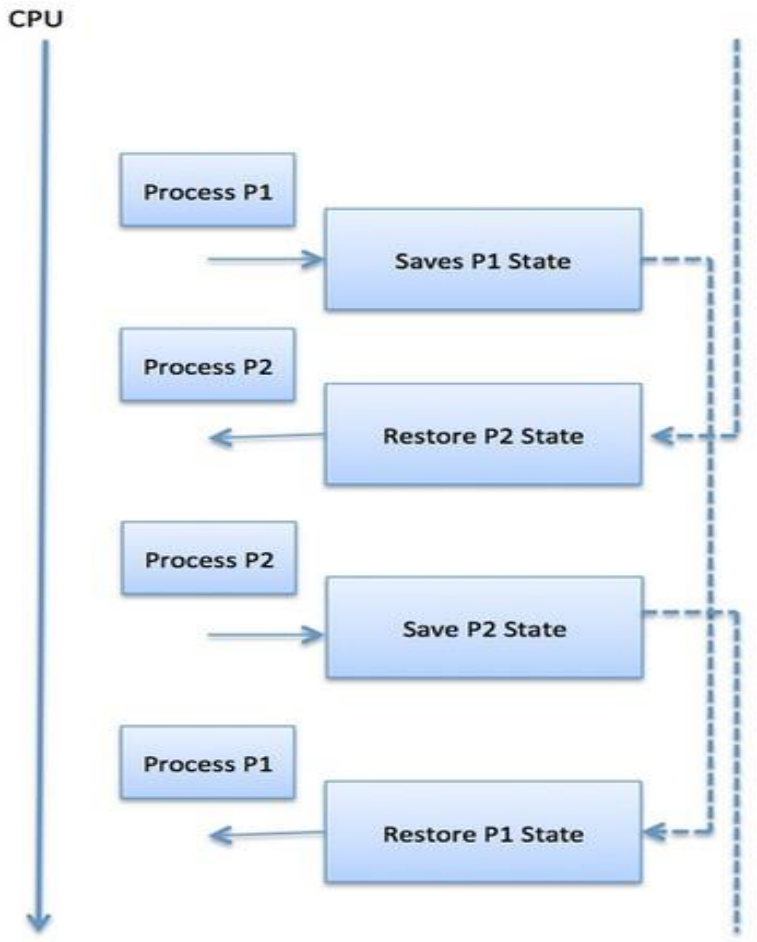
3.4.4 Comparison among Scheduler

S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

3.4.5 Context Switching

A context switching is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.



Context switches are computationally intensive since register and memory state must be saved and restored. To avoid the amount of context switching time, some hardware systems employ two or more

sets of processor registers. When the process is switched, the following information is stored for later use.

- Program Counter
- Scheduling information
- Base and limit register value
- Currently used register
- Changed State
- I/O State information
- Accounting information

3.5 Scheduling algorithms

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which we are going to discuss in this chapter −

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

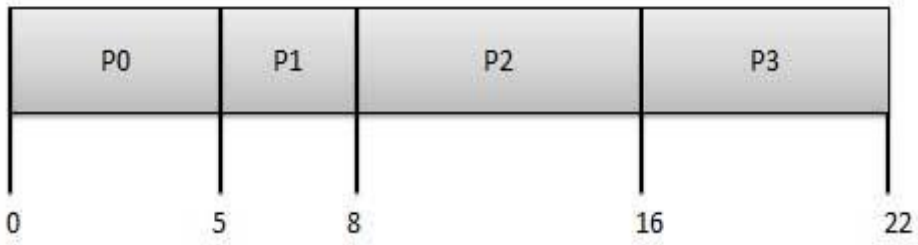
These algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state; it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.

- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.75

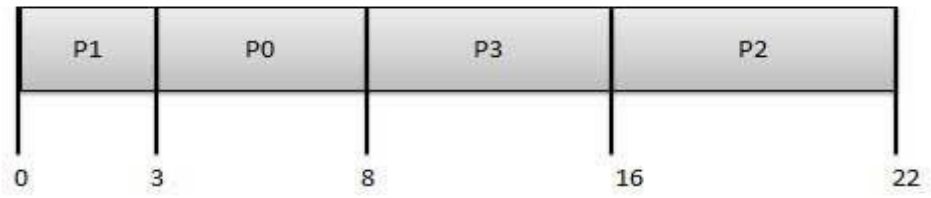
3.4.7 Shortest Job Next (SJN)

- This is also known as **shortest job first**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

Given: Table of processes, and their Arrival time, Execution time

Process	Arrival Time	Execution Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	14
P3	3	6	8

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8



Waiting time of each process is as follows −

Process	Waiting Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	14 - 2 = 12
P3	8 - 3 = 5

Average Wait Time: (0 + 4 + 12 + 5)/4 = 21 / 4 = 5.25

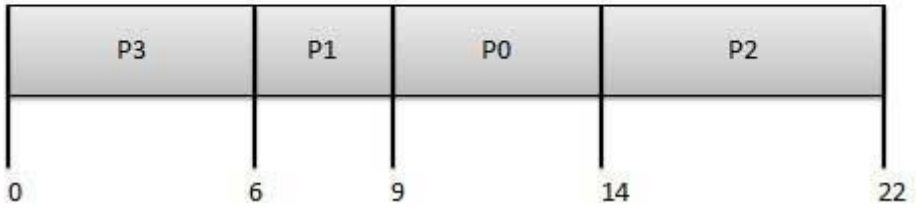
3.4.8 Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Given: Table of processes, and their Arrival time, Execution time, and priority. Here we are considering 1 is the lowest priority.

Process	Arrival Time	Execution Time	Priority	Service Time
P0	0	5	1	0
P1	1	3	2	11
P2	2	8	1	14
P3	3	6	3	5

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Waiting time of each process is as follows −

Process	Waiting Time
P0	0 - 0 = 0
P1	11 - 1 = 10
P2	14 - 2 = 12
P3	5 - 3 = 2

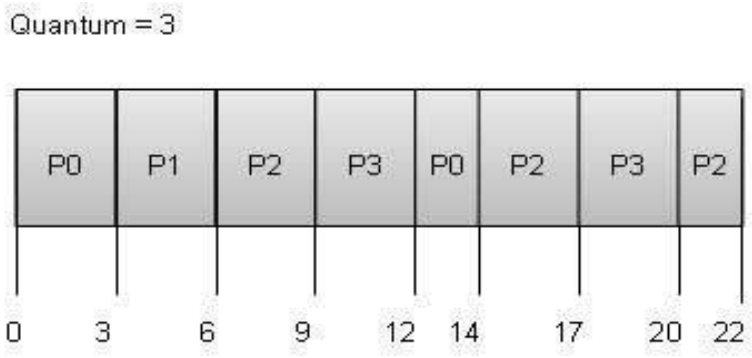
Average Wait Time: (0 + 10 + 12 + 2)/4 = 24 / 4 = 6

Shortest Remaining Time

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a quantum.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5

Multiple-Level Queues Scheduling

Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.

- Priorities are assigned to each queue.

For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

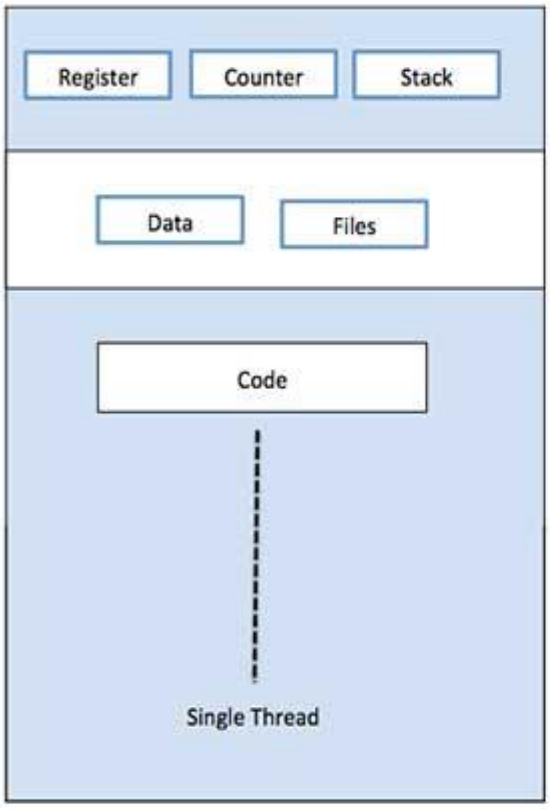
3.5 Thread

A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

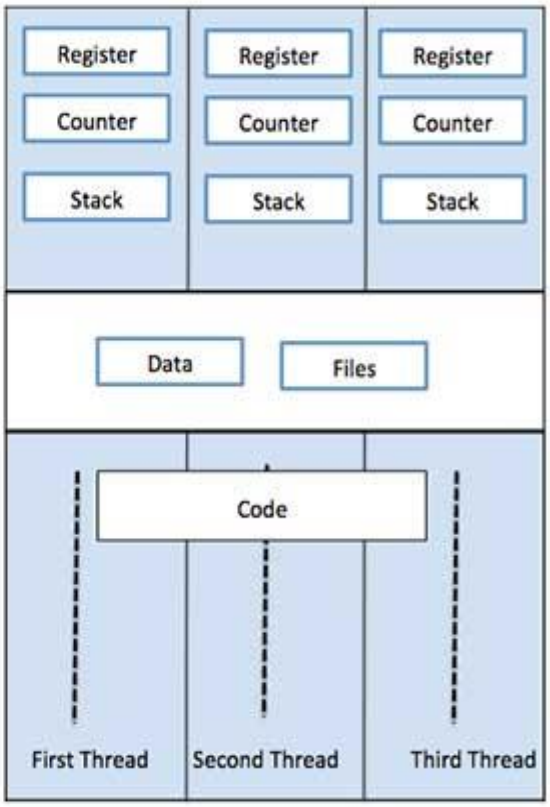
A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

A thread is also called a lightweight process. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.



Single Process P with single thread



Single Process P with three threads

Difference between Process and Thread

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.

4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

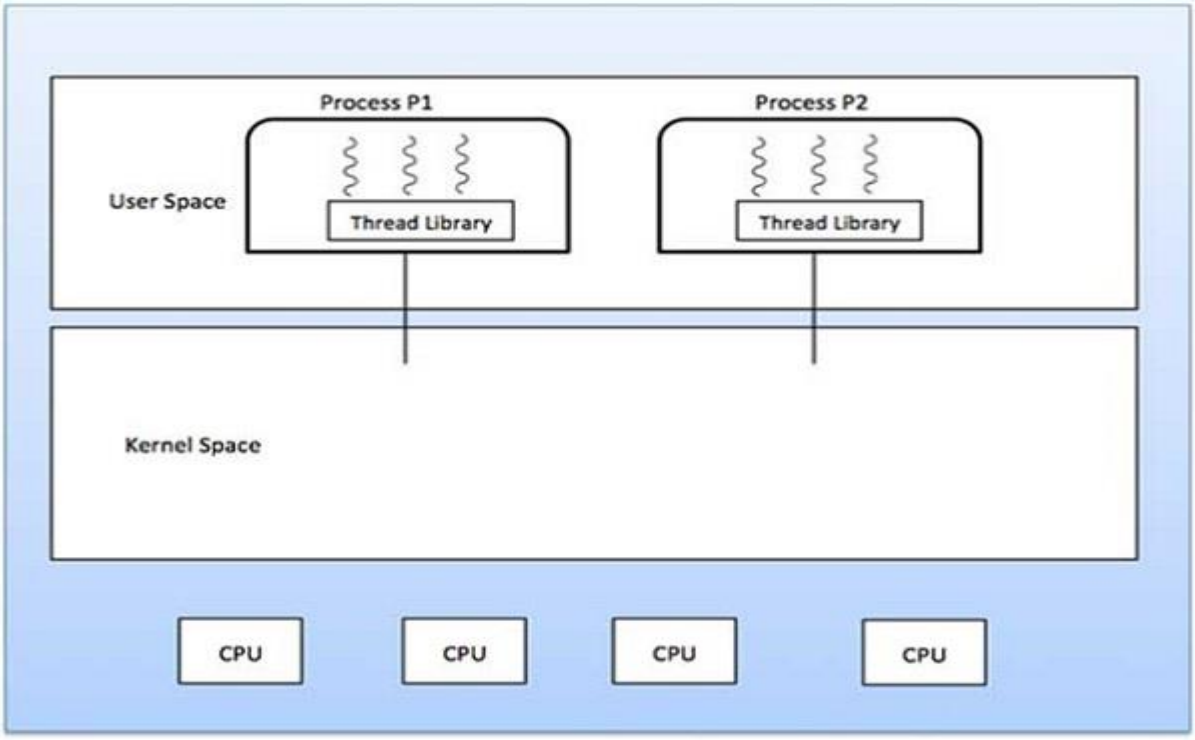
Types of Thread

Threads are implemented in following two ways −

- **User Level Threads** − User managed threads.
- **Kernel Level Threads** − Operating System managed threads acting on kernel, an operating system core.

3.5.1 User Level Threads

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.



Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

3.5.1 User Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individual's threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

3.5.3 Multithreading Models

Some operating system provides a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

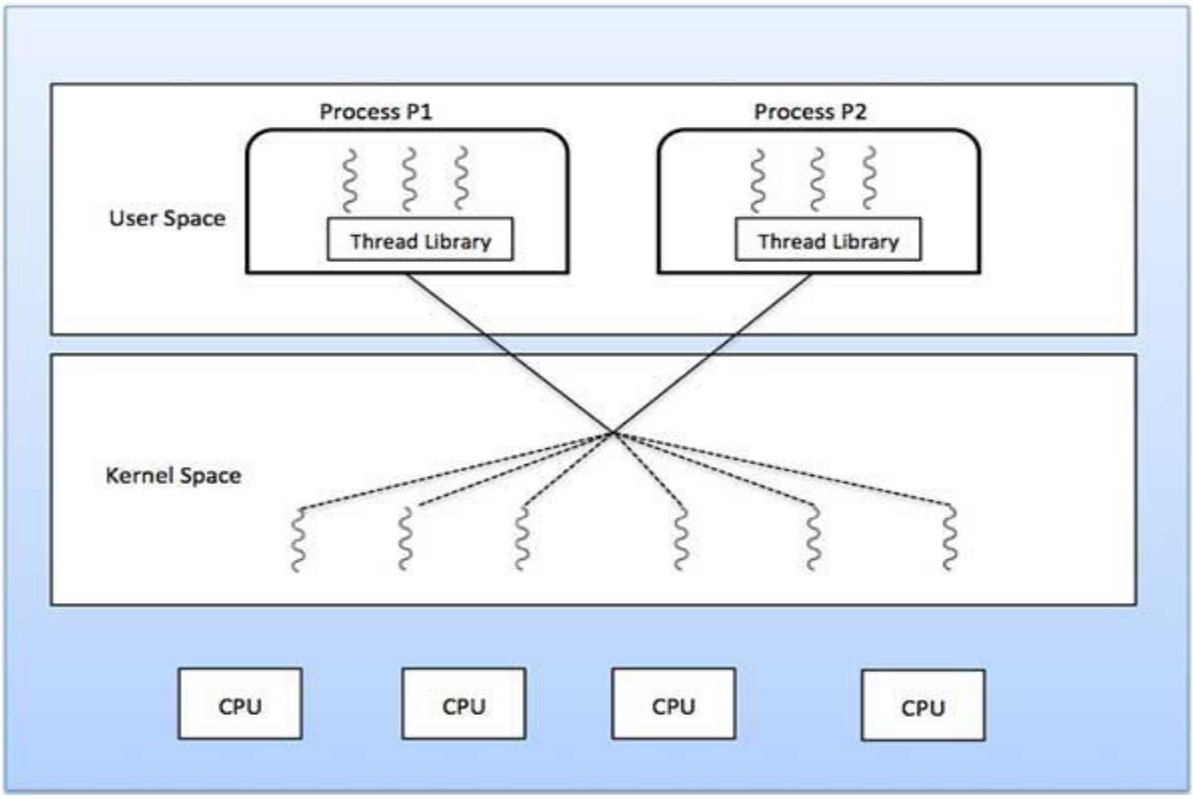
- Many to many relationship.
- Many to one relationship.
- One to one relationship.

3.5.3.1 Many to Many Model

The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.

The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine. This

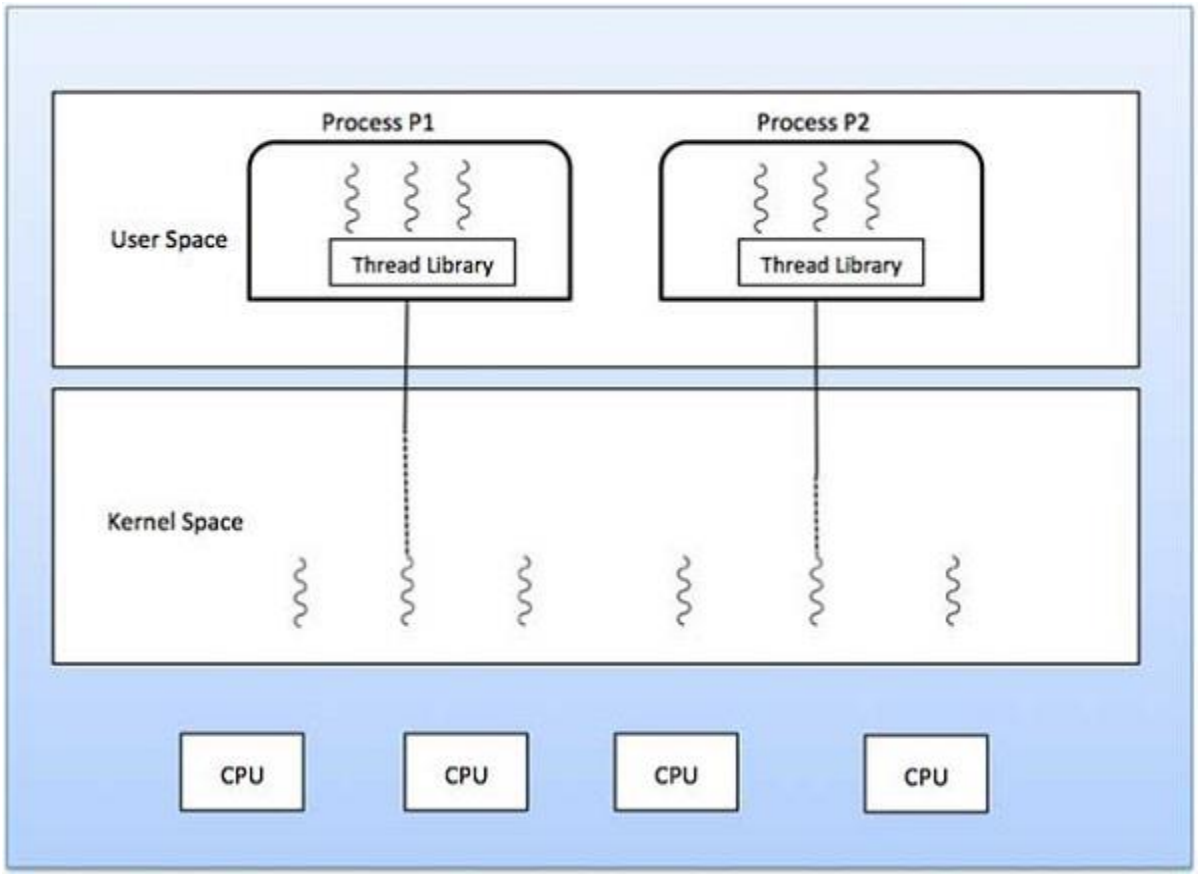
model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.



3.5.3.2 Many to One Model

Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

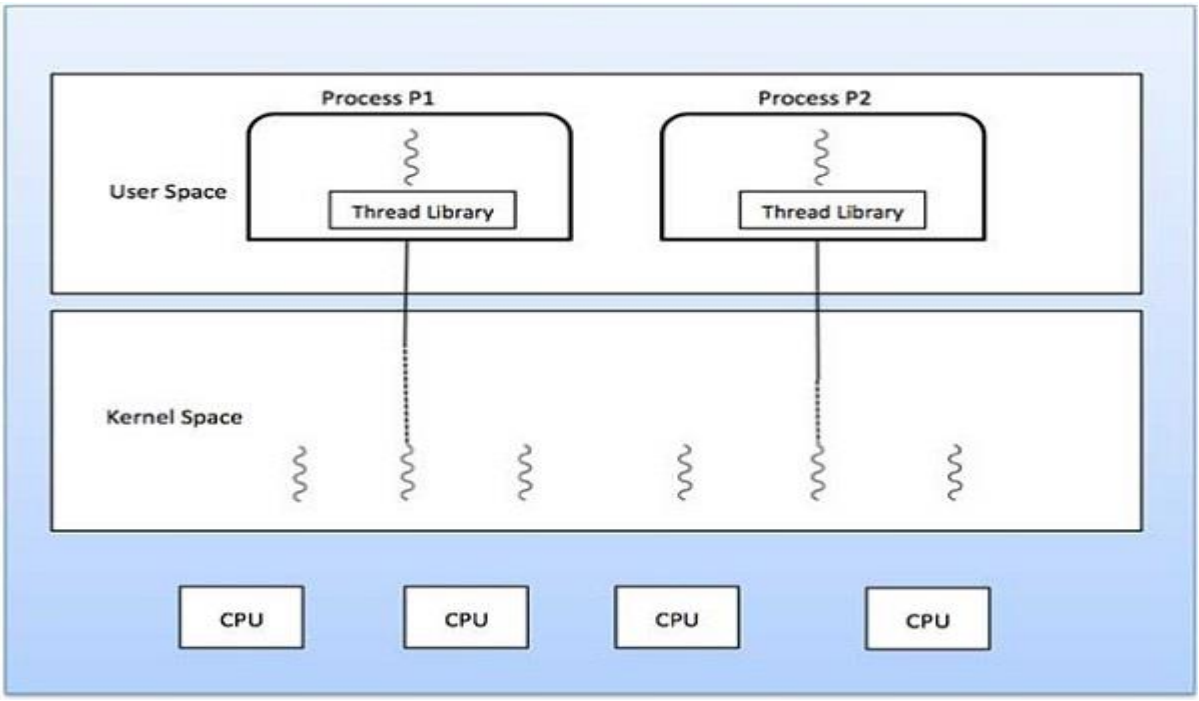
If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.



3.5.3.3 One to One Model

There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model. It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.



Difference between User-Level & Kernel-Level Thread

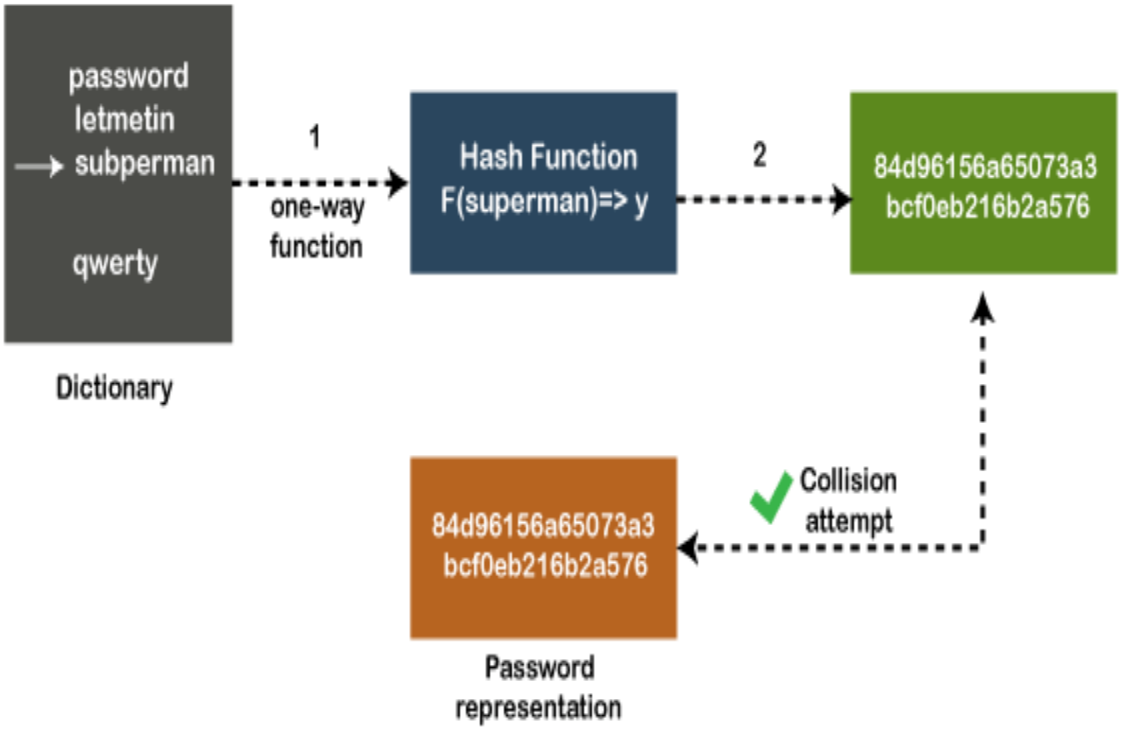
S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

3.6 Inter process Communication

In general, Inter Process Communication is a type of mechanism usually provided by the operating system (or OS). The main aim or goal of this mechanism is to provide communications in between several processes. In short, the intercommunication allows a process letting another process know that some event has occurred.

"Inter-process communication is used for exchanging useful information between numerous threads in one or more processes (or programs)."

To understand inter process communication, it can consider the following given diagram that illustrates the importance of inter-process communication:



3.6.1 Role of Synchronization in Inter Process Communication

It is one of the essential parts of inter process communication. Typically, this is provided by inter process communication control mechanisms, but sometimes it can also be controlled by communication processes.

These are the following methods that used to provide the synchronization:

1. **Mutual Exclusion**
2. **Semaphore**
3. **Barrier**
4. **Spinlock**

3.6.1.1 Mutual Exclusion:-

It is generally required that only one process thread can enter the critical section at a time. This also helps in synchronization and creates a stable state to avoid the race condition.

3.6.1.2 Semaphore:-

Semaphore is a type of variable that usually controls the access to the shared resources by several processes. Semaphore is further divided into two types which are as follows:

1. Binary Semaphore
2. Counting Semaphore

3.6.1.3 Barrier:-

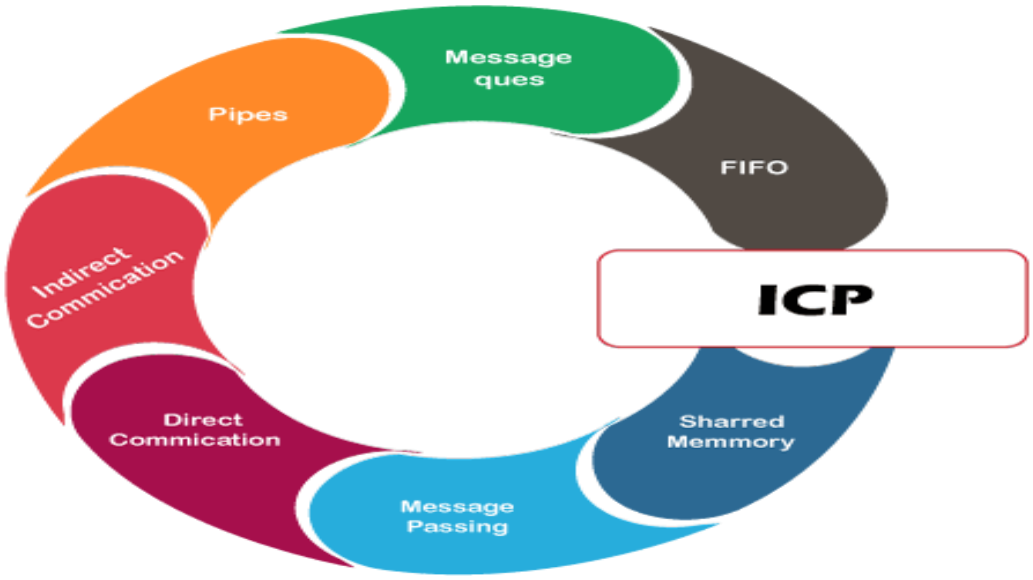
A barrier typically not allows an individual process to proceed unless all the processes do not reach it. It is used by many parallel languages, and collective routines impose barriers.

3.6.1.4 Spinlock:-

Spinlock is a type of lock as its name implies. The processes are trying to acquire the spinlock waits or stays in a loop while checking that the lock is available or not. It is known as busy waiting because even though the process active, the process does not perform any functional operation (or task).

3.7 Approaches to Inter process Communication

Here discussed some different approaches to inter-process communication which are as follows:



These are a few different approaches for Inter- Process Communication:

1. **Pipes**
2. **Shared Memory**
3. **Message Queue**
4. **Direct Communication**
5. **Indirect communication**
6. **Message Passing**
7. **FIFO**

3.7.1 Pipe:-

The pipe is a type of data channel that is unidirectional in nature. It means that the data in this type of data channel can be moved in only a single direction at a time. Still, one can use two-channel of this type, so that he can able to send and receive data in two processes. Typically, it uses the standard methods for input and output. These pipes are used in all types of POSIX systems and in different versions of window operating systems as well.

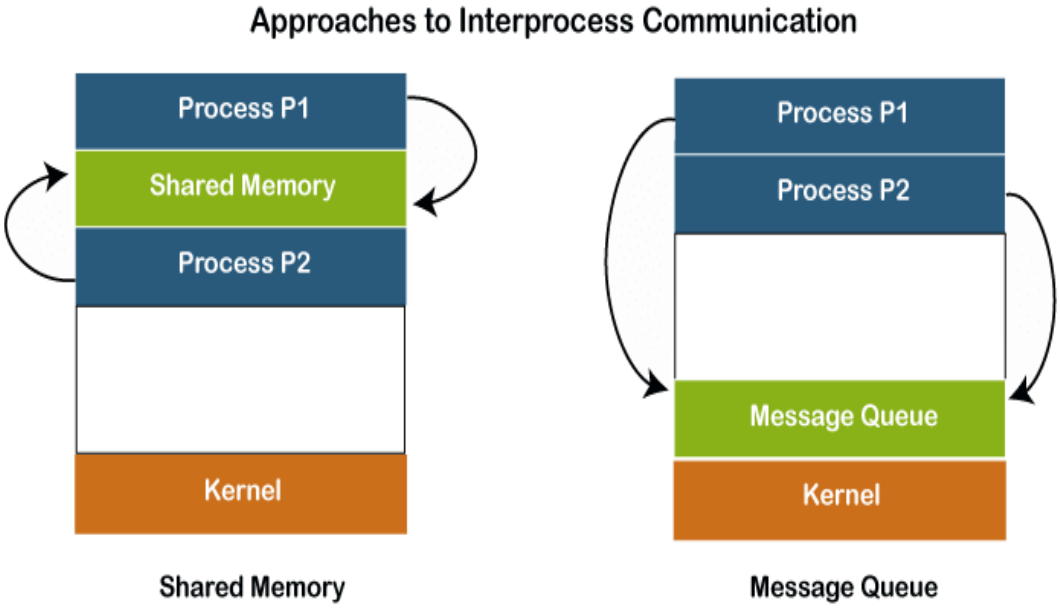
3.7.2 Shared Memory:-

It can be referred to as a type of memory that can be used or accessed by multiple processes simultaneously. It is primarily used so that the processes can communicate with each other. Therefore the shared memory is used by almost all POSIX and Windows operating systems as well.

3.7.3 Message Queue:-

In general, several different messages are allowed to read and write the data to the message queue. In the message queue, the messages are stored or stay in the queue unless their recipients retrieve them. In short, we can also say that the message queue is very helpful in inter-process communication and used by all operating systems.

To understand the concept of Message queue and Shared memory in more detail, its diagram given below:



3.7.4 Message Passing:-

It is a type of mechanism that allows processes to synchronize and communicate with each other. However, by using the message passing, the processes can communicate with each other without restoring the hared variables. Usually, the inter-process communication mechanism provides two operations that are as follows:

- send (message)
- received (message)

3.7.5 Direct Communication:-

In this type of communication process, usually, a link is created or established between two communicating processes. However, in every pair of communicating processes, only one link can exist.

3.7.6 Indirect Communication

Indirect communication can only exist or be established when processes share a common mailbox, and each pair of these processes shares multiple communication links. These shared links can be unidirectional or bi-directional.

3.7.7 FIFO:-

It is a type of general communication between two unrelated processes. It can also be considered as full-duplex, which means that one process can communicate with another process and vice versa.

3.8 Some other different approaches

- **Socket:-**

It acts as a type of endpoint for receiving or sending the data in a network. It is correct for data sent between processes on the same computer or data sent between different computers on the same network. Hence, it used by several types of operating systems.

- **File:-**

A file is a type of data record or a document stored on the disk and can be acquired on demand by the file server. Another most important thing is that several processes can access that file as required or needed.

- **Signal:-**

As its name implies, they are a type of signal used in inter process communication in a minimal way. Typically, they are the massages of systems that are sent by one process to another. Therefore, they are not used for sending data but for remote commands between multiple processes.

Usually, they are not used to send the data but to remote commands in between several processes.

3.9 Need inter process communication

There are numerous reasons to use inter-process communication for sharing the data. Here are some of the most important reasons:

- It helps to speedup modularity
- Computational
- Privilege separation
- Convenience
- Helps operating system to communicate with each other and synchronize their actions as well.

3.10 CHECK YOUR PROGRESS

1. Which module gives control of the CPU to the process selected by the short-term scheduler?
   a) Dispatcher
   b) Interrupt
   c) Scheduler
   d) None of the mentioned
2. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called _____
   a) job queue
   b) ready queue
   c) execution queue
   d) process queue
3. The interval from the time of submission of a process to the time of completion is termed as _____
   a) waiting time
   b) turnaround time
   c) response time
   d) throughput
4. Which scheduling algorithm allocates the CPU first to the process that requests the CPU first?
   a) first-come, first-served scheduling
   b) shortest job scheduling
   c) priority scheduling
   d) none of the mentioned

5. In priority scheduling algorithm _____
   a) CPU is allocated to the process with highest priority
   b) CPU is allocated to the process with lowest priority
   c) Equal priority processes cannot be scheduled
   d) None of the mentioned
6. Which algorithm is defined in Time quantum?
   a) shortest job scheduling algorithm
   b) round robin scheduling algorithm
   c) priority scheduling algorithm
   d) multilevel queue scheduling algorithm
7. Process are classified into different groups in _____
   a) shortest job scheduling algorithm
   b) round robin scheduling algorithm
   c) priority scheduling algorithm
   d) multilevel queue scheduling algorithm
8. In multilevel feedback scheduling algorithm _____
   a) a process can move to a different classified ready queue
   b) classification of ready queue is permanent
   c) processes are not classified into groups
   d) none of the mentioned
9. Which one of the following cannot be scheduled by the kernel?
   a) kernel level thread
   b) user level thread
   c) process
   d) none of the mentioned

3.11 SUMMARY

An important, although rarely explicit, function of process management is processor allocation. Three different schedulers may coexist & interact in a complex operating system: long-term scheduler, medium-term scheduler, & short-term scheduler. Of the presented scheduling disciplines, FCFS

scheduling is the easiest to implement but is a poor performer. SRTN scheduling is optimal but unrealizable. RR scheduling is most popular in time-sharing environments, & event-driven & earliest-deadline scheduling are dominant in real-time & other systems with time-critical requirements. Multiple-level queue scheduling, & its adaptive variant with feedback, is the most general scheduling discipline suitable for complex environments that serve a mixture of processes with different characteristics.

3.12 KEYWORDS

Short-term scheduling: the decisions as to which (Ready) process to execute next.

Non-preemptive scheduling: In non-preemptive scheduling, process will continue to execute until it terminates, or makes an I/O request which would block the process, or makes an operating system call.

In preemptive scheduling, the process may be pre-empted by the operating system when a new process arrives (perhaps at a higher priority), or an interrupt or signal occurs, or a (frequent) clock interrupt occurs.

File:- A file is a type of data record or a document stored on the disk and can be acquired on demand by the file server

3.13 Self-Assessment Questions (SAQ)

1. Which type of scheduling is used in real life operating systems? Why?
2. Which action should the short-term scheduler take when it is invoked but no process is in the ready state? Is this situation possible?
3. How can we compare performance of various scheduling policies before actually implementing them in an operating system?
4. SJF is a sort of priority scheduling. Comment.
5. What do you understand by starvation? How does SJF cause starvation? What is the solution of this problem?
6. What qualities are to be there in a scheduling policy? Explain.
7. Differentiate between user-oriented scheduling criteria & system-oriented scheduling criteria.

3.14 ANSWER TO CHECK YOUR PROGRESS

1. Dispatcher
2. Ready queue
3. Turnaround time
4. First-come, first-served scheduling
5. CPU is allocated to the process with highest priority
6. Round robin scheduling algorithm
7. Multilevel queue scheduling algorithm
8. A process can move to a different classified ready queue
9. User level thread

3.15 Reference and suggested readings

1. Operating System Concepts, 5th Edition, Silberschatz A., Galvin P.B., John Wiley & Sons.
2. Systems Programming & Operating Systems, 2nd Revised Edition, Dhamdhere D.M., Tata McGraw Hill Publishing Company Ltd., New Delhi.
3. Operating Systems, Madnick S.E., Donovan J.T., Tata McGraw Hill Publishing Company Ltd., New Delhi.
4. Operating Systems-A Modern Perspective, Gary Nutt, Pearson Education Asia, 2000.
5. Operating Systems, Harris J.A., Tata McGraw Hill Publishing Company Ltd., New Delhi, 2002.

Course: MCA-32	Writer: Ritu
Lesson: 4	Vetter:

CPU Scheduling

Structure

4.0 LEARNING OBJECTIVES

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.The objective of this lesson is to make the students familiar with the various issues of CPU scheduling. After studying this lesson, they will be familiar with:

1. Scheduling criteria
2. Scheduling algorithms

4.1 INTRODUCTION TO CPU SCHEDULING

In nearly every computer, the resource that is most often requested is the CPU or processor. Many computers have only one processor, so this processor must be shared via time-multiplexing among all the programs that need to execute on the computer. Here we need to make an important distinction between a program & an executing program.

"One of the most fundamental concepts of modern operating systems is the distinction between a program & the activity of executing a program. The former is merely a static set of directions; the latter is a dynamic activity whose properties change as time progresses. This activity is knows as a process. A process encompasses the current status of the activity, called the process state. This state includes the current position in the program being executed (the value of the program counter) as well as the values in the other CPU registers & the associated memory cells. Roughly speaking, the process state is a snapshot of the machine at that time. At different times during the execution of a program (at different times in a process) different snapshots (different process states) will be observed."

The operating system is responsible for managing all the processes that are running on a computer & allocated each process a certain amount of time to use the processor. In addition, the operating system also allocates various other resources that processes will need such as computer memory or disks. To keep track of the state of all the processes, the operating system maintains a table known as the process

table. Inside this table, every process is listed along with the resources the processes are using & the current state of the process. Processes can be in one of three states: running, ready, or waiting (blocked). The running state means that the process has all the resources it need for execution & it has been given permission by the operating system to use the processor. Only one process can be in the running state at any given time. The remaining processes are either in a waiting state (i.e., waiting for some external event to occur such as user input or a disk access) or a ready state (i.e., waiting for permission to use the processor). In a real operating system, the waiting & ready states are implemented as queues, which hold the processes in these states. The assignment of physical processors to processes allows processors to accomplish work.

Scheduling of processes/work is done to finish the work on time. CPU Scheduling is a process that allows one process to use the CPU while another process is delayed (in standby) due to unavailability of any resources such as I / O etc, thus making full use of the CPU. The purpose of CPU Scheduling is to make the system more efficient, faster, and fairer.

Whenever the CPU becomes idle, the operating system must select one of the processes in the line ready for launch. The selection process is done by a temporary (CPU) scheduler. The Scheduler selects between memory processes ready to launch and assigns the CPU to one of them.

When more than one process is runable, the operating system must decide which one first. The part of the operating system concerned with this decision is called the scheduler, & algorithm it uses is called the scheduling algorithm. In operating system literature, the term "scheduling" refers to a set of policies & mechanisms built into the operating system that govern the order in which the work to be done by a computer system is completed. A scheduler is an OS module that selects the next job to be admitted into the system & the next process to run. The primary objective of scheduling is to optimize system performance in accordance with the criteria deemed most important by the system designers.

4.2 SCHEDULING & PERFORMANCE CRITERIA

The objectives of a good scheduling policy include

- Fairness.
- Efficiency.
- Low response time (important for interactive jobs).
- Low turnaround time (important for batch jobs).

- High throughput
- Repeatability.
- Fair across projects.
- Degrade gracefully under load.

The success of the short-term scheduler can be characterized by its success against user-oriented criteria under which a single user (selfishly) evaluates their perceived response, or system-oriented criteria where the focus is on efficient global use of resources such as the processor & memory. A common measure of the system-oriented criteria is tough put, the rate at which tasks are completed. On a single-user, interactive operating system, & the user-oriented criteria take precedence:

it is unlikely that an individual will exhaust resource consumption, but responsiveness remains all important. On a multi-user, multi-tasking system, the global system-oriented criteria are more important as they attempt to provide fair scheduling for all, subject to priorities & available resources.

4.2.1 User-Oriented Scheduling Criteria

Response time

In an interactive system this measures the time between submissions of a new process request & the commencement of its execution. Alternatively, it can measure the time between a user issuing a request to interactive input (such as a prompt) & the time to echo the user's input or accept the carriage return.

Turnaround time

This is the time between submission of a new process & its completion. Depending on the mixture of current tasks, two submissions of identical processes will likely have different turnaround times. Turnaround time is the sum of execution & waiting times.

Deadlines

In a genuine real-time operating system, hard deadlines may be requested by processes. These either demands that the process is completed with a guaranteed upper-bound on its turnaround time, or provide a guarantee that the process will receive the processor in a guaranteed maximum time in the event of an interrupt. A real-time long-term scheduler should only accept a new process if it can guarantee required deadlines. In combination, the short-term scheduler must also meet these deadlines.

Predictability

With lower importance, users expect similar tasks to take similar times. Wild variations in response & turnaround times are distracting.

4.2.2 System Oriented Scheduling Criteria

Throughput

The short-term scheduler attempts to maximize the number of completed jobs per unit time. While this is constrained by the mixture of jobs, & their execution profiles, the policy affects utilization & thus completion.

Processor utilization

The percentage of time that the processor may be fed with work from Ready. In a single-user, interactive system, processor utilization is very unlikely to exceed a few percent.

Fairness

Subject to priorities, all processes should be treated fairly, & none should suffer processor starvation. This simply implies, in most cases, that all processes are moved to the ends of their respective state queues, & may not "jump the queue".

Priorities

Conversely, when processes are assigned priorities, the scheduling policy should favor higher priorities.

4.3 Level of scheduling

Two-level scheduling is an efficient scheduling method that uses two schedulers to perform process scheduling.

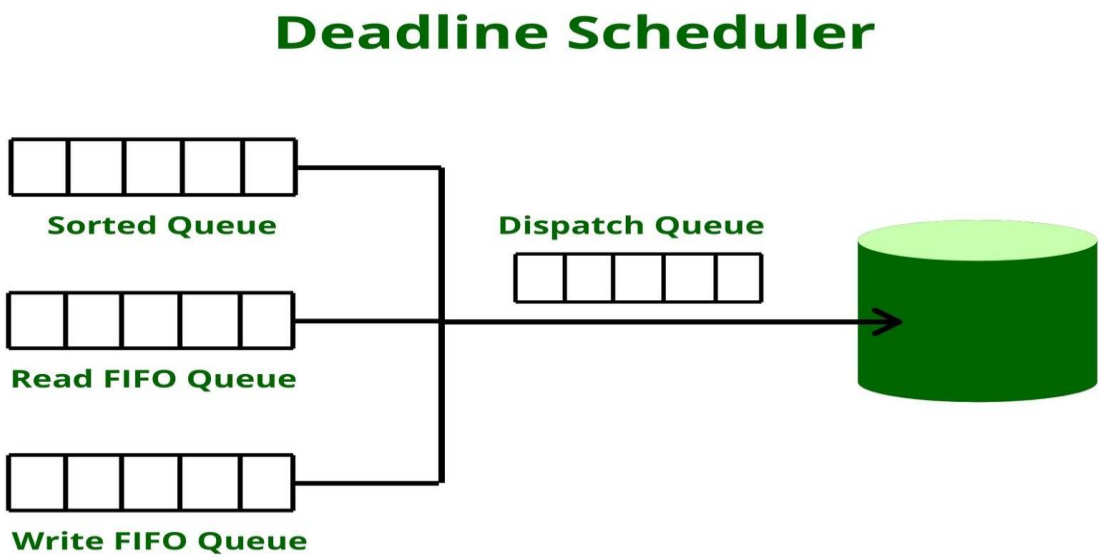
For example:

Suppose a system has 50 running processes all with equal priority and the system's memory can only hold 10 processes simultaneously. Thus, 40 processes are always swapped out and written on virtual memory on the hard disk. To swap out and swap in a process, it takes 50 ms respectively. Let us take up the above scenario with straightforward Round-robin scheduling: a process would need to be swapped in (least recently used processes are swapped in) every time when a context switch occurs. Swapping in and out costs too much, and the unnecessary swaps waste much time of scheduler.

So the solution to the problem is **two-level scheduling**. There are two different schedulers in two-level scheduling :

1. Lower level scheduler

2. Higher level scheduler



1. **Lower level scheduler –**
   This scheduler selects which process will run from memory.
2. **Higher level scheduler –**
   This scheduler focuses on swapping in and swapping out the processes between hard disk and memory. Swapping takes much time, therefore it does its scheduling much less often. It also swaps out the processes which are running for a long time in memory and are swapped with processes on disk that have not run for a long time.

Following variables are used :

Response time –

Response time variable is important as it prevents resource starvation and a process will be completed. Some other processes will have to wait unnecessarily for a long time if a process is swapped out for too long. Therefore, this variable is essential.

1. **Size of the process –**

   Larger processes are less often swapped because they take a long time to swap. As these processes are larger, only some of them can share the memory with the process.

2. **Priority –**

   The process with higher priority stays in memory for longer time so that it completes faster.

4.4 SCHEDULER DESIGN

Design process of a typical scheduler consists of selecting one or more primary performance criteria & ranking them in relative order of importance. The next step is to design a scheduling strategy that maximizes performance for the specified set of criteria while obeying the design constraints. One should intentionally avoid the word "optimization" because most scheduling algorithms actually implemented do not schedule optimally. They are based on heuristic techniques that yield good or near-optimal performance but rarely achieve absolutely optimal performance. The primary reason for this situation lies in the overhead that would be incurred by computing the optimal strategy at run-time, & by collecting the performance statistics necessary to perform the optimization. Of course, the optimization algorithms remain important, at least as a yardstick in evaluating the heuristics. Schedulers typically attempt to maximize the average performance of a system, relative to a given criterion. However, due consideration must be given to controlling the variance & limiting the worst-case behavior. For example, a user experiencing 10-second response time to simple queries has little consolation in knowing that the system's average response time is under 2 seconds.

One of the problems in selecting a set of performance criteria is that they often conflict with each other. For example, increased processor utilization is usually achieved by increasing the number of active processes, but then response time deteriorates. As is the case with most engineering problems, the design of a scheduler usually requires careful balance of all the different requirements & constraints. With the knowledge of the primary intended use of a given system, operating-system designers tend to maximize the criteria most important in a given environment. For example, throughput & component utilization are the primary design objectives in a batch system. Multi-user systems are dominated by concerns regarding the terminal response time, & real-time operating systems are designed for the ability to handle burst of external events responsively.

4.5 Scheduling Algorithms

The scheduling mechanisms described in this section may, at least in theory, be used by any of the three types of schedulers. As pointed out earlier, some algorithms are better suited to the needs of a particular type of scheduler. Depending on whether a particular scheduling discipline is primarily used by the long-term or by the short-term scheduler, we illustrate its working by using the term job or process for a unit of work, respectively.

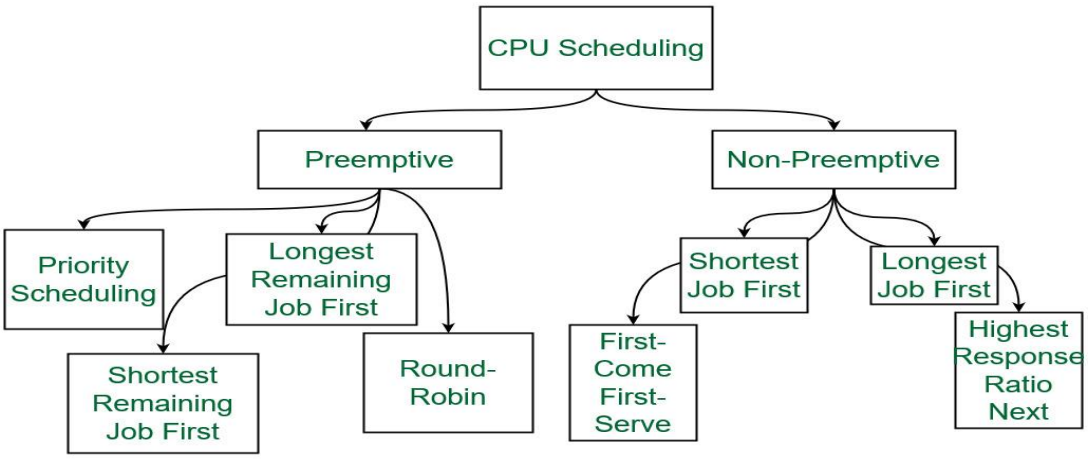
The scheduling policies may be categorized as preemptive & non-preemptive. So it is important to distinguish preemptive from non-preemptive scheduling algorithms. Preemption means the operating system moves a process from running to ready without the process requesting it. Without preemption, the system implements "run to completion". Preemption needs a clock interrupt (or equivalent). Preemption is needed to guarantee fairness & it is found in all modern general-purpose operating systems.

Non-pre-emptive: In non-preemptive scheduling, once a process is executing, it will continue to execute until

- It terminates, or
- It makes an I/O request which would block the process, or
- It makes an operating system call.

Pre-emptive: In the preemptive scheduling,the same three conditions as above apply, & in addition the process may be pre-empted by the operating system when

- A new process arrives (perhaps at a higher priority), or
- An interrupt or signal occurs, or
- A (frequent) clock interrupt occurs.



CPU Scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU. Following are some scheduling algorithms we will study: FCFS Scheduling, Round Robin Scheduling, SJF Scheduling, SRTN Scheduling, Priority Scheduling, Multilevel Queue Scheduling, & Multilevel Feedback Queue Scheduling.

4.5.1 First Come First Served(FCFS)Scheduling

The simplest selection function is the First-Come-First-Served (FCFS) scheduling policy. In it

1. The operating system kernel maintains all Ready processes in a single queue,
2. The process at the head of the queue is always selected to execute next,
3. The Running process runs to completion, unless it requests blocking I/O,
4. If the Running process blocks, it is placed at the end of the Ready queue.

Clearly, once a process commences execution, it will run as fast as possible (having 100% of the CPU, & being non-pre-emptive), but there are some obvious problems. By failing to take into consideration the state of the system & the resource requirements of the individual scheduling entities, FCFS scheduling may result in poor performance. As a consequence of no preemption, component utilization & the system throughput rate may be quite low.

Processes of short duration suffer when “stuck” behind very long-running processes. Since there is no discrimination on the basis of the required service, short jobs may suffer considerable turnaround delays & waiting times when one or more long jobs are in the system. For example, consider a system with two jobs, J1 & J2, with total execution times of 20 & 2 time units, respectively. If they arrive shortly one

after the other in the order J1-J2, the turnaround times are 20 & 22 time units, respectively (J2 must wait for J1 to complete), thus yielding an average of 21 time units. The corresponding waiting times are 0 & 20 unit, yielding an average of 10 time units. However, when the same two jobs arrive in the opposite order, J2-J1, the average turnaround time drops to 11, & the average waiting time is only 1 time unit.

Compute-bound processes are favored over I/O-bound processes.

We can measure the effect of FCFS by examining:

- The average turnaround time of each task (the sum of its waiting & running times), or
- The normalized turnaround time (the ratio of running to waiting times).

4.5.2 Shortest Job Scheduling

In this scheduling policy, the jobs are sorted on the basis of total execution time needed & then it run the shortest job first. It is a non-preemptive scheduling policy. Now First consider a static situation where all jobs are available in the beginning, & we know how long each one takes to run, & we implement “run-to-completion" (i.e., we don't even switch to another process on I/O). In this situation, SJF has the shortest average waiting time. Assume you have a schedule with a long job right before a short job. Now if we swap the two jobs, this decreases the wait for the short by the length of the long job & increases the wait of the long job by the length of the short job. & this in turn decreases the total waiting time for these two. Hence decreases the total waiting for all jobs & hence decreases the average waiting time as well. So in this policy whenever a long job is right before a short job, we swap them & decrease the average waiting time. Thus the lowest average waiting time occurs when there are no short jobs rights before long jobs. This is an example of priority scheduling. This scheduling policy can starve processes that require a long burst.

4.5.3 Shortest Remaining Time Next Scheduling

Shortest remaining time next is a scheduling discipline in which the next scheduling entity, a job or a process, is selected on the basis of the shortest remaining execution time. SRTN scheduling may be implemented in either the non-preemptive or the preemptive variety. The non-preemptive version of SRTN is called shortest job first (SJF). In either case, whenever the SRTN scheduler is invoked, it searches the corresponding queue (batch or ready) to find the job or the process with the shortest remaining execution time. The difference between the two cases lies in the conditions that lead to

invocation of the scheduler and, consequently, the frequency of its execution. Without preemption, the SRTN scheduler is invoked whenever a job is completed or the running process surrenders control to the OS. In the preemptive version, whenever an event occurs that makes a new process ready, the scheduler is invoked to compare the remaining processor execution time of the running process with the time needed to complete the next processor burst of the newcomer. Depending on the outcome, the running process may continue, or it may be preempted & replaced by the shortest-remaining-time process. If preempted, the running process joins the ready queue.

SRTN is a provably optimal scheduling discipline in terms of minimizing the average waiting time of a given workload. SRTN scheduling is done in a consistent & predictable manner, with a bias towards short jobs. With the addition of preemption, an SRTN scheduler can accommodate short jobs that arrive after commencement of a long job. Preferred treatment of short jobs in SRTN tends to result in increased waiting times of long jobs in comparison with FCFS scheduling, but this is usually acceptable.

The SRTN discipline schedules optimally assuming that the exact future execution times of jobs or processes are known at the time of scheduling. In the case of short-term scheduling & preemption's, even more detailed knowledge of the duration of each individual processor burst is required. Dependence on future knowledge tends to limit the effectiveness of SRTN implementations in practice, because future process behavior is unknown in general & difficult to estimate reliably, except for some very specialized deterministic cases.

Predictions of process execution requirements are usually based on observed past behavior, perhaps coupled with some other knowledge of the nature of the process & its long-term statistical properties, if available. A relatively simple predictor, called the exponential smoothing predictor, has the following form:

$$P_n = \alpha 0_n\text{-}1 + (1 - \alpha)P\text{-}1$$

where 0_n is the observed length of the (n-1)th execution interval, P_n-1 is the predictor for the same interval, &α is a number between 0 & 1. The parameter α controls the relative weight assigned to the past observations & predictions. For the extreme case of $\alpha = 1$, the past predictor is ignored, & the new

prediction equals the last observation. For $\alpha = 0$, the last observation is ignored. In general, expansion of the recursive relationship yields

$$P_n = \alpha \sum_{I=0}^{n-1} (1 - \alpha)^i 0_{n-i-1}$$

Thus the predictor includes the entire process history, with its more recent history weighted more.

Many operating systems measure & record elapsed execution time of a process in its PCB. This information is used for scheduling & accounting purposes. Implementation of SRTN scheduling obviously requires rather precise measurement & imposes the overhead of predictor calculation at run time. Moreover, some additional feedback mechanism is usually necessary for corrections when the predictor is grossly incorrect.

SRTN scheduling has important theoretical implications, & it can serve as a yardstick for assessing performance of other, realizable scheduling disciplines in terms of their deviation from the optimum. Its practical application depends on the accuracy of prediction of the job & process behavior, with increased accuracy calling for more sophisticated methods & thus resulting in greater overhead. The preemptive variety of SRTN incurs the additional overhead of frequent process switching & scheduler invocation to examine each & every process transition into the ready state. This work is wasted when the new ready process has a longer remaining execution time than the running process.

4.5.4 Round Robin Scheduling

In interactive environments, such as time-sharing systems, the primary requirement is to provide reasonably good response time and, in general, to share system resources equitably among all users. Obviously, only preemptive disciplines may be considered in such environments, & one of the most popular is time slicing, also known as round robin (RR).

It is a preemptive scheduling policy. This scheduling policy gives each process a slice of time (i.e., one quantum) before being preempted. As each process becomes ready, it joins the ready queue. A clock interrupt is generated at periodic intervals. When the interrupt occurs, the currently running process is preempted, & the oldest process in the ready queue is selected to run next. The time interval between each interrupt may vary.

It is one of the most common & most important scheduler. This is not the simplest scheduler, but it is the simplest *preemptive* scheduler. It works as follows:

- The processes that are ready to run (i.e. not blocked) are kept in a FIFO queue, called the "Ready" queue.
- There is a fixed time quantum (50 msec is a typical number) which is the maximum length that any process runs at a time.
- The currently active process P runs until one of two things happens:
- P blocks (e.g. waiting for input). In that case, P is taken off the ready queue; it is in the "blocked" state.
- P exhausts its time quantum. In this case, P is pre-empted, even though it is still able to run. It is put at the end of the ready queue.
    - In either case, the process at the head of the ready queue is now made the active process.
- When a process unblocks (e.g. the input it's waiting for is complete) it is put at the end of the ready queue.

Suppose the time quantum is 50 msec, process P is executing, & it blocks after 20 msec. When it unblocks, & gets through the ready queue, it gets the standard 50 msec again; it doesn't somehow "save" the 30 msec that it missed last time.

It is an important *preemptive scheduling* policy. It is essentially the preemptive version of FCFS. The key parameter here is the **quantum** size q. When a process is put into the running state a timer is set to q. If the timer goes off & the process is still running, the OS **preempts** the process. This process is moved to the ready state where it is placed at the rear of the ready queue. The process at the front of the ready list is removed from the ready list & run (i.e., moves to state running). When a process is created, it is placed at the rear of the ready list. As q gets large, RR approaches FCFS. As q gets small, RR approaches PS (Processor Sharing).

What value of q should we choose? Actually it is a tradeoff (1) Small q makes system more responsive, (2) Large q makes system more efficient since less process switching.

Round robin scheduling achieves equitable sharing of system resources. Short processes may be executed within a single time quantum & thus exhibit good response times. Long processes may require

several quanta & thus be forced to cycle through the ready queue a few times before completion. With RR scheduling, response time of long processes is directly proportional to their resource requirements. For long processes that consist of a number of interactive sequences with the user, primarily the response time between the two consecutive interactions matters. If the computational requirements between two such sequences may be completed within a single time slice, the user should experience good response time. RR tends to subject long processes without interactive sequences to relatively long turnaround & waiting times. Such processes, however, may best be run in the batch mode, & it might even be desirable to discourage users from submitting them to the interactive scheduler.

Implementation of round robin scheduling requires support of an interval timer-preferably a dedicated one, as opposed to sharing the system time base. The timer is usually set to interrupt the operating system whenever a time slice expires & thus force the scheduler to be invoked. The scheduler itself simply stores the context of the running process, moves it to the end of the ready queue, & dispatches the process at the head of the ready queue. The scheduler is also invoked to dispatch a new process whenever the running process surrenders control to the operating system before expiration of its time quantum, say, by requesting I/O. The interval timer is usually reset at that point, in order to provide the full time slot to the new running process. The frequent setting & resetting of a dedicated interval timer makes hardware support desirable in systems that use time slicing.

Round robin scheduling is often regarded as a "fair" scheduling discipline. It is also one of the best-known scheduling disciplines for achieving good & relatively evenly distributed terminal response time. The performance of round robin scheduling is very sensitive to the choice of the time slice. For this reason, duration of the time slice is often made user-tunable by means of the system generation process.

The relationship between the time slice & performance is markedly nonlinear. Reduction of the time slice should not be carried too far in anticipation of better response time. Too short a time slice may result in significant overhead due to the frequent timer interrupts & process switches. On the other hand, too long a time slice reduces the preemption overhead but increases response time.

Too short a time slice results in excessive overhead, & too long a time slice degenerates from round-robin to FCFS scheduling, as processes surrender control to the OS rather than being preempted by the interval timer. The "optimal" value of the time slice lies somewhere in between, but it is both system-dependent & workload-dependent. For example, the best value of time slice for our example may not

turn out to be so good when other processes with different behavior are introduced in the system, that is, when characteristics of the workload change. This, unfortunately, is commonly the case with time-sharing systems where different types of programs may be submitted at different times.

In summary, round robin is primarily used in time-sharing & multi-user systems where terminal response time is important. Round robin scheduling generally discriminates against long non-interactive jobs & depends on the judicious choice of time slice for adequate performance. Duration of a time slice is a tunable system parameter that may be changed during system generation.

Variants of Round Robin

State dependent RR

It is same as RR but q is varied dynamically depending on the state of the system. It favors processes holding important resources. For example, non-swappable memory.

External priorities

In it a user can pay more & get bigger q. That is one process can be given a higher priority than another. But this is not an absolute priority, i.e., the lower priority (i.e., less important) process does get to run, but not as much as the high priority process.

4.5.5 Priority –Based Preemptive Scheduling (Event Driven, ED)

In it each job is assigned a priority (externally, perhaps by charging more for higher priority) & the highest priority ready job is run. In this policy, If many processes have the highest priority, it uses RR among them. In principle, each process in the system is assigned a priority level, & the scheduler always chooses the highest-priority ready process. Priorities may be static or dynamic. In either case, the user or the system assigns their initial values at the process-creating time. The level of priority may be determined as an aggregate figure on the basis of an initial value, characteristic, resource requirements, & run-time behavior of the process. In this sense, many scheduling disciplines may be regarded as being priority-driven, where the priority of a process represents its likelihood of being scheduled next. Priority-based scheduling may be preemptive or non-preemptive.

A common problem with priority-based scheduling is the possibility that low-priority processes may be effectively locked out by the higher priority ones. In general, completion of a process within finite time of its creation cannot be guaranteed with this scheduling policy. In systems where such uncertainty

cannot be tolerated, the usually remedy is provided by the aging priority, in which the priority of each process is gradually increased after the process spends a certain amount of time in the system. Eventually, the older processes attain high priority & are ensured of completion in finite time.

By means of assigning priorities to processes, system programmers can influence the order in which an ED scheduler services coincident external events. However, the high-priority ones may starve low-priority processes. Since it gives little consideration to resource requirements of processes, event-driven scheduling cannot be expected to excel in general-purpose systems, such as university computing centers, where a large number of user processes are run at the same (default) level of priority.

Another variant of priority-based scheduling is used in the so-called hard real-time systems, where each process must be guaranteed execution before expiration of its deadline. In such systems, time-critical processes are assumed to be assigned execution deadlines. The system workload consists of a combination of periodic processes, executed cyclically with a known period, & of periodic processes, executed cyclically with a known period, & of a periodic processes whose arrival times are generally not predictable. An optimal scheduling discipline in such environments is the earliest-deadline scheduler, which schedules for execution the ready process with the earliest deadline. Another form of scheduler, called the least laxity scheduler or the least slack scheduler, has also been shown to be optimal in single-processor systems. This scheduler selects the ready process with the least difference between its deadline & computation time. Interestingly, neither of these schedulers is optimal in multiprocessor environments.

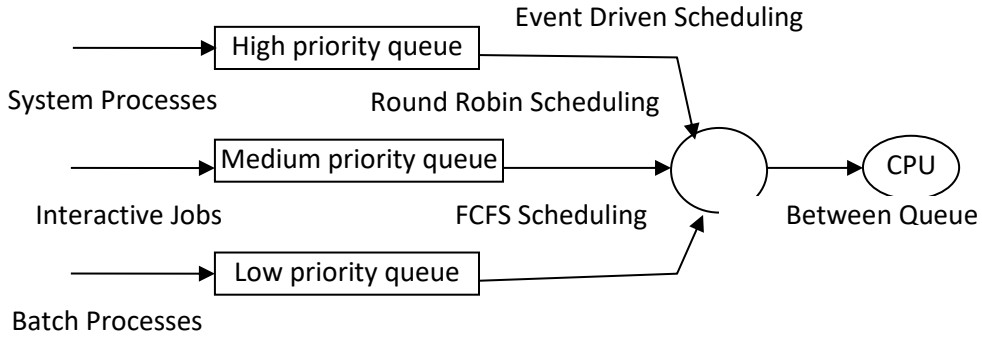
Priority aging

It is a solution to the problem of starvation. As a job is waiting, raise its priority so eventually it will have the maximum priority. This prevents starvation. It is preemptive policy. If there are many processes with the maximum priority, it uses FCFS among those with max priority (risks starvation if a job doesn't terminate) or can use RR.

4.5.6 Multiple Level Queues (MLQ)Scheduling

The scheduling policies discussed so far are more or less suited to particular applications, with potentially poor performance when applied inappropriately. What should one use in a mixed system, with some time-critical events, a multitude of interactive users, & some very long non-interactive jobs? One approach is to combine several scheduling disciplines. A mix of scheduling disciplines may best

service a mixed environment, each charged with what it does best. For example, operating-system processes & device interrupts may be subjected to event-driven scheduling, interactive programs to round robin scheduling, & batch jobs to FCFS or STRN.



Multilevel Queue Scheduling

One way to implement complex scheduling is to classify the workload according to its characteristics, & to maintain separate process queues serviced by different schedulers. This approach is often called multiple-level queues (MLQ) scheduling. A division of the workload might be into system processes, interactive programs, & batch jobs. This would result in three ready queues, as depicted in above Figure. A process may be assigned to a specific queue on the basis of its attributes, which may be user- or system-supplied. Each queue may then be serviced by the scheduling discipline best suited to the type of workload that it contains. Given a single server, some discipline must also be devised for scheduling between queues. Typical approaches are to use absolute priority or time slicing with some bias reflecting relative priority of the processes within specific queues. In the absolute priority case, the processes from the highest-priority queue (e.g. system processes) are serviced until that queue becomes empty. The scheduling discipline may be event-driven, although FCFS should not be ruled out given its low overhead & the similar characteristics of processes in that queue. When the highest-priority queue becomes empty, the next queue may be serviced using its own scheduling discipline (e.g., RR for interactive processes). Finally, when both higher-priority queues become empty, a batch-spawned process may be selected. A lower-priority process may, of course, be preempted by a higher-priority arrival in one of the upper-level queues. This discipline maintains responsiveness to external events &

interrupts at the expense of frequent preemption's. An alternative approach is to assign a certain percentage of the processor time to each queue, commensurate with its priority.

Multiple queues scheduling is a very general discipline that combines the advantages of the "pure" mechanisms discussed earlier. MLQ scheduling may also impose the combined overhead of its constituent scheduling disciplines. However, assigning classes of processes that a particular discipline handles poorly by itself to a more appropriate queue may offset the worst-case behavior of each individual discipline. Potential advantages of MLQ were recognized early on by the O/S designers who have employed it in the so-called fore-ground/background (F/B) system. An F/B system, in its usual form, uses a two-level queue-scheduling discipline. The workload of the system is divided into two queues-a high-priority queue of interactive & time-critical processes & other processes that do not service external events. The foreground queue is serviced in the event-driven manner, & it can preempt processes executing in the background.

4.5.7 Multiple –Level Queues with Feedback Scheduling

Multiple queues in a system may be used to increase the effectiveness & adaptive ness of scheduling in the form of multiple-level queues with feedback. Rather than having fixed classes of processes allocated to specific queues, the idea is to make traversal of a process through the system dependent on its run-time behavior. For example, each process may start at the top-level queue. If the process is completed within a given time slice, it departs the system after having received the royal treatment. Processes that need more than one time slice may be reassigned by the operating system to a lower-priority queue, which gets a lower percentage of the processor time. If the process is still now finished after having run a few times in that queue, it may be moved to yet another, lower-level queue. The idea is to give preferential treatment to short processes & have the resource-consuming ones slowly "sink" into lower-level queues, to be used as fillers to keep the processor utilization high. This philosophy is supported by program-behavior research findings suggesting that completion rate has a tendency to decrease with attained service. In other words, the more service a process receives, the less likely it is to complete if given a little more service. Thus the feedback in MLQ mechanisms tends to rank the processes dynamically according to the observed amount of attained service, with a preference for those that have received less.

On the other hand, if a process surrenders control to the OS before its time slice expires, being moved up in the hierarchy of queues may reward it. As before, different queues may be serviced using different scheduling discipline. In contrast to the ordinary multiple-level queues, the introduction of feedback makes scheduling adaptive & responsive to the actual, measured run-time behavior of processes, as opposed to the fixed classification that may be defeated by incorrect guessing or abuse of authority. A multiple-level queue with feedback is the most general scheduling discipline that may incorporate any or all of the simple scheduling strategies discussed earlier. Its overhead may also combine the elements of each constituent scheduler, in addition to the overhead imposed by the global queue manipulation & the process-behavior monitoring necessary to implement this scheduling discipline.

4.6 CHECK YOUR PROGRESS

10. Which module gives control of the CPU to the process selected by the short-term scheduler?
    a) Dispatcher
    b) Interrupt
    c) Scheduler
    d) None of the mentioned
11. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called _____
    e) job queue
    f) ready queue
    g) execution queue
    h) process queue
12. The interval from the time of submission of a process to the time of completion is termed as _____
    a) waiting time
    b) turnaround time
    c) response time
    d) throughput
13. Which scheduling algorithm allocates the CPU first to the process that requests the CPU first?
    a) first-come, first-served scheduling

b) shortest job scheduling

c) priority scheduling

d) none of the mentioned

14. In priority scheduling algorithm _____

a) CPU is allocated to the process with highest priority

b) CPU is allocated to the process with lowest priority

c) Equal priority processes cannot be scheduled

d) None of the mentioned

15. Which algorithm is defined in Time quantum?

a) shortest job scheduling algorithm

b) round robin scheduling algorithm

c) priority scheduling algorithm

d) multilevel queue scheduling algorithm

16. Process are classified into different groups in _____

a) shortest job scheduling algorithm

b) round robin scheduling algorithm

c) priority scheduling algorithm

d) multilevel queue scheduling algorithm

17. In multilevel feedback scheduling algorithm _____

a) a process can move to a different classified ready queue

b) classification of ready queue is permanent

c) processes are not classified into groups

d) none of the mentioned

18. Which one of the following cannot be scheduled by the kernel?

a) kernel level thread

b) user level thread

c) process

d) none of the mentioned

4.7 SUMMARY

An important, although rarely explicit, function of process management is processor allocation. Three different schedulers may coexist & interact in a complex operating system: long-term scheduler, medium-term scheduler, & short-term scheduler. Of the presented scheduling disciplines, FCFS scheduling is the easiest to implement but is a poor performer. SRTN scheduling is optimal but unrealizable. RR scheduling is most popular in time-sharing environments, & event-driven & earliest-deadline scheduling are dominant in real-time & other systems with time-critical requirements. Multiple-level queue scheduling, & its adaptive variant with feedback, is the most general scheduling discipline suitable for complex environments that serve a mixture of processes with different characteristics.

4.8 KEYWORDS

Short-term scheduling: the decisions as to which (Ready) process to execute next.

Non-preemptive scheduling: In non-preemptive scheduling, process will continue to execute until it terminates, or makes an I/O request which would block the process, or makes an operating system call.

In preemptive scheduling, the process may be pre-empted by the operating system when a new process arrives (perhaps at a higher priority), or an interrupt or signal occurs, or a (frequent) clock interrupt occurs.

4.9 Self-Assesment Questions (SAQ)

8. Which type of scheduling is used in real life operating systems? Why?
9. Which action should the short-term scheduler take when it is invoked but no process is in the ready state? Is this situation possible?
10. How can we compare performance of various scheduling policies before actually implementing them in an operating system?
11. SJF is a sort of priority scheduling. Comment.
12. What do you understand by starvation? How does SJF cause starvation? What is the solution of this problem?
13. What qualities are to be there in a scheduling policy? Explain.

14. Differentiate between user-oriented scheduling criteria & system-oriented scheduling criteria.

4.10 ANSWER TO CHECK YOUR PROGRESS

10. Dispatcher
11. Ready queue
12. Turnaround time
13. First-come, first-served scheduling
14. CPU is allocated to the process with highest priority
15. Round robin scheduling algorithm
16. Multilevel queue scheduling algorithm
17. A process can move to a different classified ready queue
18. User level thread

4.11 Reference and Suggested Readings

6. Operating System Concepts, 5th Edition, Silberschatz A., Galvin P.B., John Wiley & Sons.
7. Systems Programming & Operating Systems, 2nd Revised Edition, Dhamdhere D.M., Tata McGraw Hill Publishing Company Ltd., New Delhi.
8. Operating Systems, Madnick S.E., Donovan J.T., Tata McGraw Hill Publishing Company Ltd., New Delhi.
9. Operating Systems-A Modern Perspective, Gary Nutt, Pearson Education Asia, 2000.
10. Operating Systems, Harris J.A., Tata McGraw Hill Publishing Company Ltd., New Delhi, 2002.

Course: MCA-32	Writer: Ritu
Lesson: 5	Vetter:

File System – I

Structure

5.0 Learning Objectives

A file is a logical collection of information and file system is a collection of files. The objective of this lesson is to:

- Discuss the various concepts of file system and make the students familiar with the different techniques of file allocation and access methods.
- Discuss the ways to handle file protection, which is necessary in an environment where multiple users have access to files and where it is usually desirable to control by whom and in what ways files may be accessed.

5.1 Introduction

The file system is the most visible aspect of an operating system. While the memory

manager is responsible for the maintenance of primary memory, the file manager is responsible for the maintenance of secondary storage (e.g., hard disks). It provides the mechanism for on-line storage of and access to both data and programs of the operating system and all the users of the computer system. The file system consists of two distinct parts: a collection of files, each storing related data and a directory structure, which organizes and provides information about all the files in the system. Some file systems have a third part, partitions, which are used to separate physically or logically large collections of directories.

Nutt describes the responsibility of the file manager and defines the file, the fundamental abstraction of secondary storage:

"Each file is a named collection of data stored in a device. The file manager implements this abstraction and provides directories for organizing files. It also provides a spectrum of commands to read and write the contents of a file, to set the file read/write position, to set and use the protection mechanism, to change the ownership, to list files in a directory, and to remove a file...The file manager provides a protection mechanism to allow machine users to administer how processes executing on behalf of different users can access the information in files. File protection is a fundamental property of files because it allows different people to store their information on a shared computer, with the confidence that the information can be kept confidential."

The most important function of an operating system is the effective management of information. The modules of the operating system dealing with the management of information are known as file system. The file system provides the mechanism for online storage and access to both data and programs. The file system resides permanently on secondary storage, which has the main requirement that it must be able to hold a large amount of data, permanently. The desirable features of a file system are:

a) Minimal I/O operations.

b) Flexible file naming facilities.

c) Automatic allocation of file space.

d) Dynamic allocation of file space.

e) Unrestricted flexibility between logical record size and physical block size.

f) Protection of files against illegal forms of access.

g) Static and dynamic sharing of files.

h) Reliable storage of files.

This lesson is primarily concerned with issues concerning file storage and access on the most common secondary storage medium, the disk.

A file is a collection of related information units (records) treated as a unit. A record is itself a collection of related data elements (fields) treated as a unit. A field contains a single data item.

So file processing refers to reading/writing of records in a file and processing of the information in the fields of a record.

5.2.1 File operations

File operations are generally simple and intuitive set of operations on files. Not all of these are supported by all file systems. In some file systems, operations are implemented

in terms of other file operations. Major file operations performed are as follows:

1. Create operation:

This operation is used to create a file in the file system. It is the most widely used operation performed on the file system. To create a new file of a particular type the associated application program calls the file system. This file system allocates space to the file. As the file system knows the format of directory structure, so entry of this new file is made into the appropriate directory.

2. Open operation:

This operation is the common operation performed on the file. Once the file is created, it must be opened before performing the file processing operations. When the user wants to open a file, it provides a file name to open the particular file in the file system. It tells the operating system to invoke the open system call and passes the file name to the file system.

3. Write operation:

This operation is used to write the information into a file. A system call write is issued that specifies the name of the file and the length of the data has to be written to the file. Whenever the file length is increased by specified value and the file pointer is repositioned after the last byte written.

4. Read operation:

This operation reads the contents from a file. A Read pointer is maintained by the OS, pointing to the position up to which the data has been read.

5. Re-position or Seek operation:

The seek system call re-positions the file pointers from the current position to a specific place in the file i.e. forward or backward depending upon the user's requirement. This operation is generally performed with those file management systems that support direct access files.

6. Delete operation:

Deleting the file will not only delete all the data stored inside the file it is also used so that disk space occupied by it is freed. In order to delete the specified file the directory is searched. When the directory entry is located, all the associated file space and the directory entry is released.

7. Truncate operation:

Truncating is simply deleting the file except deleting attributes. The file is not completely deleted although the information stored inside the file gets replaced.

8. Close operation:

When the processing of the file is complete, it should be closed so that all the changes made permanent and all the resources occupied should be released. On closing it deallocates all the internal descriptors that were created when the file was opened.

9. Append operation:

This operation adds data to the end of the file.

10. Rename operation:

This operation is used to rename the existing file

5.2.2 File Naming

Each file is a distinct entity and therefore a naming convention is required to distinguish one from another. The operating systems generally employ a naming system for this purpose. In fact, there is a naming convention to identify each resource in the computer system and not files alone.

5.2.3 File Types

A common technique for implementing file types is to include the type as part of the file name. The name is split into two parts—a name and an *extension*, usually separated by a period character figure --.In this way, the user and the operating system can tell from the name alone what the type of a file is. For example, most operating systems allow users to specify a file name as a sequence of characters followed by a period and terminated by an extension of additional characters. File name examples include *resume.doc, Server.java etc.* The system uses the extension to indicate the type of the file and the type of operations that can be done on that file. Only a file with a *.com*, *.exe*, or *.bat* extension can

be *executed*, for instance. The *.com* and *.exe* files are two forms of binary executable files, whereas a *.bat* file is a **batch file** containing, in ASCII format, commands to the operating system

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Figure: File Types

The files under UNIX can be categorized as follows:

The files under UNIX can be categorized as follows:

- Ordinary files.
- Directory files.
- Special files
- FIFO files.

Ordinary files

Ordinary files are the one, with which we all are familiar. They may contain executable programs, text or databases. You can add, modify or delete them or remove the file entirely.

Directory Files

Directory files, as discussed earlier also represent a group of files. They contain list of file names and other information related to these files. Some of the commands, which manipulate these directory files, differ from those for ordinary files.

Special Files

Special files are also referred to as device files. These files represent physical devices such as terminals, disks, printers and tape-drives etc. These files are read from or written into just like ordinary files, except that operation on these files activates some physical devices. These files can be of two types (i) character device files and (ii) block device file. In character device files data are handled character by character, as in case of terminals and printers. In block device files, data are handled in large chunks of blocks, as in the case of disks and tapes.

FIFO Files

FIFO (first-in-first-out) are files that allow unrelated processes to communicate with each other. They are generally used in applications where the communication path is in only one direction, and several processes need to communicate with a single process. For an example of FIFO file, take the pipe in UNIX. This allows transfer of data between processes in a first- in-first-out manner. A pipe takes the output of the first process as the input to the next process, and so on.

5.2.4 Symbolic Link

A link is effectively a pointer or an alias to another file or subdirectory. For example, a link may be implemented as an absolute or relative path name (a symbolic link). When a reference to a file is made, we search the directory. The directory entry is marked as a link and the name of the real file (or directory) is given. We resolve the link by using the path name to locate the real file. Links are easily identified by their format in the directory entry (or by their having a special type on systems that support types), and are effectively named indirect pointers.

A symbolic link can be deleted without deleting the actual file it links. There can be any number of symbolic links attached to a single file.

Symbolic links are helpful in sharing a single file called by different names. Each time a link is created, the reference count in its inode is incremented by one. Whereas deletion of link decreases the reference count by one. The operating system denies deletion of such files whose reference count is not 0, thereby meaning that the file is in use.

In a system where sharing is implemented by symbolic links, this situation is somewhat easier to handle. The deletion of a link does not need to affect the original file; only the link is removed. If the file entry itself is deleted, the space for the file is deallocated, leaving the links dangling. We

can search for these links and remove them also, but unless a list of the associated link is kept with each file, this search can be expensive. Alternatively, we can leave the links until an attempt is made to use them. At that time, we can determine that the file of the name given by the link does not exist, and can fail to resolve the link name; the access is treated just like any other illegal file name. (In this case, the system designer should consider carefully what to do when a file is deleted and another file of the same name is created, before a symbolic link to the original file is used.) In the case of UNIX, symbolic links are left when a file is deleted, and it is up to the user to realize that the original file is gone or has been replaced.

Another approach to deletion is to preserve the file until all references to it are deleted. To implement this approach, we must have some mechanism for determining that the last reference to the file has been deleted. We could keep a list of all references to a file (directory entries or symbolic links). When a link or a copy of the directory entry is established, a new entry is added to the file-reference list. When a link or directory entry is deleted, we remove its entry on the list. The file is deleted when its file-reference list is empty.

The trouble with this approach is the variable and potentially large size of the file-reference list. However, we really do not need to keep the entire list - we need to keep only a count of the number of references. A new link or directory entry increments the reference counts; deleting a link or entry decrements the count. When the count is 0, the file can be deleted; there are no remaining references to it. The UNIX operating system uses this approach for non-symbolic links, or hard links, keeping a reference count in the file information block or inode). By effectively prohibiting multiple references to directories, we maintain an acyclic- graph structure.

To avoid these problems, some systems do not allow shared directories link. For example, in MS-DOS, the directory structure is a tree structure.

5.2.5 File Sharing and Locking

The owner of a file uses the access control list of the file to authorize some other users to access the file. In a multi-user environment a file is required to be shared among more than one user. There are several techniques and approaches to affect this operation. File sharing can occur in two modes (i) sequential sharing and (ii) concurrent sharing. Sequential sharing occurs when authorized users access a shared file one after another. So any change made by a user is reflected to other users also.

Concurrent sharing occurs when two or more users access a file over the same period of time. Concurrent sharing may be implemented in one of the following three forms:

(a) Concurrent sharing using immutable files: In it any program cannot modify the file being shared.

(b) Concurrent sharing using single image mutable files: An image is a view of a file. All programs concurrently sharing the file see the same image of the file. So changes made by one program are also visible to other programs sharing the file.

(c) Concurrent sharing using multiple image mutable files: Each program accessing the file has its own image of the file. So many versions of the file at a time may exist and updates made by a user may not be visible to some concurrent user.

There are three different modes to share a file:

- Read only: In this mode the user can only read or copy the file.
- Linked shared: In this mode all the users sharing the file can make changes in this file but the changes are reflected in the order determined by the operating systems.
- Exclusive mode: In this mode a single user who can make the changes (while others can only read or copy it) acquires the file.

Another approach is to share a file through symbolic links. This approach poses a couple of problems - concurrent updation problem, deletion problem. If two users try to update the same file, the updating of one of them will be reflected at a time. Besides, another user must not delete a file while it is in use.

File locking gives processes the ability to implement mutually exclusive access to a file. Locking is mechanism through which operating systems ensure that the user making changes to the file is the one who has the lock on the file. As long as the lock remains with this user, no other user can alter the file. Locking can be limited to files as a whole or parts of a file. Locking may apply to any access or different levels of locks may exist such as read/write locks etc.

5.2.6 File-System Structure

Disks provide the bulk of secondary storage on which a file system is maintained. To improve I/O efficiency, I/O transfers between memory and disks are performed in units of blocks. Each block is

one or more sectors. Depending on the disk drive, sectors vary from 32 bytes to 4096 bytes; usually, they are 512 bytes. The blocking method determines how a file's records are allocated into blocks:

Fixed blocking: An integral number of fixed-size records are stored in each block. No record may be larger than a block.

Unspanned blocking: Multiple variable size records can be stored in each block but no record may span multiple blocks.

Spanned blocking: Records may be stored in multiple blocks. There is no limit on the size of a record.

Disks have two important characteristics that make them a convenient medium for storing multiple files:

(a) They can be rewritten in place; it is possible to read a block from the disk, to modify the block, and to write it back into the same place.

(b) One can access directly any given block of information on the disk. Thus, it is simple to access any file either sequentially or randomly, and switching from one file to another added requires only moving the read-write heads and waiting for the disk to rotate.

To provide an efficient and convenient access to the disk, the operating system imposes a file system to allow the data to be stored, located, and retrieved easily. A file system poses two quite different design problems.

(a) How the file system should look to the user? This task involves the definition of a file and its attributes, operations allowed on a file and the directory structure for organizing the files.

(b) Algorithms and data structure must be created to map the logical file system onto the physical secondary storage devices.

5.2.7 File-System Mounting

Just as a file must be opened before it is used, a file system must be mounted before it can be available to processes on the system. The mount procedure is straightforward. The operating system is given the name of the device and the location within the file structure at which to attach the file system (called the mount point). For instance, on the UNIX system, a file system

containing user's home directory might be mounted as /home; then, to access the directory structure within that file system, one could precede the directory names with

/home, as in /home/jane. Mounting that file system under /users would result in the path name /users/jane to reach the same directory.

Next, the operating system verifies that the device contains a valid file system. It does so by asking the device driver to read the device directory and verifying that the directory has the expected format. Finally, the operating system notes its directory structure that a file system is mounted at the specified mount point. This scheme enables the operating system to traverse its directory structure, switching among file systems as appropriate. Consider the actions of the Macintosh Operating System.

Whenever the system encounters a disk for the first time (hard disks are found at boot time, floppy disks ate seen when they are inserted into the drive), the Macintosh Operating System searches for a file system on the device. If it finds one, it automatically mounts the file system at the boot-level, adds a folder icon to the screen labeled with the name of the file system (as stored in the device directory). The user is then able to click on the icon and thus to display the newly mounted file system.

5.2.8 File space allocations

The direct-access nature of disks allows flexibility in the implementation of files. In almost every case, many files will be stored on the same disk. The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly. There are three major methods of allocating disk space:

(a) Contiguous space allocation

(b) Linked allocation

(c) Indexed allocation

Each method has its advantages and disadvantages. Accordingly some systems support all three. More common system will use one particular method for all files.

5.2.8.1 Contiguous space Allocation

The simplest scheme is contiguous allocation. The logical blocks of a file are stored in a partition of contiguous physical blocks. Disk addresses define a linear ordering on the disk.

In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: b, b+1, b+2,……b+n-1. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file. The directory entry for a file with contiguous allocation contains

- Address of starting block
- Length of the allocated portion.

The file 'mail' in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies 19, 20, 21, 22, 23, 24 blocks.

Directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Advantages:

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).

- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

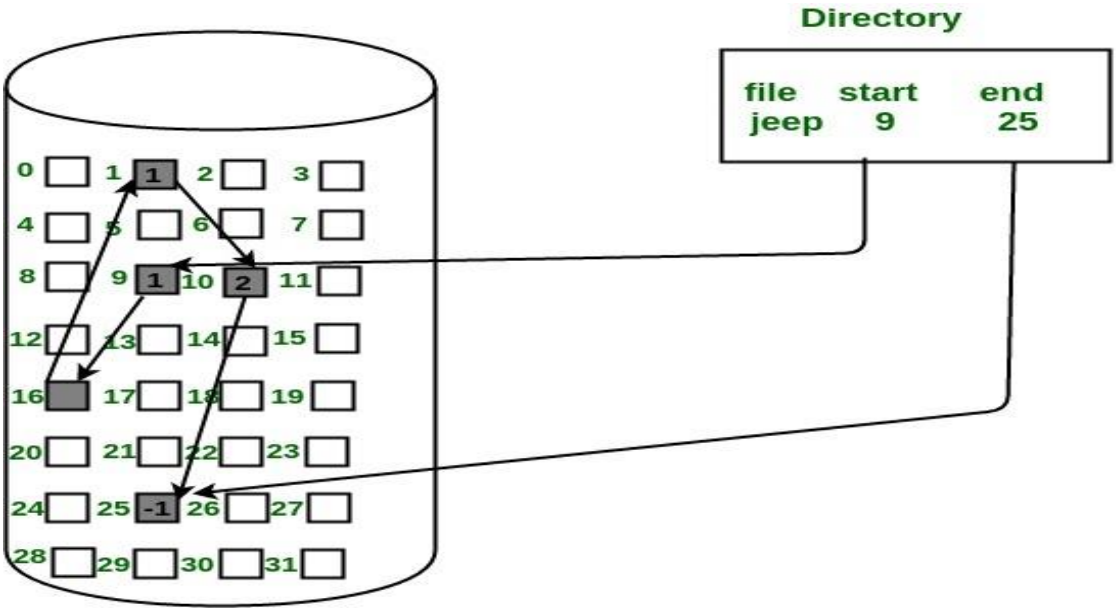
Disadvantages:

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

5.2.8.2 Linked Allocation

In this scheme, each file is a linked list of disk blocks which **need not be** contiguous. The disk blocks can be scattered anywhere on the disk. The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.

The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.



Advantages:

- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.

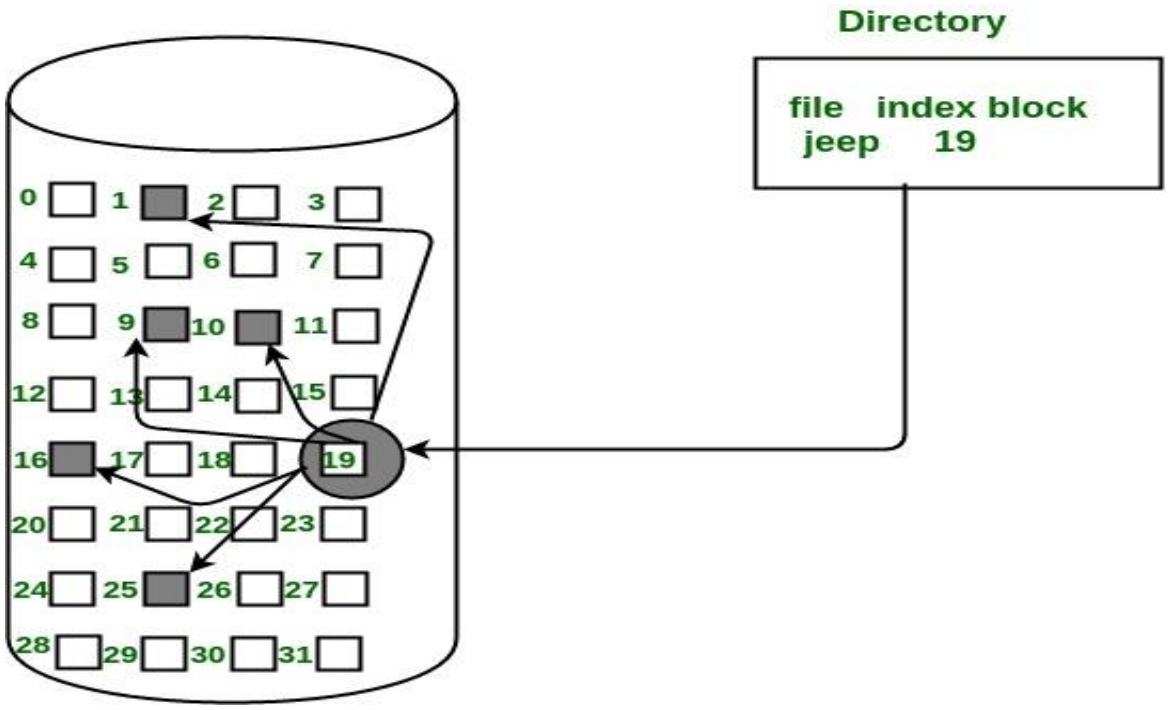
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages:

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We cannot directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

5.2.1.8.3 Indexed Allocation

In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file. Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block. The directory entry contains the address of the index block as shown in the image:



Advantages:

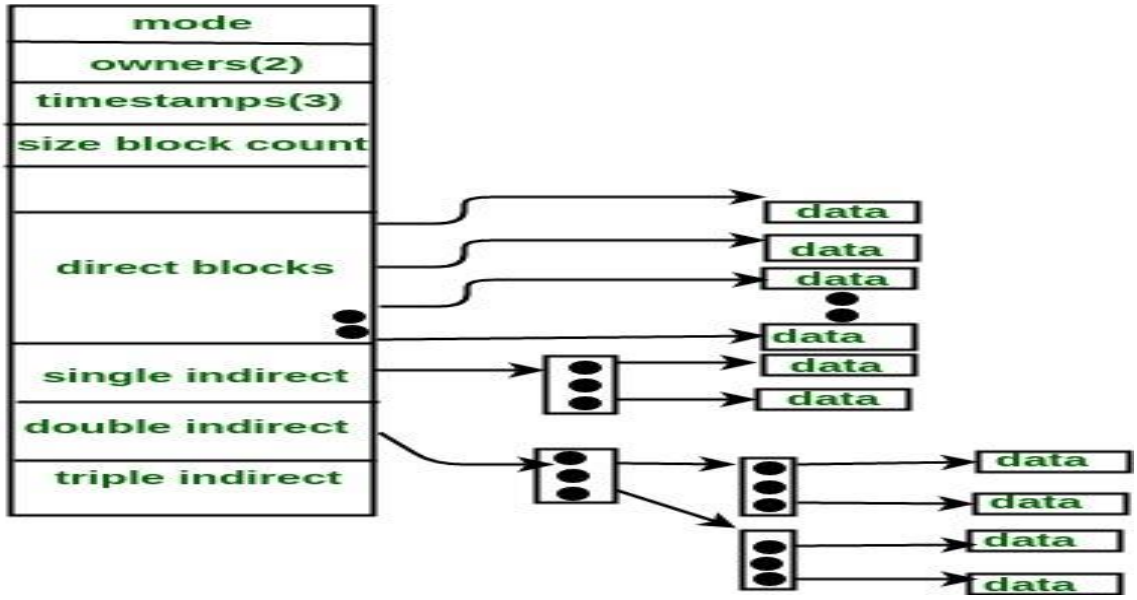
- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages:

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

For files that are very large, single index block may not be able to hold all the pointers. Following mechanisms can be used to resolve this:

1. **Linked scheme:** This scheme links two or more index blocks together for holding the pointers. Every index block would then contain a pointer or the address to the next index block.
2. **Multilevel index:** In this policy, a first level index block is used to point to the second level index blocks which inturn points to the disk blocks occupied by the file. This can be extended to 3 or more levels depending on the maximum file size.
3. **Combined Scheme:** In this scheme, a special block called the **Inode (information Node)** contains all the information about the file such as the name, size, authority, etc and the remaining space of Inode is used to store the Disk Block addresses which contain the actual file *as shown in the image below*. The first few of these pointers in Inode point to the **direct blocks** i.e the pointers contain the addresses of the disk blocks that contain data of the file. The next few pointers point to indirect blocks. Indirect blocks may be single indirect, double indirect or triple indirect. **Single Indirect block** is the disk block that does not contain the file data but the disk address of the blocks that contain the file data. Similarly, **double indirect blocks** do not contain the file data but the disk address of the locks that contain the address of the blocks containing the file data.



Indexed allocation of disk space

Indexed allocation does suffer from wasted space. The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.

Note that indexed allocation schemes suffer from some of the same performance problems, as does linked allocation. Specifically, the index blocks can be cached in memory, but the data blocks may be spread all over a partition.

5.2.1.8.4 Performance

To evaluate the performance of allocation methods, two important criteria are storage

efficiency and data-block access times. Both are important criteria in selecting the proper method or methods for an operating system to implement.

One difficulty in comparing in performance of the various systems is determining how the systems will be used – in a sequential access manner or random access. A system with mostly sequential access should use a method different from that for a system with mostly random access. For any type of access, contiguous allocation requires only one access to get disk block. Since we can easily keep the initial address of the file in memory, we can calculate immediately the disk address of the i^{th} block (or the next block) and read it directly. For linked allocation, we can also keep the address of the next block in memory and read it directly. This method is fine for sequential access; for

direct access, however, an access to the i^{th} block might require i disk reads. This problem indicates why linked allocation should not be used for an application requiring direct access.

As a result, some systems support direct-access files by using contiguous allocation and sequential access by linked allocation. For these systems, the type of access to be made must be declared when the file is created. A file created for sequential access will be linked and cannot be used for direct access. A file created for direct access will be contiguous and can support both direct access and sequential access, but its maximum length must be declared with it.

5.2.9 File attributes

Attributes are properties of a file. The operating system treats a file according to its attributes. Following are a few common attributes of a file:

- H for hidden
- A for archive
- D for directory
- X for executable
- R for read only

These attributes can be used in combination also.

5.3 ACCESS METHODS

Files store information, which is when required, may be read into the main memory. There are several different ways in which the data stored in a file may be accessed for reading and writing. The operating system is responsible for supporting these file access methods. The fundamental methods for accessing information in the file are (a) sequential access: in it information in the file must be accessed in the order it is stored in the file, (b) direct access, and (c) index sequential access.

5.3.1 Sequential access

A sequential file is the most primitive of all file structures. It has no directory and no linking pointers. The records are generally organized in a specific sequence according to the key field. In other words, a particular attribute is chosen whose value will determine the order of the

records. Access proceeds sequentially from start to finish. Operations to read or write the file need not specify the logical location within the file, because operating system maintains a file pointer that determines the location of the next access. Sometimes when the attribute value is constant for a large number of records a second key is chosen to give an order when the first key fails to discriminate. Use of sequential file requires data to be sorted in a desired sequence according to the key field before storing or processing them.

Its main advantages are:

- It is easy to implement
- It provides fast access to the next record if the records are to be accessed using lexicographic order.

Its disadvantages are:

- It is difficult to update and insertion of a new record may require moving a large proportion of the file
- Random access is extremely slow.

Sometimes a file is considered to be sequentially organised despite the fact that it is not ordered according to any key. Perhaps the date of acquisition is considered to be the key value; the newest entries are added to the end of the file and therefore pose no difficulty to updating. Sequential files are advisable if the applications are sequential by nature.

5.3.2 Index-sequential

An index-sequential file each record is supposed to have a unique key and the set of records may be ordered sequentially by a key. An index is maintained to determine the location of a record from its key value. Each key value appears in the index with the associated address of its record. To access a record with key k, the index entry containing k is found by searching the index and the disk address mentioned in the entry is used to access the record.

In the following figure an employee file is illustrated where records are arranged in ascending order according to the employee #.

Track #	

1	1 2 5 8 16 20 25 30 32 36
2	38 40 41 43 44 45 50 52
3	53 57 59 60 62 64 67 70

A track index is maintained as shown in the following figure to speed up the search:

Track	Low	High
1	1	36
2	38	52
3	53	70

For example, to locate the record of employee # 41, index is searched. It is evident from the index that the record of employee #41 will be on track no.2 because it has the lowest key value 48 and highest key value 52.

In the literature an index-sequential file is usually thought of as a sequential file with a hierarchy of indices. For example, there might be three levels of indexing: track, cylinder and master. Each entry in the track index will contain enough information to locate the start of the track, and the key of the last record in the track, which is also normally the highest value on that track. There is a track index for each cylinder. Each entry in the cylinder index gives the last record on each cylinder and the address of the track index for that cylinder. If the cylinder index itself is stored on tracks, then the master index will give the highest key referenced for each track of the cylinder index and the starting address of that track. No mention has been made of the possibility of overflow during an updating process. Normally provision is made in the directory to administer an overflow area. This of course increases the number of book- keeping entries in each entry of the index.

5.3.3 Direct access

In direct access file organization, any records can be accessed irrespective of the current position in the file. Direct access files are created on direct access storage devices. Whenever a record is to be

inserted, its key value is mapped into an address using a hashing function. On that address record is stored. The advantage of direct access file organization is realized when the records are to be accessed randomly (not sequentially). Otherwise this organization has a number of limitations such as (a) poor utilization of the I/O medium and (b) Time consumption during record address calculation.

5.4 Check your progress

1. _____ is a unique tag, usually a number identifies the file within the file system.

a) File identifier

b) File name

c) File type

d) None of the mentioned

2. To create a file _____

a) allocate the space in file system

b) make an entry for new file in directory

c) allocate the space in file system & make an entry for new file in directory

d) none of the mentioned

3. By using the specific system call, we can _____

a) open the file

b) read the file

c) write into the file

d) all of the mentioned

4. File type can be represented by _____

a) file name

b) file extension

c) file identifier

d) none of the mentioned

5. Which file is a sequence of bytes organized into blocks understandable by the system's linker?

a) object file

b) source file

c) executable file

d) text file

6. What is the mounting of file system?

a) crating of a filesystem

b) deleting a filesystem

c) attaching portion of the file system into a directory structure

d) removing the portion of the file system into a directory structure

7. Mapping of file is managed by _____

a) file metadata

b) page table

c) virtual memory

d) file system

8. Mapping of network file system protocol to local file system is done by _____

a) network file system

b) local file system

c) volume manager

d) remote mirror

9. Which one of the following explains the sequential file access method?

a) random access according to the given byte number

b) read bytes one at a time, in order

c) read/write sequentially by record

d) read/write randomly by record

10. When will file system fragmentation occur?

a) unused space or single file are not contiguous

b) used space is not contiguous

c) unused space is non-contiguous

d) multiple files are non-contiguous

5.4 SUMMARY

The file system resides permanently on secondary storage, which has the main requirement that it

must be able to hold a large amount of data, permanently.

The various files can be allocated space on the disk in three ways: through contagious, linked or indexed allocation. Contagious allocation can suffer from external fragmentation. Direct- access is very inefficient with linked-allocation. Indexed allocation may require substantial overhead for its index block. There are many ways in which these algorithms can be optimized.

Free space allocation methods also influence the efficiency of the use of disk space, the performance of the file system and the reliability of secondary storage.

5.6 Key words

Contiguous space Allocation: The logical blocks pf a file are stored in a partition of contiguous physical blocks.

Linked allocation: In it each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file.

Indexed Allocation: In Indexed allocation all the pointers together are stored into one location, called the index block. Each file has its own index block, which is an array of disk- block addresses.

Sequential access: In it information in the file must be accessed in the order it is stored in the file.

Index-sequential: In index-sequential file each record is supposed to have a unique key and the set of records may be ordered sequentially by a key. An index is maintained to determine the location of a record from its key value.

Direct access: In direct access file organization, records can be accessed randomly. The key value of the record is mapped into an address using a hashing function. On that address record is stored.

5.7 SELF-ASSESSMENT QUESTIONS (SAQ)

1. What do you understand by a file? What is a file system?
2. What are the different modes to share a file?
3. What are the different methods to access the information from a file? Discuss their advantages and disadvantages.
4. What are the advantages of indexed allocation over linked allocation and contiguous space

allocation? Explain.

5. Differentiate between first fit, best fit and worst fit storage allocation strategies.

5.8 Answer to check your progress

1. File identifier
2. allocate the space in file system & make an entry for new file in directory
3. all of the mentioned
4. file extension
5. object file
6. attaching portion of the file system into a directory structure
7. file metadata
8. network file system
9. read bytes one at a time, in order
10. unused space or single file are not contiguous

5.9 SUGGESTED READINGS / REFERENCE MATERIAL

1. Operating System Concepts, 5th Edition, Silberschatz A., Galvin P.B., John Wiley and Sons.
2. Systems Programming and Operating Systems, 2nd Revised Edition, Dhamdhere D.M., Tata McGraw Hill Publishing Company Ltd., New Delhi.
3. Operating Systems, Madnick S.E., Donovan J.T., Tata McGraw Hill Publishing Company Ltd., New Delhi.
4. Operating Systems-A Modern Perspective, Gary Nutt, Pearson Education Asia, 2000.
5. Operating Systems, Harris J.A., Tata McGraw Hill Publishing Company Ltd., New Delhi, 2002.

Course: MCA-32	Writer: Ritu
Lesson: 6	Vetter:

File System - II

Structure

6.0 Learning Objectives

The objectives of this lesson are to make the students familiar with directory system and file protection mechanism. After studying this lesson students will become familiar with:

(a) Different types of directory structures.

(b) Different protection structures such as:

- Access Control Matrix
- Access Control Lists

6.1 Introduction

A file system provides the following facilities to its users:

(a) Directory structure and file naming facilities,

(b) Protection of files against illegal form of access,

(c) Static and dynamic sharing of files, and

(d) Reliable storage of files.

A file system helps the user in organizing the files through the use of directories. A directory may be defined as an object that contains the names of the file system objects. Entries in the directory determine the names associated with a file system object. A directory contains information about a group of files. A typical structure of a directory entry is as under:

File name – Locations Information – Protection Information – Flags

The presences of directories enable file system to support file sharing and protection. Sharing is simply a matter of permitting a user to access the files of other user stored in some other directory. Protection is implemented by permitting the owner of a file to specify which other users may access his files and in what manner. All these issues are discussed in detail in this lesson.

6.2 Hierarchical Directory Systems

Files are generally stored on secondary storage devices. Numerous files are to be stored on storage of giga-byte capacity. To handle such a huge size of data, there is a need to properly organize the files. The organization, usually, done in two parts. In the first part, a file system may incorporate the notion of a partition, which determines on which device a file will be stored. The file system is broken into partitions, also known as minidisks or volumes. Typically, a disk contains at least one partition, which is a low-level structure in which files and directories reside. Sometimes, there may be more than one partition on a disk, each partition acting as a virtual disk. The users do not have to concern themselves with the translating the physical address; the system does the required job.

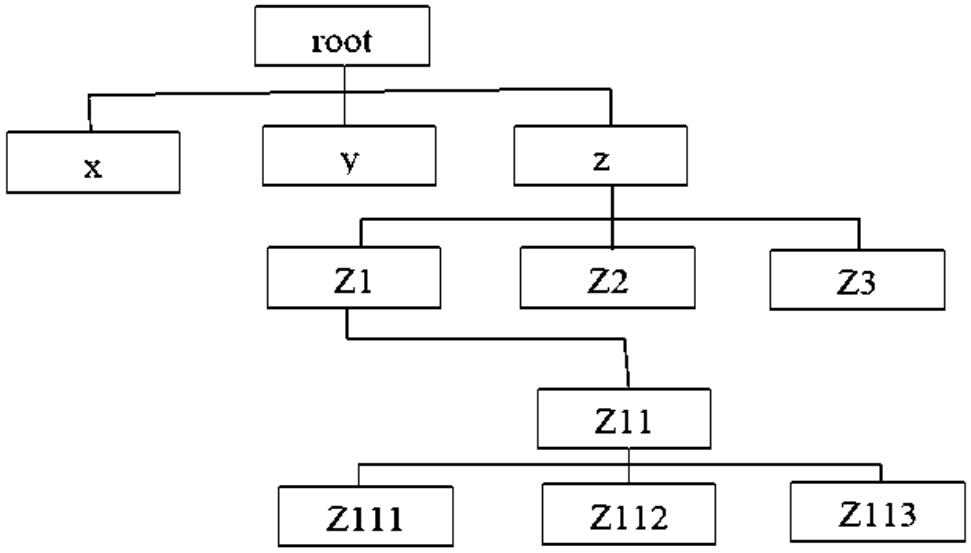


Figure 1 Directory Hierarchy

Partitions contain information about itself in a file called partition table. It also contains information about files and directories on it. Typical file information is name, size, type, location etc. The entries are kept in a device directory or volume table of contents (VTOC).

Directories may be created within another directory. Directories have parent-child relationship as shown in the Figure 1.

6.2.1 Directory Structure

The file systems of computers can be extensive. Some systems store thousands of files on hundreds of gigabytes of disk. To manage all these data, we need to organize them. This organization is usually done in two parts; first, the file system is broken into in the IBM world or volumes in the PC and Macintosh arenas. Sometimes, partitions are used to provide several separate areas within one disk, each treated as a separate storage device, whereas other systems allow partitions to be larger than a disk to group disks into one logical structure. In this way, the user needs to be concerned with only the logical directory and file structure, and can ignore completely the problems of physically allocating space for files. For this reason partitions can be thought of as virtual disks.

Second, each partition contains information about files within it. This information is kept in a device directory or volume table of contents. The device directory (more commonly known simply as a "directory") records information such as name, location, size, and type for all files on that partition.

6.2.2 The logical Structure of a Directory

The simplest directory structure is the single-level tree. A single level tree system has only one directory. All files are contained in the same directory, which is easy to support and understand. Names in that directory refer to files or other non-directory objects. Such a system is practical only on systems with very limited numbers of files. The advantage of this structure is its simplicity only. But it has got many limitations:

Limitations:

A single-level directory is manageable when the number of files is small. When the number of files increases or when there is more than one user, it suffers from the following problems:

(a) Since all files are stored in the same directory, the name given to each file should be unique. If there are two users and they give the same name to their file, then there is a problem.

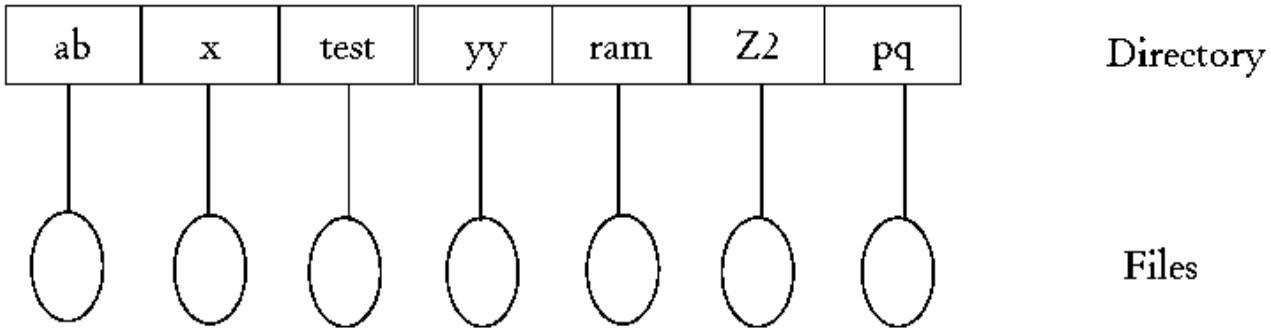


Figure 2 Single-level Directory

(b) Even with a single user, as the number of files increase, it becomes difficult to remember the names of all the files, so as to create only files with unique names. It is not uncommon for a user to have hundreds of files on one computer system and an equal number of additional files on another system. In such an environment, keeping track of so many files is a daunting task.

6.2.2.2 Two-Level Directory

The disadvantage of a single-level directory is confusion of file names. The standard solution is to create a separate directory for each user. In a two level system, only the root level directory may contain names of directories and all other directories refer only to non-directory objects. In the two-level directory structure, each user has his/her own user file directory (UFD). Each UFD has a similar structure, but lists only the files of a single user. When a user starts or a user logs in, the system's master file directory is searched. The master file directory is indexed by user name or account.

When in a UFD a user refers to a particular file, only his own UFD is searched. Thus, different users may have files with the same name, as long as till the filenames within each UFD are unique. To create a file for a user, the operating system searches only that user's UFD to ascertain whether another file of that name exists. To delete a file, the operating system confines its search to the local UFD; thus, it cannot accidentally delete another user's file that has the same name.

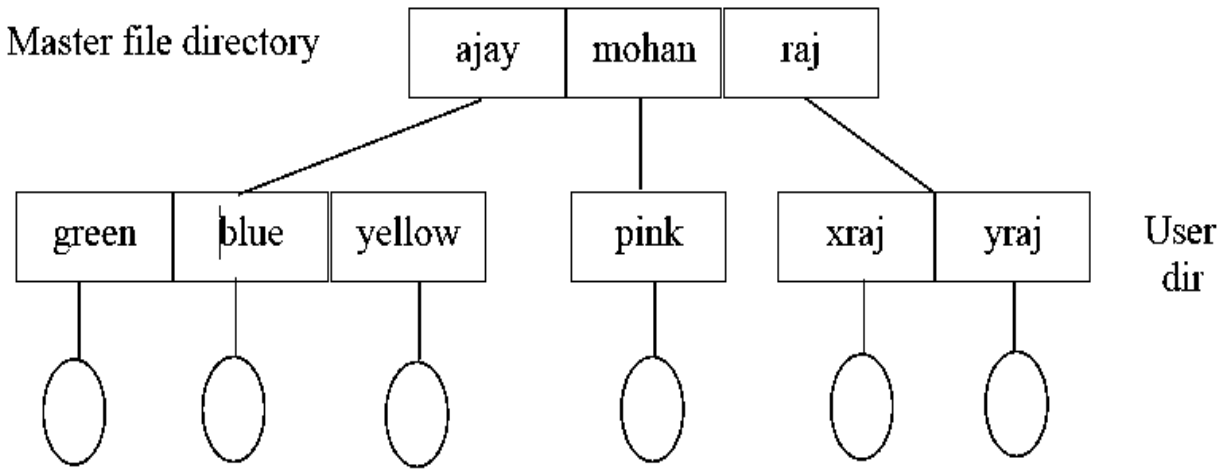


Figure 3 Two-Level Directories

The user directories themselves must be created and deleted as necessary. A special system program is run with the appropriate user name and account information. The program creates a new user file directory and adds an entry for it to the master file directory. The execution of this program might be restricted to system administrators.

The two-level directory structure solves the name-collision problem, but it still has problems. This structure effectively isolates one user from another. This isolation is an advantage when the users are completely independent, but is a disadvantage when the users co-operate on some task and to access one user's account by other users is not allowed. If access is to be permitted, one user must have the ability to name a file in another user's directory.

A two-level directory can be thought of as a tree, or at least an inverted tree. The root of the tree is the master file directory. Its direct descendants are the UFDs. The descendants of the user file directories are the files themselves. Thus, a user name and a file name define a path name. Every file in the system has a path name. To name a file uniquely, user must know the path name of the file desired.

For example, if user A wishes to access her own test file named *test*, he can simply refer to *test*. To access the test file of user B (with directory-entry name *userb*), however, he might have to refer to */userb/test*. Every system has its own syntax for naming files in directories other than the user's own.

There is additional syntax to specify the partition of a file. For instance, in MS-DOS a letter followed by a colon specifies a partition. Thus, file specification might be "C:\userb\bs.test". Some systems go even

further and separate the partition, directory name, and file name parts of the specification. For instance, in VMS, the file "login.com" might be specified as:

"u:[sst.deck.1]login.com;"

where "u" is the name of the partition, "sst" is the name of the directory, "deck" is the name of subdirectory, and "1", is the version number. Other systems simply treat the partition name as part of the directory name. The first name given is that of the partition, and the rest is the directory and file. For instance, "/u/pbg/test" might specify partition "u", directory "pbg", and file "test". A special case of this situation occurs in regard to the system files. Those programs provided as parts of the system (loaders, assemblers, compilers, utility routines, libraries, and so on) are generally defined as files. When the appropriate commands are given to the operating system, these files are read by the loader and are executed. Many command interpreters act by simply treating the command as the name of a file to load and execute. As the directory system is defined presently, this file name would be searched for in the current user file directory .One solution would be to copy the system files into each user file directory. However, copying all the system files would be enormously wasteful of space. The standard solution is to complicate the search procedure slightly. A special user directory is defined to contain the system files.

Whenever a file name is given to be loaded, the operating system first searches the local user file directory. If the file is found, it is used. If it is not found, the system automatically searches the special user directory that contains the system files. The sequence of directories searched when a file is named is called the search path. This idea can be extended, such that the search path contains an unlimited list of directories to search when a command name is given. This method is used in UNlX and MS-DOS.

6.2.2.3 Tree-Structured Directories

A tree system allows growth of the tree beyond the second level. Any directory may contain names of additional directories as well as non-directory objects. This generalization allows users to create their own sub-directories and to organize their files accordingly. The MS-DOS system, for instance, is structured as a tree. In fact, a tree is the most common directory structure. The tree has a root directory. Every file in the system has a unique path name. A path name is the path from the root, through all the subdirectories, to a specified file.

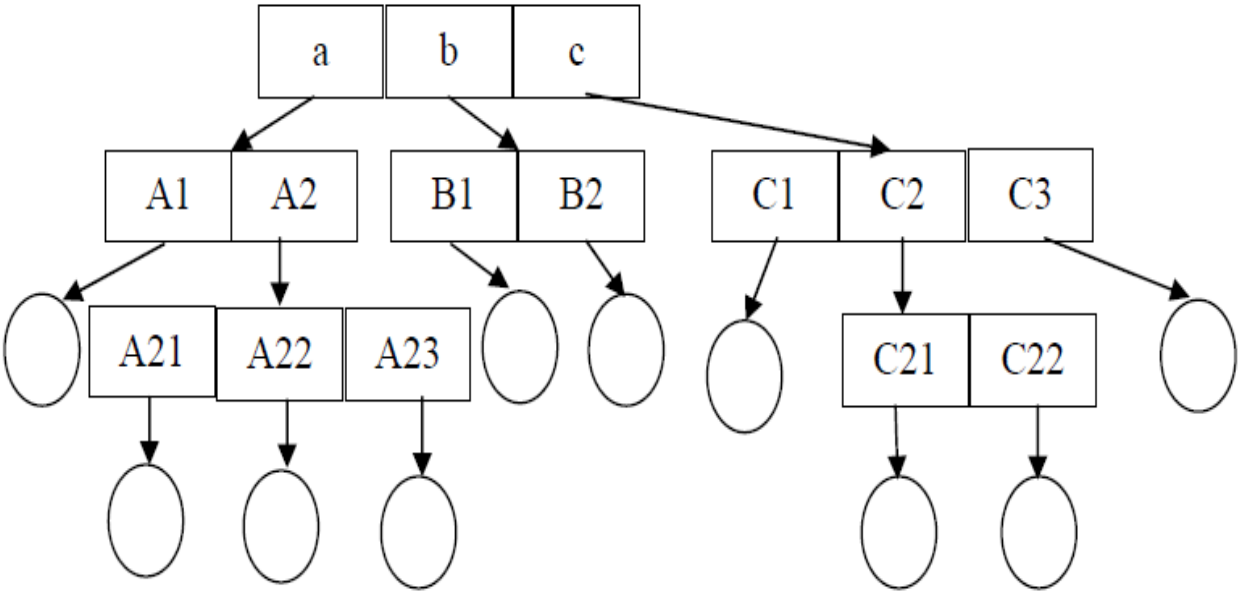


Figure 4 Tree-Structured Directories

A directory (or subdirectory) contains a set of files or subdirectories. A directory is simply another file but it is treated in a special way. All directories have the same internal format, one bit in each directory entry defines the entry as a file (0) or as a subdirectory (1) Special system calls are used to create and delete directories.

In normal use, each user has a current directory .The current directory should contain most of the files that are of current interest to the user. When reference is made to a file, the current directory is searched. If a file is needed that is not in the current directory, then the user must either specify a path name or change the current directory to be the directory holding that file. To change the current directory to a different directory, a system call is provided that takes a directory name as a parameter and uses it to redefine the current directory.

Thus, the user can change his current directory whenever he desires. From one change directory system call to the next, all open system calls search the current directory for the specified file.

The initial current directory of a user is designated when the user job starts or the user logs in. The operating system searches the accounting file (or ask) other predefined location tofind an entry for this user (for accounting). In the accounting file is a pointer to (or the name of) the user's initial directory. This pointer is copied to a local variable for this user, which specifies the user's initial current directory.

Path names can be of two types: absolute path names or relative path names.

(a) **Absolute path:** An absolute path name begins at the root and follows a path down to the desired file, giving the directory names on the path. An absolute path name is an unambiguous way of referring to a file. Thus identically named files created by different users differ in their absolute path names.

(b) **Relative path:** A relative path name defines a path from the current directory.

Allowing the user to define his own subdirectories permits him to impose a structure on his files. This structure might result in separate directories for files associated with different topics (for example, a subdirectory was created to hold the text of this book or different forms of information for example, the directory programs may contain source programs; the directory bin may store all the binary files. An interesting policy decision in a tree-structured directory structure is how to handle the deletion of a directory. If a directory is empty, its entry in its containing directory can simply be deleted. But if the directory to be deleted is not empty, containing files and subdirectories then one of two approaches can be taken. As in MS-DOS, if we want to delete a directory then first of all we have to empty it i.e. delete its contents and If there are any subdirectories, the procedure must be applied recursively to them, so that they can be deleted also. But this approach may be time consuming.

An alternative approach, such as that taken by the UNIX rm command, to provide the option that, when a request is made to delete a directory, and that directory's files and subdirectories are also to be deleted. Note that either approach is fairly easy to implement; the choice is one of policy. The latter policy is more convenient, but more dangerous, because an entire director structure may be removed with one command. If that command was issued in error, a large number of files and directories would need to be restored from backup tapes.

With a tree-structured directory system, users can access, in addition their files, the files of other users. For example, user B can access files of user A by specifying their path names. User B can specify either an absolute or relative path name. Alternatively, user B could change her current directory be user A's directory, and access the files by their file names. Some systems also allow users to define their own search paths. In this case, user B could define her search path to be (1) her local directory, (2) the system file directory, and user A's directory, in that order. As long as the name of a file of user A did not conflict with the name of a local file or system file, it could be referred to simply by its name.

A path to a file in a tree-structured directory can be longer than that in a two-level directory. To allow users to access programs without having to remember these long paths, the Macintosh operating system automates the search for executable programs. It maintains a file called the "Desktop File", containing the name and location of all executable programs it has seen. Where a new hard disk or floppy disk is added to the system, or the network accessed, the operating system traverses the directory structure, searching for executable programs on the device and recording the pertinent information. This mechanism supports the double-click execution functionality. A double-click on a file causes its creator attribute to be read, and the "Desktop File" to be searched for a match.

6.2.2.4 Acyclic-Graph Directories

Sharing of file is another important issue in deciding the directory structure. If more than one user are working on some common project. So the files associated with that project should be placed in a common directory that can be shared among a number of users.

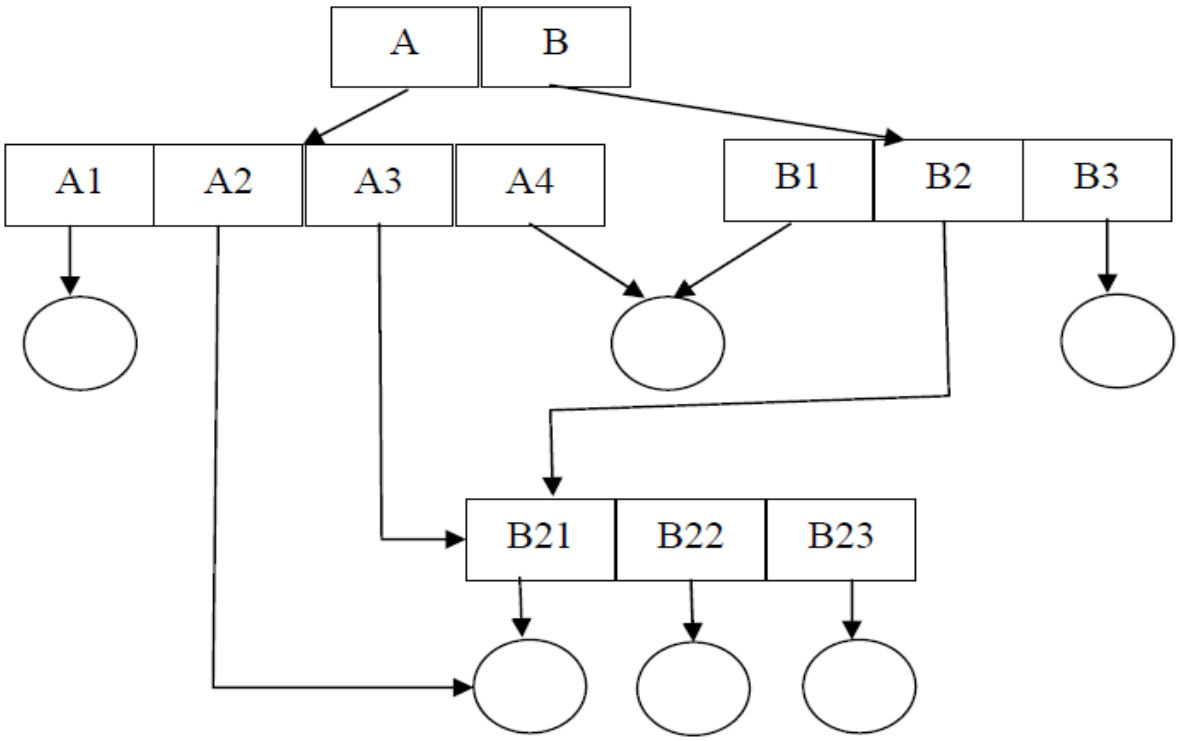


Figure 5 Acyclic-Graph Directories

The important characteristic of sharing is that if a user is making a change in a shared file then that is to be reflected to other user also. In this way a shared file is not the same as two copies of the file. With two copies, each programmer can view the copy rather than the original, but if one programmer changes the file, the changes will not appear in the other's copy. With a shared file, there is only one actual file, so any changes made by the person would be immediately visible to the other.

This form of sharing is particularly important for shared subdirectories; a new file created by one person will automatically appear in all the shared subdirectories. File sharing is facilitated by acyclic graph structure. The tree structure doesn't permit the sharing of files.

In a situation where several people are working as a team, all the files to be shared may be put together into one directory. The user file directories of all the team members would each contain this directory of shared files as a subdirectory. Even when there is a single user, his file organization may require that some files be put into several different subdirectories. For example, a program written for a particular project should be both in the directory of all programs and in the directory for that project.

Shared files and subdirectories can be implemented in several ways. A common approach used in UNIX systems, is to create a new directory entry called a link. A link is a pointer to another file or subdirectory. For example, a link may be implemented as an absolute or relative path name. When a reference to a file is made, we search the directory. The directory entry is marked as a link and the name of the real file (or directory) is given. We resolve the link by using the path name to locate the real file. Links are easily identified by their format in the directory entry (or by their having a special type on systems that support types), and are effectively named indirect pointers. The operating system ignores these links when traversing directory trees to preserve the acyclic structure of the system.

The other approach to implementing shared files is simply to duplicate all information about them in both sharing directories. Thus, both entries are identical and equal. A link is clearly different from the original directory entry; thus, the two are not equal. Duplicate directory entries, however, make the original and the copy indistinguishable. A major problem with duplicate directory entries is maintaining consistency if the file is modified. An acyclic-graph directory structure is more flexible than is a simple tree structure, but is also more complex. Several problems must be considered carefully. Notice that a file may now have multiple absolute path names. Consequently, distinct file names may refer to the same file. This situation is similar to the aliasing problem for programming languages. If we are trying

to traverse the entire file system this problem becomes significant, since we do not want to traverse shared structures more than once.

Another problem involves deletion. When can the space allocated to a shared file be de-allocated and reused? One possibility is to remove the file whenever anyone deletes it, but this action may leave dangling pointers to the now non-existent file. Worse, if the remaining file pointers contain actual disk addresses, and the space is subsequently reused for other files, these dangling pointers may point into the middle of other files.

In a system where sharing is implemented by symbolic links, this situation is somewhat easier to handle. The deletion of a link does not need to affect the original file; only the link is removed. If the file entry itself is deleted, the space for the file is de-allocated, leaving the links dangling. We can search for these links and remove them also, but unless a list of the associated link is kept with each file, this search can be expensive. Alternatively, we can leave the links until an attempt is made to use them. At that time, we can determine that the file of the name given by the link does not exist, and can fail to resolve the link name; the access is treated just like any other illegal file name. (In this case, the system designer should consider carefully what to do when a file is deleted and another file of the same name is created, before a symbolic link to the original file is used.) In the case of UNIX, symbolic links are left when a file is deleted, and it is up to the user to realize that the original file is gone or has been replaced.

Another approach to deletion is to preserve the file until all references to it are deleted. To implement this approach, we must have some mechanism for determining that the last reference to the file has been deleted. We could keep a list of all references to a file (directory entries or symbolic links). When a link or a copy of the directory entry is established, a new entry is added to the file-reference list. When a link or directory entry is deleted, we remove its entry on the list. The file is deleted when its file-reference list is empty. The trouble with this approach is the variable and potentially large size of the file-reference list.

However, we really do not need to keep the entire list -we need to keep only a count of the number of references. So a reference count is maintained with shared file, whenever a reference is made to it, it is incremented by one. On deleting a link, the reference count is decremented by one, when it becomes zero the file can be deleted. The UNIX operating system uses this approach for non-symbolic links, or

hard links, keeping a reference count in the file information block or inode. By effectively prohibiting multiple references to directories, we maintain an acyclic-graph structure.

To avoid these problems, some systems do not allow shared directories link. For example, in MS-DOS, the directory structure is a tree structure, rather than an acyclic graph, thereby avoiding the problems associated with file deletion in an acyclic-graph directory structure.

6.2.2.5 General Graph Directory

One serious problem with using an acyclic graph structure is ensuring that there are no cycles. If we start with a two-level directory and allow users to create subdirectories, a tree-structured directory results. It should be fairly easy to see that simply adding new files and subdirectories to existing tree structure preserves the tree-structured nature. However, when we add links to an existing tree-structured directory, the tree structure is destroyed, resulting in a simple graph structure.

The primary advantage of an acyclic graph is the relative simplicity of the algorithms to traverse file in the graph and to determine when there are no more references to a file. We want to avoid file traversing shared sections of an acyclic graph twice, mainly for performance reasons. If we have just searched a major shared subdirectory for a particular file, without finding that file, we want to avoid searching that subdirectory again; the second search would be a waste of time.

To improve the performance of the system we should avoid searching any component twice in the systems where cycles are permitted. If cycles are not identified by the algorithm then it can be trapped in an infinite loop. One solution is to arbitrarily limit the number of directories, which will be accessed during a search.

A similar problem exists when we are trying to determine when a file can be deleted. As with acyclic-graph directory structures, a value zero in the reference count means that there are no more references to the file or directory, and the file can be deleted. However, it is also possible, when cycles exist, that the reference count may be nonzero, even when it is no longer possible to refer to a directory or file. This anomaly is due to the self-referencing (a cycle) in the directory structure. In this case, it is generally necessary to use a garbage collection scheme to determine when the last reference has been deleted and the disk space can be reallocated.

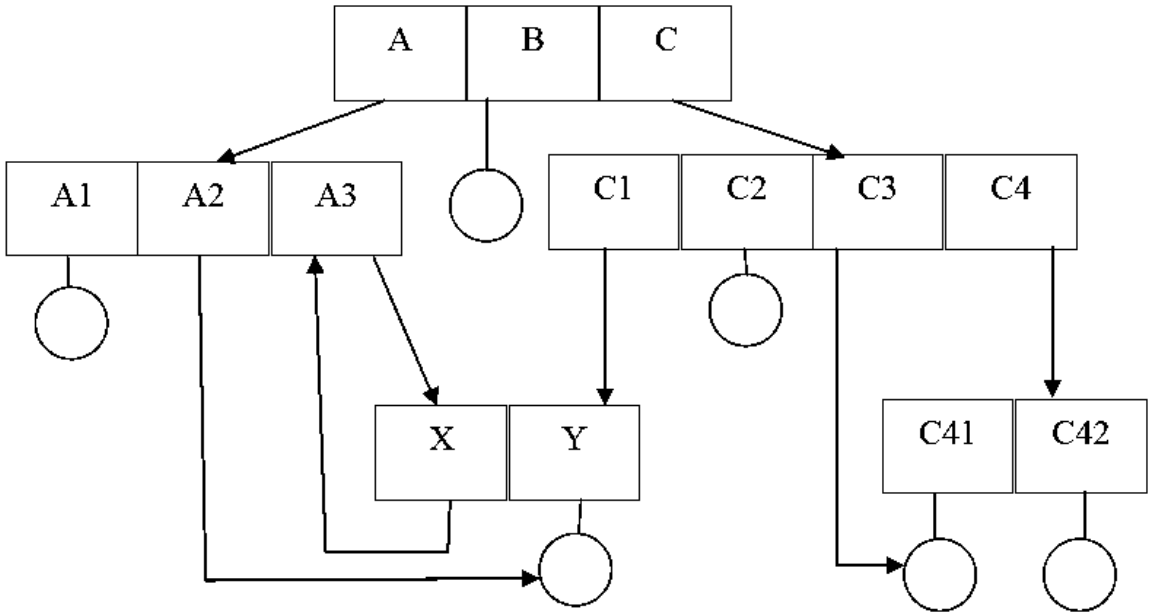


Figure 6 General Graph Directory

Garbage collection involves traversing the entire file system, marking everything that can be accessed. Then, a second pass collects everything that is not marked onto a list of free space. Garbage collection for a disk based file system, however, is extremely time-consuming and is thus seldom attempted. Garbage collection is necessary only because of possible cycles in the graph. Thus, an acyclic-graph structure is much easier to work with. The difficulty is to avoid cycles, as new links are added to the structure. There are algorithms to detect cycles in graphs. However, they are computationally expensive, especially when the graph is on disk storage. Generally, tree directory structures are more common than are acyclic-graph structures.

6.2.3 Directory Operations

The directory can be viewed as a symbol table that translates file names into their directory entries. If we take such a view, then it becomes apparent that the directory itself can be organized in many ways. The different operations that are to be carried out on directories are:

(a) To insert entries.

(b) To delete entries.

(c) To search for a named entry.

(d) To list all the entries in the directory.

When considering a particular directory structure, we need to keep in mind the operations that are to be performed on a directory:

- **Search for a directory:** We need to be able to search a directory structure to find the entry for a particular file. Since files have symbolic names and similar names may indicate a relationship between files, we may want to be able to find all files whose names match a particular pattern.
- **Create a directory:** New files need to be created and added to the directory.
- **Delete a directory:** When a file is no longer needed, we want to remove it from the directory.
- **List a directory:** We need to be able to list the files in a directory and the contents of the directory entry for each file in the list.
- **Rename a directory:** Because the name of a file represents its contents to its users, the name must be changeable when the contents or use of the file changes. Renaming a file may also allow its position within the directory structure to be changed.
- **Traverse the file system:** It is useful to be able to access every directory and every file within a directory structure. For reliability it is a good idea to save the contents and structure of the entire file system at regular intervals. This saving often consists of copying all files to magnetic tape. This technique provides a backup copy in case of system failure or if the file is simply no longer in use. In this case, the file can be copied to tape, and the disk space of that file released for reuse by another file.
- **Copying a directory:** A directory may be copied from one location to another.
- **Moving a directory:** A directory may be moved from one location to a new location with all its contents.

6.3 File Protection & Security

The security of the information is a major issue in file system. The files are to be protected from the physical damage as well as improper access. One way of ensuring the security is through backup. By maintaining the duplicate copy of the files, the reliability is improved. In many systems this is done automatically without human intervention. The backup of the files is done at regular interval automatically. So if a copy of the file is accidentally destroyed, we have its backup copy.

There are a number of factors causing the damage to the file system such as:

(a) Hardware problems.

(b) Power failure

(c) Head crashes

(d) Dirt

(e) Temperature

(f) Bugs in the software

These things can result into the loss of contents of files. Protection can be provided in many ways. For a small single-user system, we might provide protection by physically removing the floppy disks and locking them in a desk drawer or file cabinet. In a multi-user system, however, other mechanisms are needed.

6.3.1 Types of Access

The need for protecting files is a direct result of the ability to access files. On systems that do not permit access to the files of other users, protection is not needed. Thus, one extreme would be to provide complete protection by prohibiting access. The other extreme is to provide free access with no protection. Both of these approaches are too extreme for general use. What is needed is the controlled access.

Protection mechanisms provide controlled access by limiting the types of file access that can be made. Access is permitted or denied depending on several factors, one of which is the type of access requested. Several different types of operations may be controlled:

- **Read** - Read information contained in the file.
- **Write** - Write new information into a file at any point or overwrite existing information in a file.
- **Execute** - Load the contents of a file into main memory and create a process to execute it.
- **Append** - Write new information at the end of the file.
- **Delete** - Delete the file and release its storage space for use in other files.
- **List** – Read the names contained in a directory.

- **Change access** - Change some user's access rights for some controlled operation.

Other operations, such as renaming, copying, or editing the file, may also be controlled. For many systems, however, these higher-level functions (such as copying) may be implemented by a system program that makes lower-level system calls. Protection is provided at only the lower level. For instance, copying a file may be implemented simply by a sequence of read requests. In this case, a user with read access can also cause the file to be copied, printed, and so on.

Many different protection mechanisms have been proposed. Each scheme has its advantages and disadvantages and must be selected as appropriate for intended application. A small computer system that is used by only a few members of a research group may not need the same types of protection as will a large corporate computer that is used for research, finance, and personnel iterations.

6.3.2 Protection structures

An access privilege is a right to make a specific form of access to a file. An access descriptor describes access privileges for a file. The common accesses privileges read, write, and execute are generally represented by r, w, and x descriptors. A user holds access privileges to one or more files and a file is accessible to one or more users. Access control information for a file is a collection of access descriptors for access privileges held by various users. Access control information can be organized in various forms such as Access Control Matrix, access Control Lists etc. which are discussed in the following section:

6.3.2.1 Access Control Matrix

Access control matrix (ACM) consists of rows and columns as shown in the following figure. Each row describes the access privileges held by a user. Each column describes the access control information for a file. Thus ACM (u_i, f_j) =a_{ij} implies that user u_i can access file f_j in accordance with access privileges a_{ij}.

Files Users ↓	f_1	f_2	f_3
u_1	{r}	(r, w}	{r, w, x
u_2		{r}	{r, x}

u_3	{w}	{r, w, x}	{r}

Figure 7 Access Control Matrix

The important advantages of ACM are:

(a) Simplicity and efficiency of access.

(b) All information is stored in one structure.

But its main drawback is its size and sparseness. The size can be reduced by assigning access privileges to group of users rather than the individual users resulting in the reduction of number of rows. The solution of sparseness is the use of lists instead of matrix as discussed following.

6.3.2.2 Access control Lists and Groups

The most common approach to the protection problem is to make access dependent on the

stems identity of the user. Various users may need different types of access to a file or directory. The most general scheme to implement identity-dependent access is to associate with each file and directory an access control list (ACL), specifying the user name and the types of access allowed for each user. Each element of the access control list is an access control pair (<user name>, <list of access privileges>).

When a user requests access to a particular file, the operating system checks the access list associated with that file. If that user is listed for the requested access, the access is allowed. Otherwise, a protection violation occurs, and the user job is denied access to the file.

The main problem with access lists is their length. It depends on the number of users and the number of access privileges defined in the system. Most file systems uses three kinds of access privileges: (a) Read - file can be read, (b) write – file can be modified and new data can be added, and (c) execute – permits the execution of the program. If we want to allow everyone to read a file, we must list all users with read access. This technique has two undesirable consequences:

(a) Constructing such a list may be a tedious and unrewarding task, especially if we do not know in advance the list of users in the system.

(b) The directory entry that previously was of fixed size needs now to be of variable size, resulting in space management being more complicated.

To reduce the size of protection information, users can be classified in some convenient manner and an access control pair can be specified for each class of user rather than for individual users. Now an access control list has only as many pairs as the number of user

classes. To condense the length of the access list, many systems recognize three classifications of users in connection with each file (e.g. in UNIX):

1) Owner - The user who created the file is the owner
2) Group - A set of users who are sharing the file and need similar access is a group or workgroup.
3) Universe - All other users in the system constitute the universe.

Note that, for this scheme to work properly, group membership must be controlled tightly. This control can be accomplished in a number of different ways. For example, in the UNIX system, groups can be created and modified by only the manager of the facility (or by any super-user). Thus, this control is achieved through human interaction. In the VMS system, with each file, an access list (also known as an access control list) may be associated, listing those users who can access the file. The owner of the file can create and modify this access lists are discussed above.

With this more limited protection classification, only three fields are needed to define protection. Each field is often a collection of bits, each of which either allows or prevents the access associated with it. For example, the UNIX system defines three fields of 3 bits each: rwx, where r controls read access, w controls write access, and x controls execution. A

separate field is kept for the file owner, for the owner's group and for all other users. In this scheme, 9 bits per file are needed to record protection information.

6.3.2.3 Other Protection Approaches

Another approach to the protection problem is to associate a password with each file. Access to each file can be controlled by a password. If the passwords are chosen randomly and changed often, this scheme may be effective in limiting access to a file to only those users who know the password.

There are several disadvantages to this scheme.

(a) First, if we associate a separate password with each file, then the number of passwords that a user needs to remember may become large, making the scheme impractical.

(b) If only one password is used for all the files, then, once it is discovered, all files are accessible.

Some systems allow a user to associate a password with a subdirectory, rather than with an individual file, to deal with this problem. The IBM VM/CMS operating system allows three passwords for a minidisk: one each for read, write, and multi write access. Second, commonly, only one password is associated with each file. Thus, protection is on an all-or-nothing basis. To provide protection on a more detailed level, we must use multiple passwords.

Limited file protection is also currently available on single user systems, such as MS-DOS and Macintosh operating system. These operating systems, when originally designed, essentially ignored dealing with the protection problem. However, since these systems are being placed on networks where file sharing and communication is necessary, protection mechanisms have to be retrofitted into the operating system. Note that it is almost always easier to design a feature into a new operating system than it is to add a feature to an existing one. Such updates are usually less effective and are not seamless.

We note that, in a multilevel directory structure, we need not only to protect individual files, but also to protect collections of files contained in a subdirectory, that is, we need to provide a mechanism for directory protection.

The directory operations that must be protected are somewhat different from the file operations. We want to control the creation and deletion of files in a directory. In addition, we probably want to control whether a user can determine the existence of a file in a directory. Sometimes, knowledge of the existence and name of a file may be significant in itself. Thus, listing the contents of a directory must be a protected operation. Therefore, if a path name refers to a file in a directory, the user must be allowed access to both the directory and the file. In systems where files may have numerous path names (such as acyclic or general graphs), a given user may have different access rights to a file, depending on the path name used.

6.4 Check your progress

1. _____ is a unique tag, usually a number identifies the file within the file system.

a) File identifier

b) File name

c) File type

d) None of the mentioned

2. To create a file _____

a) allocate the space in file system

b) make an entry for new file in directory

c) allocate the space in file system & make an entry for new file in directory

d) none of the mentioned

3. By using the specific system call, we can _____

a) open the file

b) read the file

c) write into the file

d) all of the mentioned

4. File type can be represented by _____

a) file name

b) file extension

c) file identifier

d) none of the mentioned

5. Which file is a sequence of bytes organized into blocks understandable by the system's linker?

a) object file

b) source file

c) executable file

d) text file

6. What is the mounting of file system?

a) crating of a filesystem

b) deleting a filesystem

c) attaching portion of the file system into a directory structure

d) removing the portion of the file system into a directory structure

7. Mapping of file is managed by _____

a) file metadata

b) page table

c) virtual memory

d) file system

8. Mapping of network file system protocol to local file system is done by _____

a) network file system

b) local file system

c) volume manager

d) remote mirror

9. Which one of the following explains the sequential file access method?

a) random access according to the given byte number

b) read bytes one at a time, in order

c) read/write sequentially by record

d) read/write randomly by record

10. When will file system fragmentation occur?

a) unused space or single file are not contiguous

b) used space is not contiguous

c) unused space is non-contiguous

d) multiple files are non-contiguous

6.5 Summary

The file system resides permanently on secondary storage, which has the main requirement that it must be able to hold a large amount of data, permanently.

The various files can be allocated space on the disk in three ways: through contagious, linked or indexed allocation. Contagious allocation can suffer from external fragmentation. Direct-access is very inefficient with linked-allocation. Indexed allocation may require substantial overhead for its index block. There are many ways in which these algorithms can be optimized.

Free space allocation methods also influence the efficiency of the use of disk space, the performance of the file system and the reliability of secondary storage.

6.6 Keywords

FIFO Files: FIFO (first-in-first-out) are files that allow unrelated processes to communicate with each other.

Contiguous allocation, each file occupies a set of contiguous blocks on the disk.

Indexed allocation all pointers are brought together into one block called the index block.

Sequential file is the most primitive of all file structures it has no directory and no linking pointers

Direct access file organization, any records can be accessed irrespective of the current position in the file

Directory: A directory may be defined as an object that contains the names of the file system objects.

Access Control Matrix: ACM is a matrix in which each row describes the access privileges held by a user and each column, access control information for a file.

Access Control List: It is a structure to implement identity-dependent access to each file and directory where each element of the ACL is an access control pair (<user name>, <list of access privileges>).

6.7 Self-Assessments Test

1. Define field, record, file, file sharing, and file protection.
2. What are the limitations of acyclic directory structure?
3. Which file operations are applicable to directories? Which are not?
4. How is a directory different from a file?
5. What are the different logical structures of the directory? Discuss their merits and demerits?
6. Discuss the advantages and disadvantages of Access Control Lists (ACL) and Access Control Matrix (ACM).
7. Discuss the advantages and disadvantages of two-level directory structure over single-level directory structure.

6.8 Answers to Check Your Progress

11. File identifier
12. allocate the space in file system & make an entry for new file in directory
13. all of the mentioned
14. file extension

15. object file
16. attaching portion of the file system into a directory structure
17. file metadata
18. network file system
19. read bytes one at a time, in order
20. unused space or single file are not contiguous

6.9 Reference/Suggested Readings

1. Operating System Concepts, 5th Edition, Silberschatz A., Galvin P.B., John Wiley & Sons.
2. Systems Programming & Operating Systems, 2nd Revised Edition, Dhamdhere D.M., Tata McGraw Hill Publishing Company Ltd., New Delhi.
3. Operating Systems, Madnick S.E., Donovan J.T., Tata McGraw Hill Publishing Company Ltd., New Delhi.
4. Operating Systems-A Modern Perspective, Gary Nutt, Pearson Education Asia, 2000.
5. Operating Systems, Harris J.A., Tata McGraw Hill Publishing Company Ltd., New Delhi, 2002.

Course: MCA-32	Writer: Ritu
Lesson: 7	Vetter:

Deadlocks

Structure

7.0 Objectives

The objectives of this lesson are to make the students acquainted with the problem of deadlocks. In this lesson, we characterize the problem of deadlocks and discuss policies, which an Operating System can use to ensure their absence. Deadlock detection, resolution, prevention and avoidance have been discussed in detail in the present lesson.

After studying this lesson the students will be familiar with following:

(a) Condition for deadlock.

(b) Deadlock prevention

(c) Deadlock avoidance

(d) Deadlock detection and recovery

7.1 Introduction

A serious danger of concurrent programming is deadlock. A race condition produces incorrect results, where a deadlock results in the deadlocked processes never making any progress. In the simplest form it's a process waiting for a resource held by a second process that's waiting for a resource that the first holds.

In a multiprogramming environment where several processes compete for resources, a situation may arise where a process is waiting for resources that are held by other waiting processes. This situation is called a deadlock. Generally, a system has a finite set of resources (such as memory, IO devices, etc.) and a finite set of processes that need to use these resources. A process which wishes to use any of these resources makes a request to use that resource. If the resource is free, the process gets it. If it is used by another process, it waits for it to become free. The assumption is that the resource will

eventually become free and the waiting process will continue on to use the resource. But what if the other process is also waiting for some resource?

"A set of processes is in a deadlock state when every process in the set is waiting for an event that can only be caused by another process in the set."If a process is in the need of some resource, physical or logical, it requests the kernel of operating system. The kernel, being the resource manager, allocates the resources to the processes. If there is a delay in the allocation of the resource to the process, it results in the idling of process. The deadlock is a situation in which some processes in the system faces indefinite delays in resource allocation. In this lesson, we identify the problems causing deadlocks, and discuss a number of policies used by the operating system to deal with the problem of deadlocks.

Deadlock involving a set of processes D is a situation in which:

(a) Every process P_i in D is blocked on some event E_i.

(b) Event E_i can be caused only by action of some process (es) in D.

In other words, each member of the set of deadlock processes is waiting for a resource that can be released only by a deadlock process. None of the processes can run, none of them can release any resources, and none of them can be awakened. The resources may be either physical or logical. Examples of physical resources are Printers, Tape Drivers, Memory Space, and CPU Cycles. Examples of logical resources are Files, Semaphores, and Monitors.

Here's an example:

```
#define N 100 /* Number of free slots */
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0; void
producer()
{
        item i; while (1)
        {
```

```
            i = produce_item(); P
            (mutex);
            P (empty);
            insert_new_item(i); V
            (mutex);
            V (full);
        }
}
void consumer()
{
        item i; while (1)
        {
            P (full);
            P (mutex);
            i = get_next_item(); V
            (mutex);
            V (empty); consume_item
            (i);}
        }
```

When the producer arrives at a full queue, it will block on empty while holding mutex, and the consumer will subsequently block on mutex forever.

What are the consequences of deadlocks?

- Response times and elapsed times of processes suffer.
- If a process is allocated a resource R1 that it is not using and if some other process P2 requires the resource, then P2 is denied the resource and the resource remains idle.

7.3 Preemptable and Non-preemptable Resources

Resources come in two flavors: preemptable and non-preemptable. A preempt able resource is one that can be taken away from the process with no ill effects. Memory is an example of a preemptable resource. On the other hand, a non-preempt able resource is one that cannot be taken away from process (without causing ill effect). For example, CD resources are not preemptable at an arbitrary moment. Reallocating resources can resolve deadlocks that involve preemptable resources. Deadlocks that involve nonpreemptable resources are difficult to deal with.

7.4 Necessary and Sufficient Deadlock Conditions

Coffman (1971) identified four (4) conditions that must hold simultaneously for there to be a deadlock.

1. Mutual Exclusion Condition

The resources involved are non-shareable.

Explanation: At least one resource must be held in a non-shareable mode, that is, only one process at a time claims exclusive control of the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

2. Hold and Wait Condition

3. Requesting process hold already, resources while waiting for requested resources.

Explanation: There must exist a process that is holding a resource already allocated to it while waiting for additional resource that are currently being held by other processes.

3. No-Preemptive Condition

Resources already allocated to a process cannot be preempted.

Explanation: Resources cannot be removed from the processes are used to completion or released voluntarily by the process holding it.

3. Circular Wait Condition

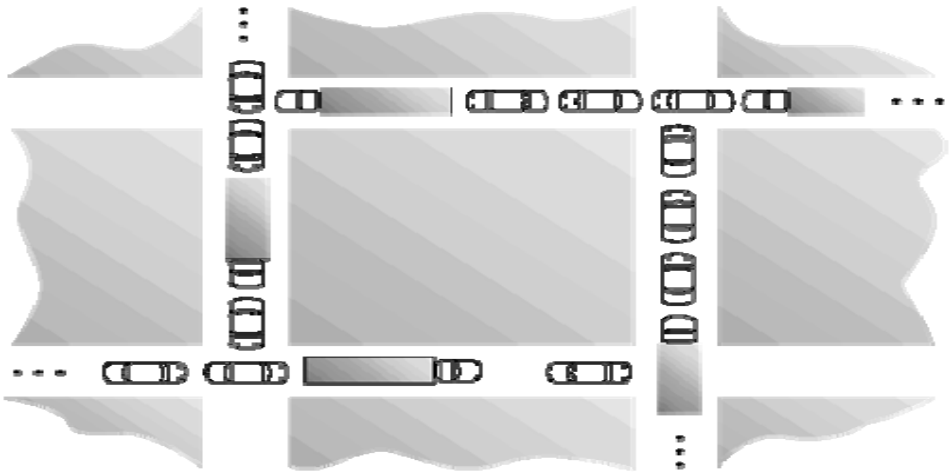
The processes in the system form a circular list or chain where each process in the list is waiting for a resource held by the next process in the list.

A set {P0, P1, P2, …, Pn} of waiting processes must exist such that P0 is waiting for a resource

that is held by P1, P1 is waiting for a resource that is held by P2,

…, Pn-1 is waiting for a resource that is held by Pn, and Pn is waiting for a resource that is held by P0.

Conditions 1 and 3 pertain to resource utilization policies, while condition 2 pertains to resource requirements of individual processes. Only condition 4 pertains to relationships between resource requirements of a group of processes. As an example, consider the traffic deadlock in the following figure:



Consider each section of the street as a resource.

1. Mutual exclusion condition applies, since only one vehicle can be on a section of the street at a time.
2. Hold-and-wait condition applies, since each vehicle is occupying a section of the street, and waiting to move on to the next section of the street.
3. No-preemptive condition applies, since a section of the street that is occupied by a vehicle cannot be taken away from it.
4. Circular wait condition applies, since each vehicle is waiting on the next vehicle to move. That is, each vehicle in the traffic is waiting for a section of street held by the next vehicle in the traffic.

The simple rule to avoid traffic deadlock is that a vehicle should only enter an intersection if it is assured that it will not have to stop inside the intersection.

It is not possible to have a deadlock involving only one single process. The deadlock involves a circular "hold-and-wait" condition between two or more processes, so "one" process cannot hold a resource, yet be waiting for another resource that it is holding. In addition, deadlock is not possible between two threads in a process, because it is the process that holds resources, not the thread that is, each thread has access to the resources held by the process.

7.5 Resource-Allocation Graph

The deadlock conditions can be modeled using a directed graph called a **resource allocation graph** (RAG). A resource allocation graph is a directed graph. It consists of 2 kinds of nodes:

Boxes — Boxes represent resources, and Instances of the resource are represented as dots within the box i.e. how many units of that resource exist in the system.

Circles — Circles represent threads / processes. They may be a user process or a system process.

An edge can exist only between a process node and a resource node. There are 2 kinds of (directed) edges:

Request edge: It represents resource request. It starts from process and terminates to a resource. It indicates the process has requested the resource, and is waiting to acquire it.

Assignment edge: It represents resource allocation. It starts from resource instance and terminates to process. It indicates the process is holding the resource instance.

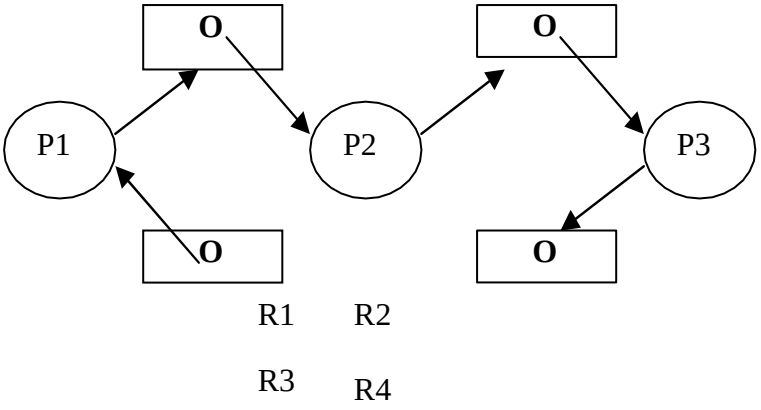
When a request is made, a request edge is added.

When request is fulfilled, the request edge is transformed into an assignment edge.

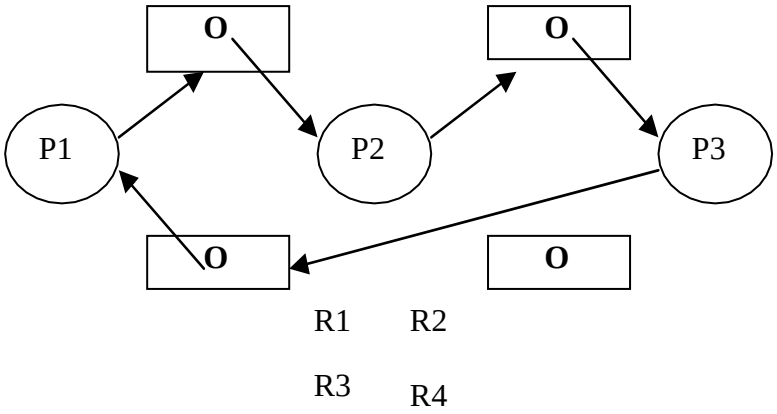
When process releases the resource, the assignment edge is deleted.

7.5.1 Interpreting a Resource Allocation Graph with Single ResourceInstances

Following figure shows a resource allocation graph. If the graph does not contain a cycle, then no deadlock exists. Following figure is an example of a no deadlock situation.



If the graph does contain a cycle, then a deadlock does exist. As following resource allocation graph depicts a deadlock situation.



With single resource instances, a cycle is a necessary and sufficient condition for deadlock

So basic fact is that If graph contains no cycles then there is no deadlock. But If graph contains a cycle then there are two possibilities:

(a) If only one instance per resource type, then there is a deadlock.

(b) If several instances per resource type, possibility of deadlock is there.

7.6 Dealing with Deadlock

There are following approaches to deal with the problem of deadlock. The Ostrich Approach: sticks your head in the sand and ignores the problem. This approach can be quite useful if you believe that they are rarest chances of deadlock occurrence. In that situation it is not a justifiable proposition to invest a lot in identifying deadlocks and tackling with it. Rather a better option is ignore it. For example if each PC deadlocks once per 100 years, the one reboot may be less

painful that the restrictions needed to prevent it. But clearly it is not a good philosophy for nuclear missile launchers.

Deadlock prevention: This approach prevents deadlock from occurring by eliminating one of the four (4) deadlock conditions.

Deadlock detection algorithms: This approach detects when deadlock has occurred.

Deadlock recovery algorithms: After detecting the deadlock, it breaks the deadlock.

Deadlock avoidance algorithms: This approach considers resources currently available, resources allocated to each thread, and possible future requests, and only fulfill requests that will not lead to deadlock

7.7 Deadlock Prevention

Deadlock prevention is based on designing resource allocation policies, which make deadlocks impossible. Use of the deadlock prevention approach avoids the over- head of deadlock detection and resolution. However, it incurs two kinds of costs - overhead of using the resource allocation policy, and cost of resource idling due to the policy.

As described in earlier section, four conditions must hold for a resource deadlock to arise in a system:

- Non-shareable resources
- Hold-and-wait by processes
- No preemption of resources
- Circular waits.

Havender in his pioneering work showed that since all four of the conditions are necessary for deadlock to occur, it follows that deadlock might be prevented by denying any one of the conditions. Ensuring that one of these conditions cannot be satisfied prevents deadlocks. We first discuss how each of these conditions can be prevented and then discuss a couple of resource allocation policies based on the prevention approach.

7.7.1 Elimination of "Mutual Exclusion" Condition

The mutual exclusion condition must hold for non-sharable resources. That is, several processes cannot simultaneously share a single resource. This condition is difficult to eliminate because

some resources, such as the tap drive and printer, are inherently non-shareable. Note that shareable resources like read- only-file do not require mutually exclusive access and thus cannot be involved in deadlock.

Some resources can be made sharable. Spooling can make devices like printers or tape drives sharable. Spooling is storing the output on a shared medium, like disk, and using a single process to coordinate access to the shared resource. Print spooling is the cannonical example. There are generally some resources that cannot be spooled - semaphores, for example. Removing mutual exclusion is not always an option.

7.7.2 Elimination of "Hold and Wait" Condition

There are two possibilities for elimination of the second condition. The first alternative is that a process request be granted all of the resources it needs at once, prior to execution. The second alternative is to disallow a process from requesting resources whenever it has previously allocated resources. This strategy requires that all of the resources a process will need must be requested at once. The system must grant resources on "all or none" basis. If the complete set of resources needed by a process is not currently available, then the process must wait until the complete set is available. While the process waits, however, it may not hold any resources. Thus the "wait for" condition is denied and deadlocks simply cannot occur. This strategy can lead to serious waste of resources. For example, a program requiring ten tap drives must request and receive all ten derives before it begins executing. If the program needs only one tap drive to begin execution and then does not need the remaining tap drives for several hours. Then substantial computer resources (9 tape drives) will sit idle for several hours. This strategy can cause indefinite postponement (starvation). Since not all the required resources may become available at once.

7.7.3 Elimination of "No-preemption" Condition

The nonpreemption condition can be alleviated by forcing a process waiting for a resource that cannot immediately be allocated to relinquish all of its currently held resources, so that other processes may use them to finish. Suppose a system does allow processes to hold resources while requesting additional resources. Consider what happens when a request cannot be satisfied. A process holds resources a second process may need in order to proceed while second process may hold the resources needed by the first process. This is a deadlock. This strategy requires that when

a process that is holding some resources is denied a request for additional resources. The process must release its held resources and, if necessary, request them again together with additional resources. Implementation of this strategy denies the "no-preemptive" condition effectively.

The main drawback of this approach is high cost. When a process releases resources the process may lose all its work to that point. One serious consequence of this strategy is the possibility of indefinite postponement (starvation). A process might be held off indefinitely as it repeatedly requests and releases the same resources.

7.7.4 Elimination of "Circular Wait" Condition

Presence of a cycle in resource allocation graph indicates the "circular wait" condition. The last condition, the circular wait, can be denied by imposing a total ordering on all of the resource types and then forcing, all processes to request the resources in numerical order (increasing or decreasing). With this rule, the resource allocation graph can never have a cycle. For example, provide a global numbering of all the resources, as shown

Card Reader

1Printer

2Plotter

3Tape Drive

4Card Punch

Now the rule is this: processes can request resources whenever they want to, but all requests must be made in numerical order. A process may request first printer and then a tape drive (order: 2, 4), but it may not request first a plotter and then a printer (order: 3, 2). The problem with this strategy is that it may be impossible to find an ordering that satisfies everyone. The resource ranking policy works best when all processes require their resources in the order of increasing ranks.

However, difficulty arises when a process requires resources in some other order. Now processes may tend to circumvent such difficulties by acquiring lower ranking resources much before they are actually needed. In the worst case this policy may degenerate into the 'all requests together' policy of resource allocation. Anyway this policy is attractive due to its simplicity once resource ranks have been assigned. "All requests together" is the simplest of all deadlock prevention

policies. A process must make its resource requests together-typically, at the start of its execution. This restriction permits a process to make only one multiple request in its lifetime. Since resources requested in a multiple request are allocated together, a blocked process does not hold any resources. The hold-and-wait condition is satisfied. Hence paths of length larger than 1 cannot exist in the Resource Allocation Graph, a mutual wait-for relationships cannot develop in the system. Thus, deadlocks cannot arise.

Example: The most common method of preventing deadlock is to prevent the circular wait. A simple way to do this, when possible, is to order the resources and always acquire them in order. Because a process can't be waiting on a lower numbered process while holding a higher numbered one, a cycle is impossible. One can consider the Dining Philosophers to be a deadlock problem, and can apply deadlock prevention to it by numbering the forks and always acquiring the lowest numbered fork first. #define N 5 /* Number of philosphers */ #define RIGHT(i) (((i)+1) %N)

```
#define LEFT(i) (((i)==N) ? 0 : (i)+1)
typedef enum { THINKING, HUNGRY, EATING } phil_state; phil_state state[N];
semaphore mutex =1;
semaphore f[N]; /* one per fork, all 1*/ void get_forks (int i)
{
    int max, min;
    if ( RIGHT(i) > LEFT(i) )
    {
    }
    else
    {

    }
max = RIGHT(i); min = LEFT(i);
min = RIGHT(i); max = LEFT(i);
```

```
        P (f [min]);
        P (f [max]);
}
void put_forks (int i)
{
        V (f [LEFT (i)]);
        V (f [RIGHT (i)]);
}
void philosopher (int process)
{
While (1)
        {
        think ();
        get_forks (process); eat ();
        put_forks (process);
        }
}
```

This solution doesn't get maximum parallelism, but it is an otherwise valid solution.

7.8 Deadlock Avoidance

This approach to the deadlock problem anticipates deadlock before it actually occurs. This approach employs an algorithm to access the possibility that deadlock could occur and acting accordingly. This method differs from deadlock prevention, which guarantees that deadlock cannot occur by denying one of the necessary conditions of deadlock. If the necessary conditions for a deadlock are in place, it is still possible to avoid deadlock by being careful when resources are allocated. Perhaps the most famous deadlock avoidance algorithm, due to Dijkstra [1965], is the Banker's algorithm. So named because the process is analogous to that used by a banker in deciding if a loan can be safely made.

7.8.1 Banker's Algorithm

In this analogy

Customers ≡ processes

Units ≡ resources, say, tape drive

Banker ≡ Operating System

Customers	Used	Max	
A	0	6	
B	0	5	Available
C	0	4	Units = 10
D	0	7	

In the above figure, we see four customers each of whom has been granted a number of credit units. The banker reserved only 10 units rather than 22 units to service them. At certain moment, the situation becomes

Customers	Used	Max	
A	1	6	
B	1	5	Available
C	2	4	Units = 2
D	4	7	

Safe State The key to a state being safe is that there is at least one way for all users to finish. In other analogy, the state of figure 2 is safe because with 2 units left, the banker can delay any request except C's, thus letting C finish and release all four resources. With four units in hand, the banker can let either D or B have the necessary units and so on.

Unsafe State Consider what would happen if a request from B for one more unit were granted in above figure 2. We would have following situation

Customers	Used	Max	
A	1	6	
B	2	5	Available
C	2	4	Units = 1
D	4	7	

This is an unsafe state.

If all the customers namely A, B, C, and D asked for their maximum loans, then banker could not satisfy any of them and we would have a deadlock.

Important Note: It is important to note that an unsafe state does not imply the existence or even the eventual existence a deadlock. What an unsafe state does imply is simply that some unfortunate sequence of events might lead to a deadlock.

The Banker's algorithm is thus to consider each request as it occurs, and see if granting it leads to a safe state. If it does, the request is granted, otherwise, it postponed until later. Haberman [1969] has shown that executing of the algorithm has complexity proportional to N^2 where N is the number of processes and since the algorithm is executed each time a resource request occurs, the overhead is significant.

7.8.2 Evaluation of Deadlock Avoidance Using the Banker's Algorithm

There are following advantages and disadvantages of deadlock avoidance using Banker's algorithm.

Advantages:

- There is no need to preempt resources and rollback state (as in deadlock detection and recovery)
- It is less restrictive than deadlock prevention

Disadvantages:

- In this case maximum resource requirement for each process must be stated in advance.
- Processes being considered must be independent (i.e., unconstrained by synchronization

requirements)

- There must be a fixed number of resources (i.e., can't add resources, resources can't break) and processes (i.e., can't add or delete processes)
- Huge overhead — Operating system must use the algorithm every time a resource is requested. So a huge overhead is involved.

7.9 Deadlock Detection

Deadlock detection is the process of actually determining that a deadlock exists and identifying the processes and resources involved in the deadlock. The basic idea is to check allocation against resource availability for all possible allocation sequences to determine if the system is in deadlocked state. Of course, the deadlock detection algorithm is only half of this strategy. Once a deadlock is detected, there needs to be a way to recover. Several alternatives exist:

- Temporarily prevent resources from deadlocked processes.
- Back off a process to some check point allowing preemption of a needed resource and restarting the process at the checkpoint later.
- Successively kill processes until the system is deadlock free.

These methods are expensive in the sense that each iteration calls the detection algorithm until the system proves to be deadlock free. The complexity of algorithm is O (N^2) where N is the number of processes. Another potential problem is starvation; same process killed repeatedly.

7.10 Deadlock Recovery

Once you have discovered that there is a deadlock, what do you do about it? One thing to do is simply re-boot. A less drastic approach is to yank back a resource from a process to break a cycle. As we saw, if there are no cycles, there is no deadlock. If the resource is not preemptable, snatching it back from a process may do irreparable harm to the process. It may be necessary to kill the process, under the principle that at least that's better than crashing the whole system.

So once a deadlock has been detected, one of the 4 conditions must be invalidated to remove the deadlock. Generally, the system acts to remove the circular wait, because making a system suddenly preemptive with respect to resources, or making a resource suddenly sharable is usually

impractical. Because resources are generally not dynamic, the easiest way to break such a cycle is to terminate a process.

Usually the process is chosen at random, but if more is known about the processes, that information can be used. For example the largest or smallest process can be disabled or the one waiting the longest. Such discrimination is based on assumptions about the system workload.

Some systems facilitate deadlock recovery by implementing check pointing and rollback. Check pointing is saving enough state of a process so that the process can be restarted at the point in the computation where the checkpoint was taken. Auto saving file edits is a form of check pointing. Check pointing costs depend on the underlying algorithm. Very simple algorithms (like linear primarily testing) can be check pointed with a few words of data. More complicated processes may have to save all the process state and memory. Checkpoints are taken less frequently than deadlock is checked for. If a deadlock is detected, one or more processes are restarted from their last checkpoint. The process of restarting a process from a checkpoint is called rollback. The hope is that the resource requests will not interleave again to produce deadlock.

Deadlock recovery is generally used when deadlocks are rare, and the cost of recovery (process termination or rollback) is low. Process check pointing can also be used to improve reliability (long running computations), assist in process migration (Sprite, Mach), or reduce startup costs (emacs).

Database systems use checkpoints, as well as a technique called logging, allowing them to run processes “backwards,” undoing everything they have done. It works like this: Each time the process performs an action, it writes a log record containing enough information to undo the action. For example, if the action is to assign a value to a variable, the log record contains the previous value of the record. When a database discovers a deadlock, it picks a victim and rolls it back. Rolling back processes involved in deadlocks can lead to a form of starvation, if we always choose the same victim. We can avoid this problem by always choosing the youngest process in a cycle. After being rolled back enough times, a process will grow old enough that it never gets chosen as the victim--at worst by the time it is the oldest process in the system. If deadlock recovery involves killing a process altogether and restarting it, it is important to mark the “starting

time" of the reincarnated process as being that of its original version, so that it will look older that new processes started since then.

When should you check for deadlock? There is no one best answers to this question; it depends on the situation. The most "eager" approach is to check whenever we do something that might create a deadlock. Since a process cannot create a deadlock when releasing resources, we only have to check on allocation requests. If the Operating System always grants requests as soon as possible, a successful request also cannot create a deadlock. Thus we only have to check for a deadlock when a process becomes blocked because it made a request that cannot be immediately granted. However, even that may be too frequent. As we saw, the deadlock-detection algorithm can be quite expensive if there are a lot of processes and resources, and if deadlock is rare, we can waste a lot of time checking for deadlock every time a request has to be blocked.

What's the cost of delaying detection of deadlock? One possible cost is poor CPU utilization. In an extreme case, if all processes are involved in a deadlock, the CPU will be completely idle. Even if there are some processes that are not deadlocked, they may all be blocked for other reasons (e.g. waiting for I/O). Thus if CPU utilization drops, that might be a sign that it's time to check for deadlock. Besides, if the CPU isn't being used for other things, you might as well use it to check for deadlock!

On the other hand, there might be a deadlock, but enough non-deadlocked processes to keep the system busy. Things look fine from the point of view of the Operating System, but from the selfish point of view of the deadlocked processes, things are definitely not fine. If the processes may represent interactive users, who can't understand why they are getting no response. Worse still, they may represent time-critical processes (missile defense, factory control, hospital intensive care monitoring, etc.) where something disastrous can happen if the deadlock is not detected and corrected quickly. Thus another reason to check for deadlock is that a process has been blocked on a resource request "too long." The definition of "too long" can vary widely from process to process. It depends both on how long the process can reasonably expect to wait for the request, and how urgent the response is. If an overnight run deadlocks at 11pm and nobody is going to look at its output until 9am the next day, it doesn't matter whether the deadlock is detected at 11:01pm or 8:59am. If all the processes in a system are sufficiently similar, it may be adequate simply to

check for deadlock at periodic intervals (e.g., one every 5 minutes in a batch system; once every millisecond in a real-time control system).

7.11 Mixed approaches to deadlock handling

The deadlock handling approaches differ in terms of their usage implications. Hence it is not possible to use a single deadlock handling approach to govern the allocation of all resources. The following mixed approach is found useful:

System control block: Control blocks like JCB, PCB etc. can be acquired in a specific order. Hence resource ranking can be used here. If a simpler strategy is desired, all control blocks for a job or process can be allocated together at its initiation.

I/O devices files: Avoidance is the only practical strategy for these resources. However, in order to eliminate the overheads of avoidance, new devices are added as and when needed. This is done using the concept of spooling. If a system has only one printer, many printers are created by using some disk area to store a file to be printed. Actual printing takes place when a printer becomes available.

Main memory: No deadlock handling is explicitly necessary. The memory allocated to a program is simply preempted by swapping out the program whenever the memory is needed for another program.

7.12 Evaluating the Approaches to Dealing with Deadlock

- The Ostrich Approach — ignoring the problem

  It is a good solution if deadlock is not frequent.

- Deadlock prevention — eliminating one of the four (4) deadlock conditions This approach may be overly restrictive and results into the under utilization of the resources.

- Deadlock detection and recovery — detect when deadlock has occurred, then break the deadlock

  In it there is a tradeoff between frequency of detection and performance / overhead added.

- Deadlock avoidance — only fulfilling requests that will not lead to deadlock

  It needs too much a priori information and not very dynamic (can’t add processes or

resources), and involves huge overhead

7.13 Check your Progress

1. Which of the following condition is required for a deadlock to be possible?

a) mutual exclusion

b) a process may hold allocated resources while awaiting assignment of other resources

c) no resource can be forcibly removed from a process holding it

d) all of the mentioned

2. A system is in the safe state if _____

a) the system can allocate resources to each process in some order and still avoid a deadlock

b) there exist a safe sequence

c) all of the mentioned

d) none of the mentioned

3. A problem encountered in multitasking when a process is perpetually denied necessary resources is called _____

a) deadlock

b) starvation

c) inversion

d) aging

4. To avoid deadlock _____

a) there must be a fixed number of resources to allocate

b) resource allocation must be done only once

c) all deadlocked processes must be aborted

d) inversion technique can be used

5. The request and release of resources are _____

a) command line statements

b) interrupts

c) system calls

d) special programs

6. For a deadlock to arise, which of the following conditions must hold simultaneously?

a) Mutual exclusion

b) No preemption

c) Hold and wait

d) All of the mentioned

7. Deadlock prevention is a set of methods _____

a) to ensure that at least one of the necessary conditions cannot hold

b) to ensure that all of the necessary conditions do not hold

c) to decide if the requested resources for a process have to be given or not

d) to recover from a deadlock

8. A deadlock avoidance algorithm dynamically examines the _____ to ensure that a circular wait condition can never exist.

a) resource allocation state

b) system storage state

c) operating system

d) resources

9. If no cycle exists in the resource allocation graph _____

a) then the system will not be in a safe state

b) then the system will be in a safe state

c) all of the mentioned

d) none of the mentioned

10. If deadlocks occur frequently, the detection algorithm must be invoked _____

a) rarely

b) frequently

c) rarely & frequently

d) none of the mentioned

7.14 Summary

- A set of process is in a deadlock state if each process in the set is waiting for an event that can be caused by only another process in the set. Processes compete for physical and logical

resources in the system. Deadlock affects the progress of processes by causing indefinite delays in resource allocation.

- There are four Necessary and Sufficient Deadlock Conditions (1) Mutual Exclusion Condition: The resources involved are non-shareable, (2) Hold and Wait Condition: Requesting process hold already, resources while waiting for requested resources,(3) No-Preemptive Condition: Resources already allocated to a process cannot be preempted,(4) Circular Wait Condition: The processes in the system form a circular list or chain where each process in the list is waiting for a resource held by the next process in the list.
- The deadlock conditions can be modeled using a directed graph called a resource allocation graph (RAG) consisting of boxes (resource), circles (process) and edges (request edge and assignment edge). The resource allocation graph helps in identifying the deadlocks.
- There are following approaches to deal with the problem of deadlock: (1) The Ostrich Approach — stick your head in the sand and ignore the problem, (2)

  Deadlock prevention — prevent deadlock from occurring by eliminating one of the 4 deadlock conditions, (3) Deadlock detection algorithms — detect when deadlock has occurred, (4) Deadlock recovery algorithms — break the deadlock, (5) Deadlock avoidance algorithms — consider resources currently available, resources allocated to each thread, and possible future requests, and only fulfill requests that will not lead to deadlock

There are merits/demerits of each approach. The Ostrich Approach is a good solution if deadlock is not frequent. Deadlock prevention may be overly restrictive. In Deadlock detection and recovery there is a tradeoff between frequency of detection and performance / overhead added, Deadlock avoidance needs too much a priori information and not very dynamic (can't add processes or resources), and involves huge overhead

7.15 Keywords

Deadlock: A deadlock is a situation in which some processes in the system face indefinite delays in resource allocation.

Preemptable resource: A preemptable resource is one that can be taken away from the process with no ill effects.

Nonpreemptable resource: It is one that cannot be taken away from process (without causing ill effect).

Mutual exclusion: several processes cannot simultaneously share a single resource

7.16 SELF-ASSESMENT QUESTIONS (SAQ)

1. What do you understand by deadlock? What are the necessary conditions for deadlock?
2. What do you understand by resource allocation graph (RAG)? Explain using suitable examples, how can you use it to detect the deadlock?
3. What do you mean by pre-emption and non-preemption discuss with an example?
4. Compare and contrast the following policies of resource allocation:

(a) All resources requests together.

(b) Allocation using resource ranking.

(c) Allocation using Banker's algorithm

On the basis of (a) resource idling and (b) overhead of the resource allocation algorithm.

5. How can pre-emption be used to resolve deadlock?
6. Why Banker's algorithm is called so?
7. Under what condition(s) a wait state becomes a deadlock?
8. Explain how mutual exclusion prevents deadlock.
9. Discuss the merits and demerits of each approach dealing with the problem of deadlock.
10. Differentiate between deadlock avoidance and deadlock prevention.
11. A system contains 6 units of a resource, and 3 processes that need to use this resource. If the maximum resource requirement of each process is 3 units, will the system be free of deadlocks for all time? Explain clearly.

    If the system had 7 units of the resource, would the system be deadlock-free?

7.17 Answer to check your progress

1.all of the mentioned

2.the system can allocate resources to each process in some order and still avoid a deadlock

3.starvation

4.there must be a fixed number of resources to allocate

5.system calls

6.All of the mentioned

7.to ensure that at least one of the necessary conditions cannot hold

8.resource allocation state

9.then the system will be in a safe state

10. frequently

7.18 SUGGESTED READINGS / REFERENCE MATERIAL

1. Operating System Concepts, 5th Edition, Silberschatz A., Galvin P.B., John Wiley and Sons.
2. Systems Programming and Operating Systems, 2nd Revised Edition, Dhamdhere D.M., Tata McGraw Hill Publishing Company Ltd., New Delhi.
3. Operating Systems, Madnick S.E., Donovan J.T., Tata McGraw Hill Publishing Company Ltd., New Delhi.
4. Operating Systems-A Modern Perspective, Gary Nutt, Pearson Education Asia, 2000.
5. Operating Systems, Harris J.A., Tata McGraw Hill Publishing Company Ltd., New Delhi, 2002.
6. Operating Systems, A Concept-based Approach, Dhamdhere D.M., Tata McGraw Hill Publishing Company Ltd., New Delhi.

Course: MCA-32	Writer: Ritu
Lesson: 8	Vetter:

Distributed Operating System

Structure

8.0 Learning Objectives

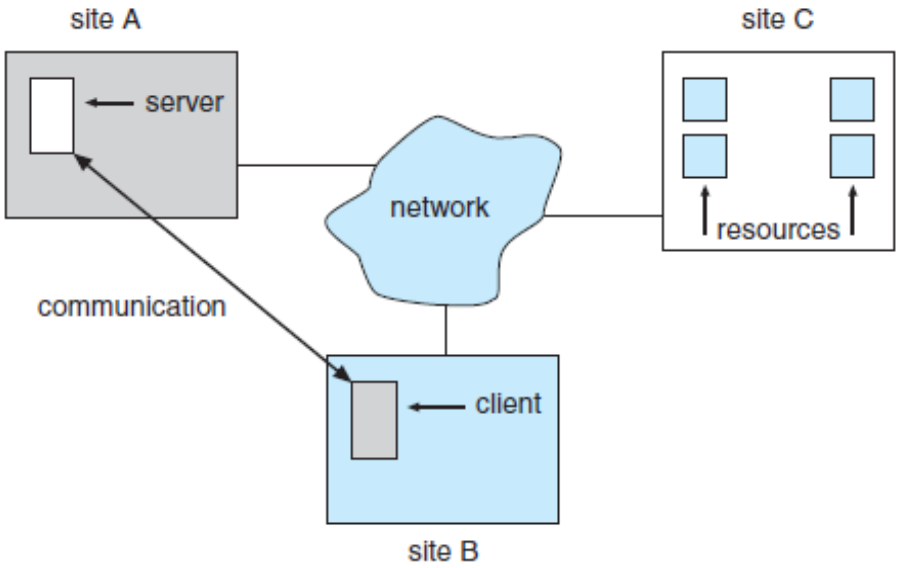
- Explain the advantages of networked and distributed systems.
- Provide a high-level overview of the networks that interconnect distributed systems
- Define the roles and types of distributed systems in use today.
- Discuss issues concerning the design of distributed file systems.

8.1 Introduction

A distributed system is a collection of processors that do not share memory or a clock. Instead, each node has its own local memory. The nodes communicate with one another through various networks,

such as high-speed buses. Distributed systems are more relevant than ever, and you have almost certainly used some sort of distributed service. Applications of distributed systems

range from providing transparent access to files inside an organization, to large-scale cloud file and photo storage services, to business analysis of trends on large data sets, to parallel processing of scientific data, and more.



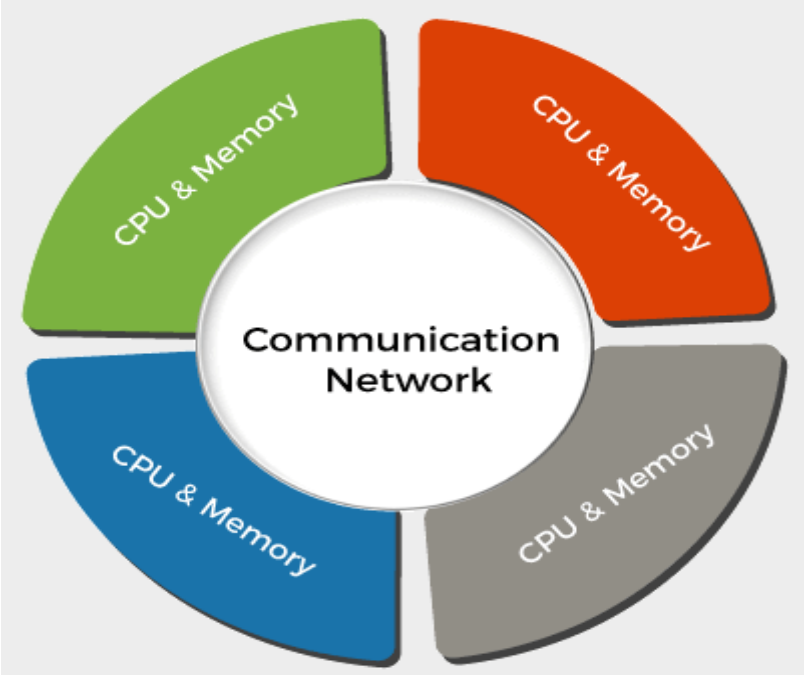
In fact, the most basic example of a distributed system is one we are all likely very familiar with—the Internet. In this chapter, we discuss the general structure of distributed systems and the networks that interconnect them. We also contrast the main differences in the types and roles of current distributed system designs. Finally, we investigate some of the basic designs and design challenges of distributed file systems.

8.2 Distributed Operating System

A distributed operating system **(DOS)** is an essential type of operating system. Distributed systems use many central processors to serve multiple real-time applications and users. As a result, data processing jobs are distributed between the processors.

It connects multiple computers via a single communication channel. Furthermore, each of these systems has its own processor and memory. Additionally, these **CPUs** communicate via high speed buses or

telephone lines. Individual systems that communicate via a single channel are regarded as a single entity. They're also known as loosely coupled systems.



This operating system consists of numerous computers, nodes, and sites joined together via LAN/WAN lines. It enables the distribution of full systems on a couple of center processors, and it supports many real-time products and different users. Distributed operating systems can share their computing resources and I/O files while providing users with virtual machine abstraction.

8.2.1 Types of Distributed Operating System

There are various types of Distributed Operating systems. Some of them are as follows:

1. **Client-Server Systems**
2. **Peer-to-Peer Systems**
3. **Middleware**
4. **Three-tier**
5. **N-tier**

8.2.1.1 Client-Server System

This type of system requires the client to request a resource, after which the server gives the requested resource. When a client connects to a server, the server may serve multiple clients at the same time. Client-Server Systems are also referred to as "Tightly Coupled Operating Systems". This system is primarily intended for multiprocessors and homogenous multicomputer. Client-Server Systems function as a centralized server since they approve all requests issued by client systems.

Server systems can be divided into two parts:

1. Computer Server System

This system allows the interface, and the client then sends its own requests to be executed as an action. After completing the activity, it sends a back response and transfers the result to the client.

2. File Server System

It provides a file system interface for clients, allowing them to execute actions like file creation, updating, deletion, and more.

8.2.1.2 Peer-to-Peer System

The nodes play an important role in this system. The task is evenly distributed among the nodes. Additionally, these nodes can share data and resources as needed. Once again, they require a network to connect.

The Peer-to-Peer System is known as a "Loosely Couple System". This concept is used in computer network applications since they contain a large number of processors that do not share memory or clocks. Each processor has its own local memory, and they interact with one another via a variety of communication methods like telephone lines or high-speed buses.

8.2.1.3 Middleware

Middleware enables the interoperability of all applications running on different operating systems. Those programs are capable of transferring all data to one other by using these services.

8.2.1.4 Three-tier

The information about the client is saved in the intermediate tier rather than in the client, which simplifies development. This type of architecture is most commonly used in online applications.

8.2.1.5 N-tier

When a server or application has to transmit requests to other enterprise services on the network, n-tier systems are used.

8.2.2 Features of Distributed Operating Systems

There are various features of the distributed operating system. Some of them are as follows:

Openness

It means that the system's services are freely displayed through interfaces. Furthermore, these interfaces only give the service syntax. For example, the type of function, its return type, parameters, and so on. Interface Definition Languages are used to create these interfaces (IDL).

Scalability

It refers to the fact that the system's efficiency should not vary as new nodes are added to the system. Furthermore, the performance of a system with 100 nodes should be the same as that of a system with 1000 nodes.

Resource Sharing

Its most essential feature is that it allows users to share resources. They can also share resources in a secure and controlled manner. Printers, files, data, storage, web pages, etc., are examples of shared resources.

Flexibility

A DOS's flexibility is enhanced by modular qualities and delivers a more advanced range of high-level services. The kernel/ microkernel's quality and completeness simplify the implementation of such services.

Transparency

It is the most important feature of the distributed operating system. The primary purpose of a distributed operating system is to hide the fact that resources are shared. Transparency also implies that the user should be unaware that the resources he is accessing are shared. Furthermore, the system should be a separate independent unit for the user.

Heterogeneity

The components of distributed systems may differ and vary in operating systems, networks, programming languages, computer hardware, and implementations by different developers.

Fault Tolerance

Fault tolerance is that process in which user may continue their work if the software or hardware fails.

Examples of Distributed Operating System

There are various examples of the distributed operating system. Some of them are as follows:

Solaris

It is designed for the SUN multiprocessor workstations

OSF/1

It's compatible with Unix and was designed by the Open Foundation Software Company.

Micros

The MICROS operating system ensures a balanced data load while allocating jobs to all nodes in the system.

DYNIX

It is developed for the Symmetry multiprocessor computers.

Locus

It may be accessed local and remote files at the same time without any location hindrance.

Mach

It allows the multithreading and multitasking features.

8.2.3 Applications of Distributed Operating System

There are various applications of the distributed operating system. Some of them are as follows:

Network Applications

DOS is used by many network applications, including the Web, peer-to-peer networks, multiplayer web-based games, and virtual communities.

Telecommunication Networks

DOS is useful in phones and cellular networks. A DOS can be found in networks like the Internet, wireless sensor networks, and routing algorithms.

Parallel Computation

DOS is the basis of systematic computing, which includes cluster computing and grid computing, and a variety of volunteer computing projects.

Real-Time Process Control

The real-time process control system operates with a deadline, and such examples include aircraft control systems.

8.2.4 Advantages and Disadvantages of Distributed Operating System

There are various advantages and disadvantages of the distributed operating system. Some of them are as follows:

Advantages

There are various advantages of the distributed operating system. Some of them are as follow:

1. It may share all resources (CPU, disk, network interface, nodes, computers, and so on) from one site to another, increasing data availability across the entire system.
2. It reduces the probability of data corruption because all data is replicated across all sites; if one site fails, the user can access data from another operational site.
3. The entire system operates independently of one another, and as a result, if one site crashes, the entire system does not halt.
4. It increases the speed of data exchange from one site to another site.
5. It is an open system since it may be accessed from both local and remote locations.
6. It helps in the reduction of data processing time.
7. Most distributed systems are made up of several nodes that interact to make them fault-tolerant. If a single machine fails, the system remains operational.

Disadvantages

There are various disadvantages of the distributed operating system. Some of them are as follows:

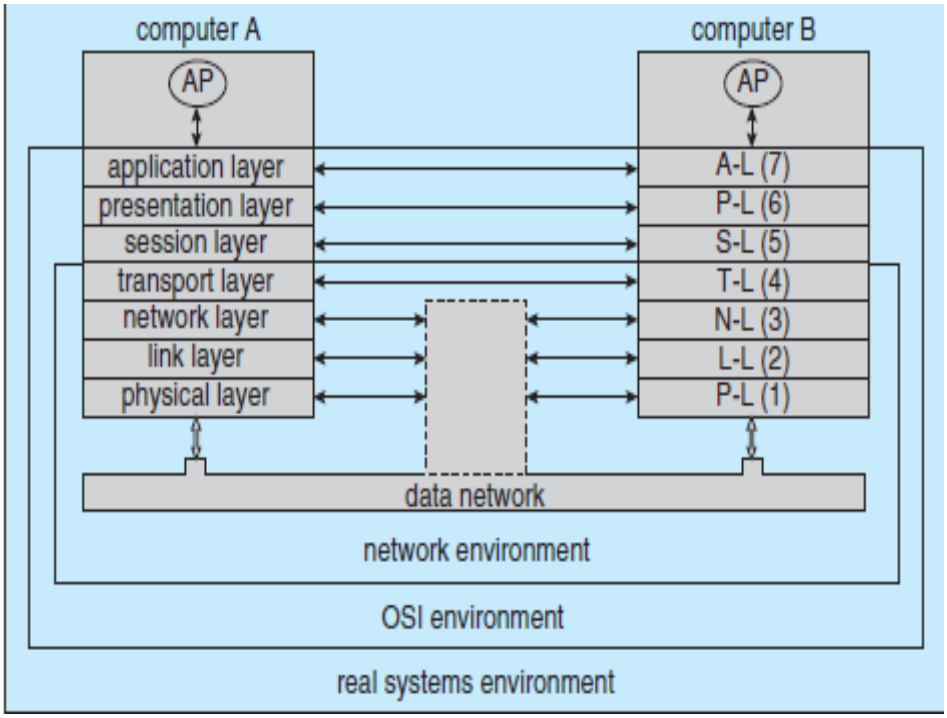
1. The system must decide which jobs must be executed when they must be executed, and where they must be executed. A scheduler has limitations, which can lead to underutilized hardware and unpredictable runtimes.
2. It is hard to implement adequate security in DOS since the nodes and connections must be secured.
3. The database connected to a DOS is relatively complicated and hard to manage in contrast to a single-user system.
4. The underlying software is extremely complex and is not understood very well compared to other systems.
5. The more widely distributed a system is, the more communication latency can be expected. As a result, teams and developers must choose between availability, consistency, and latency.
6. These systems aren't widely available because they're thought to be too expensive.
7. Gathering, processing, presenting, and monitoring hardware use metrics for big clusters can be a real issue.

8.3 Communication Protocols

When we are designing a communication network, we must deal with the inherent complexity of coordinating asynchronous operations communicating in a potentially slow and error-prone environment. In addition, the systems on the network must agree on a protocol or a set of protocols for determining host names, locating hosts on the network, establishing connections, and so on. We can simplify the design problem (and related implementation) by partitioning the problem into multiple layers. Each layer on one system communicates with the equivalent layer on other systems. Typically, each layer has its own protocols, and communication takes place between peer layers using a specific protocol. The protocols may be implemented in hardware or

software. For instance, Figure 19.5 shows the logical communications between two computers, with the three lowest-level layers implemented in hardware. The International Standards Organization created the Open Systems Interconnection (OSI) model for describing the various layers of networking. While these layers are not implemented in practice, they are useful for understanding how networking logically works, and we describe them below:

• **Layer 1: Physical layer**. The physical layer is responsible for handling both the mechanical and the electrical details of the physical transmission of a bit stream. At the physical layer, the communicating systems must agree on the electrical representation of a binary 0 and 1, so that when data are sent as a stream of electrical signals, the receiver is able to interpret the data



Two computers communicating via the OSI network model.

properly as binary data. This layer is implemented in the hardware of the networking device. It is responsible for delivering bits.

Layer 2: Data-link layer the data-link layer is responsible for handling *frames*, or fixed-length parts of packets, including any error detection and recovery that occur in the physical layer. It sends frames between physical addresses.

Layer 3: Network layer The network layer is responsible for breaking messages into packets, providing connections between logical addresses, and routing packets in the communication network, including handling the addresses of outgoing packets, decoding the addresses of incoming packets, and maintaining routing information for proper response to changing

load levels. Routers work at this layer.

Layer 4: Transport layer the transport layer is responsible for transfer of messages between nodes, maintaining packet order, and controlling flow to avoid congestion.

Layer 5: Session layer the session layer is responsible for implementing sessions, or process-to-process communication protocols.

Layer 6: Presentation layer the presentation layer is responsible for resolving the differences in formats among the various sites in the network, including character conversions and half duplex–full duplex modes(character echoing).

Layer 7: Application layer the application layer is responsible for interacting directly with users. This layer deals with file transfer, remote-login protocols, and electronic mail, as well as with schemas for distributed databases.

Figure summarizes the **OSI protocol stack**—a set of cooperating protocols—showing the physical flow of data. As mentioned, logically each layer of a protocol stack communicates with the equivalent layer on other systems. But physically, a message starts at or above the application layer and is passed through each lower level in turn. Each layer may modify the message and

include message-header data for the equivalent layer on the receiving side. Ultimately, the message reaches the data-network layer and is transferred as one or more packets (Figure). The data-link layer of the target system receives these data, and the message is moved up through the protocol stack. It is analyzed, modified, and stripped of headers as it progresses. It finally reaches

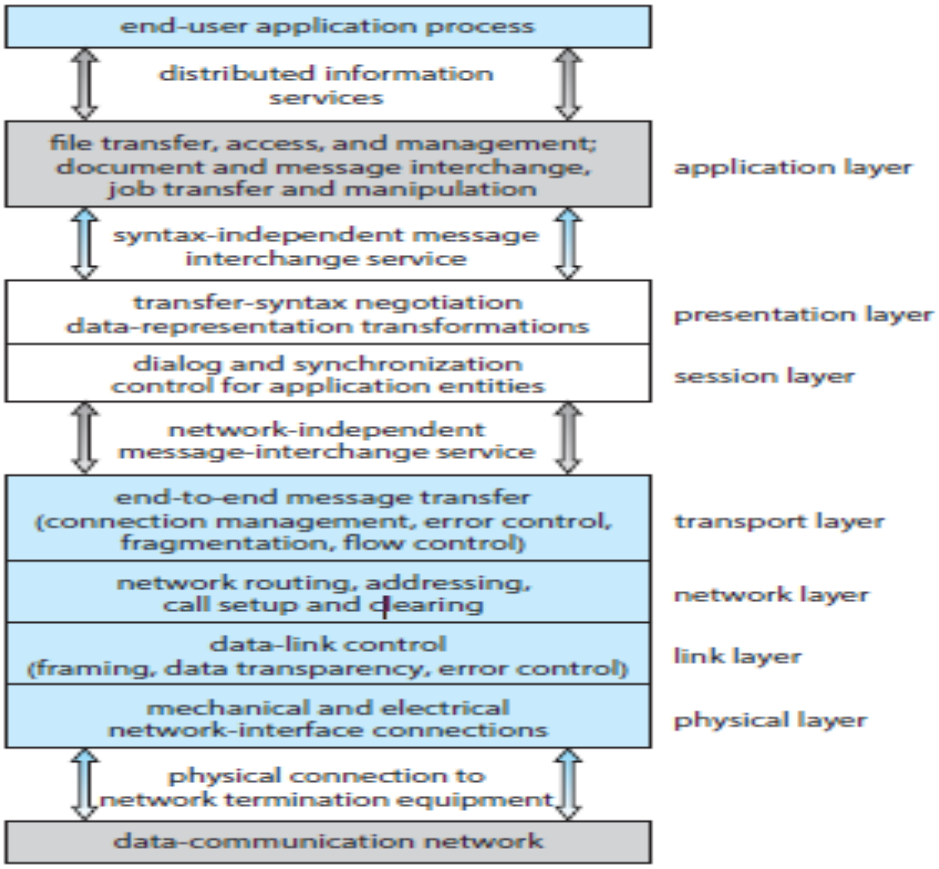
the application layer for use by the receiving process.

The OSI model formalizes some of the earlier work done in network protocols but was developed in the late 1970s and is currently not in widespread use. Perhaps the most widely adopted protocol stack is the TCP/IP model (sometimes called the *Internet model*), which has been adopted by virtually all Internet sites.

The TCP/IP protocol stack has fewer layers than the OSI model. Theoretically, because it combines several functions in each layer, it is more difficult to implement but more efficient than OSI networking. The relationship between the OSI and TCP/IP models is shown in Figure

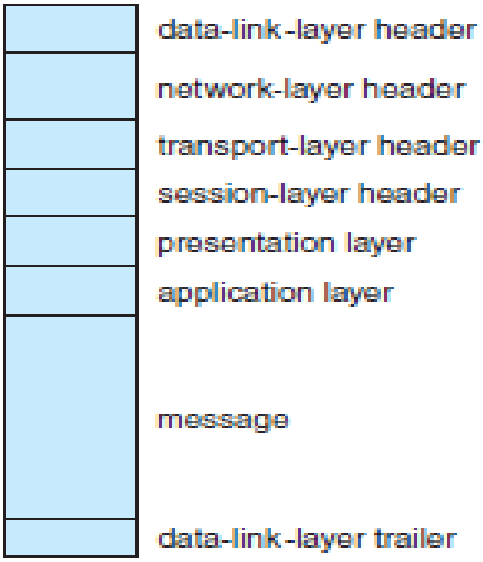
The TCP/IP application layer identifies several protocols in widespread use in the Internet, including HTTP, FTP, SSH, DNS, and SMTP. The transport layer identifies the unreliable, connectionless **user**

datagram protocol (**UDP**) and the reliable, connection-oriented **transmission control protocol** (**TCP**). The **Internet protocol** (**IP**) is responsible for routing IP **datagrams**, or packets, through the Internet. The TCP/IP model does not formally identify a link or physicallayer, allowing TCP/IP traffic to run across any physical network. In this section the TCP/IP model running over an Ethernet network.

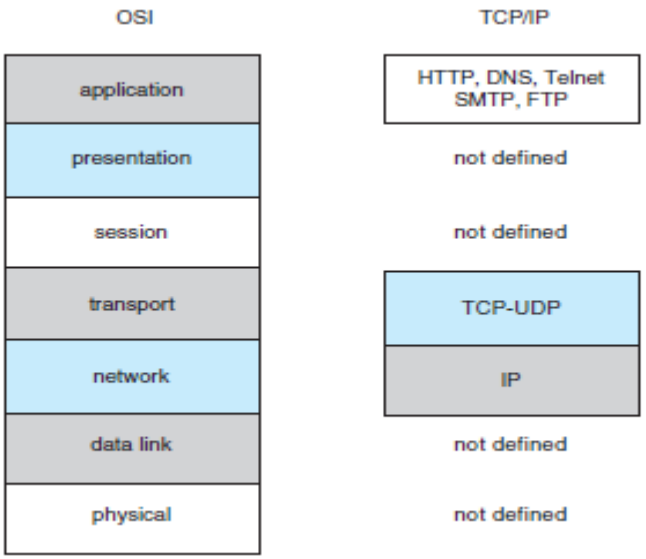


The OSI protocol stack

Security should be a concern in the design and implementation of any modern communication protocol. Both strong authentication and encryption are needed for secure communication. Strong authentication ensures that the sender and receiver of a communication are who or what they are supposed to be. Encryption protects the contents of the communication from eavesdropping. Weak authentication and clear-text communication are still very common, however, for a variety of reasons. When most of the common protocols were designed, security was frequently less important than performance, simplicity, and efficiency.



This legacy is still showing itself today, as adding security to existing infrastructure is proving to be difficult and complex. Strong authentication requires a multistep handshake protocol or authentication devices, adding complexity to a protocol. As to the encryption requirement, modern CPUs can efficiently perform encryption, frequently including cryptographic acceleration instructions so system performance is not compromised. Long-distance communication can be made secure by authenticating



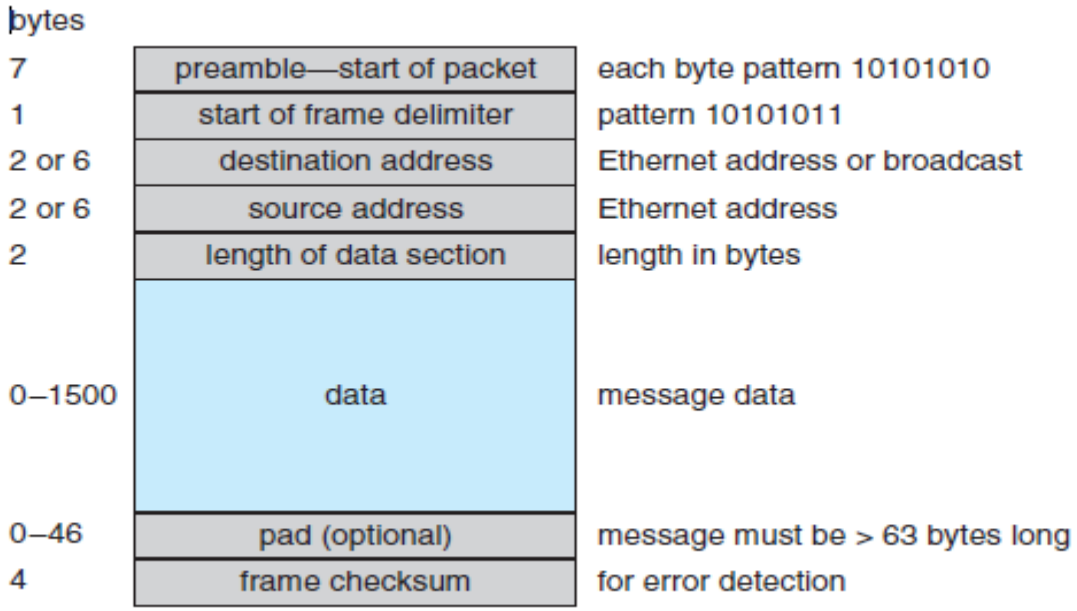
The OSI and TCP/IP protocol stacks.

the endpoints and encrypting the stream of packets in a virtual private network, which includes strong native authentication and encryption, should help improve even LAN security.TCP/IP Example Next, we address name resolution and examine its operation with respect to the TCP/IP protocol stack on the Internet. Then we consider the processing needed to transfer a packet between hosts on different Ethernet networks. We base our description on the IPV4 protocols, which are the type most commonly used today.

In a TCP/IP network, every host has a name and an associated IP address (or host-id). Both of these strings must be unique; and so that the name space can be managed, they are segmented. As described earlier, the name is hierarchical, describing the host name and then the organization with which the host is associated. The host-id is split into a network number and a host number. The proportion of the split varies, depending on the size of the network. Once the Internet administrators assign a network number, the site with that number is free to assign host-ids.

The sending system checks its routing tables to locate a router to send the frame on its way. This routing table is either configured manually by the system administrator or is populated by one of several routing protocols, such as the Border Gateway Protocol (BGP). The routers use the network part of the hosted to transfer the packet from its source network to the destination network. The destination system then receives the packet. The packet may be a complete message, or it may just be a component of a message, with more packets needed before the message can be reassembled and passed to the TCP/UDP (transport)layer for transmission to the destination process. Within a network, how does a packet move from sender (host or router) to receiver? Every Ethernet device has a unique byte number, called the **medium access control** (**MAC**) **address**, assigned to it for addressing. Two devices on a LAN communicate with each other only with this number. If a system needs to send data to another system, the networking software generates an **address resolution protocol** (**ARP**) packet containing the IP address of the destination system. This packet is **broadcast** to all other systems on that Ethernet network. A broadcast uses a special network address (usually, the maximum address) to signal that all hosts should receive and process the packet. The broadcast is not re-sent by routers in between different networks, so only systems on the local network receive it. Only the system whose IP address matches the IP address of the ARP request responds and sends back its MAC address to the system that initiated the query. For efficiency, the host caches the IP–MAC address pair in an internal table. The cache

entries are aged, so that an entry is eventually removed from the cache if an access to that system is not required within a given time. In this way, hosts that are removed from a network are eventually forgotten. For added performance, ARP entries for heavily used hosts may be pinned in the ARP cache. Once an Ethernet device has announced its host-id and address, communication can begin. A process may specify the name of a host with which to communicate. Networking software takes that name and determines the IPaddress of the target, using a DNS lookup or an entry in a local hosts file



where translations can be manually stored. The message is passed from the application layer, through the software layers, and to the hardware layer. At the hardware layer, the packet has the Ethernet address at its start; a trailer indicates the end of the packet and contains a checksum for detection of packet damage (Figure 19.9). The packet is placed on the network by the Ethernet

device. The data section of the packet may contain some or all of the data of the original message, but it may also contain some of the upper-level headers that compose the message. In other words, all parts of the original message must be sent from source to destination, and all headers above the 802.3 layer(data-link layer) are included as data in the Ethernet packets. If the destination is on the same local network as the source, the system can look in its ARP cache, find the Ethernet address of the host, and place the packet on the wire. The destination Ethernet device then sees its address in the packet and reads in the packet, passing it up the protocol stack. If the destination system is on a network different

from that of the source, the source system finds an appropriate router on its network and sends the packet there. Routers then pass the packet along the WAN until it reaches its destination network. The router that connects the destination network check sits ARP cache, finds the Ethernet number of the destination, and sends the packet to that host. Through all of these transfers, the data-link-layer header may change as the Ethernet address of the next router in the chain is used, but the other headers of the packet remain the same until the packet is received and processed by the protocol stack and finally passed to the receiving process by the kernel.

8.3.1 Transport Protocols UDP and TCP

Once a host with a specific IP address receives a packet, it must somehow pass it to the correct waiting process. The transport protocols TCP and UDP identify the receiving (and sending) processes through the use of a **port number**. Thus, a host with a single IP address can have multiple server processes running and waiting for packets as long as each server process specifies a different port number.

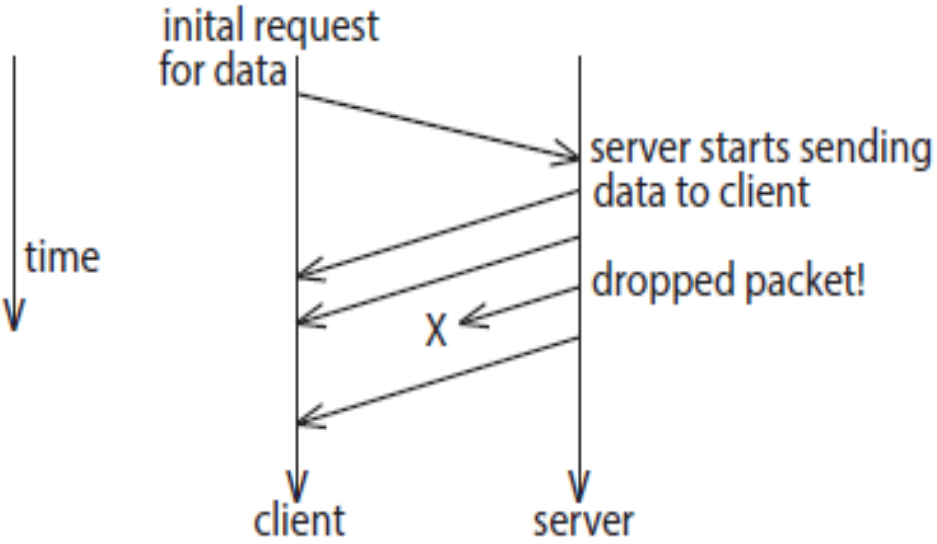
By default, many common services use *well-known* port numbers. Some examples include FTP (21), SSH (22), SMTP (25), and HTTP (80). For example, if you wish to connect to an "http" website through your web browser, your browser will automatically attempt to connect to port 80 on the server by using the number 80 as the port number in the TCP transport header. For an extensive list of well-known ports, log into your favorite Linux or UNIX machine and take a look at the file /etc/services. The transport layer can accomplish more than just connecting a network packet to a running process. It can also, if desired, add reliability to a network packet stream. To explain how, we next outline some general behavior of the transport protocols UDP and TCP.

8.3.1.1 User Datagram Protocol

The transport protocol UDP is *unreliable* in that it is a bare-bones extension to IP with the addition of a port number. In fact, the UDP header is very simple and contains only four fields: source port number, destination port number, length, and checksum. Packets may be sent quickly to a destination using UDP. However, since there are no guarantees of delivery in the lower layers of the network stack, packets may become lost.

Packets can also arrive at the receiver out of order. It is up to the application to figure out these error cases and to adjust (or not adjust).Figure illustrates a common scenario involving loss of a packet between a client and a server using the UDP protocol. Note that this protocol is known as a

connectionless protocol because there is no connection setup at the beginning of the transmission to set up state—the client just starts sending data. Similarly, there is no connection tear down. The client begins by sending some sort of request for information to the server. The server then responds by sending four datagrams, or packets, to the client. Unfortunately, one of the packets is dropped by an overwhelmed router. The client must either make do with only three packets or use logic programmed into the application to request the missing packet. Thus, we need to use a different transport protocol if we want any additional reliability guarantees to be handled by the network.



Example of a UDP data transfer with dropped packet.

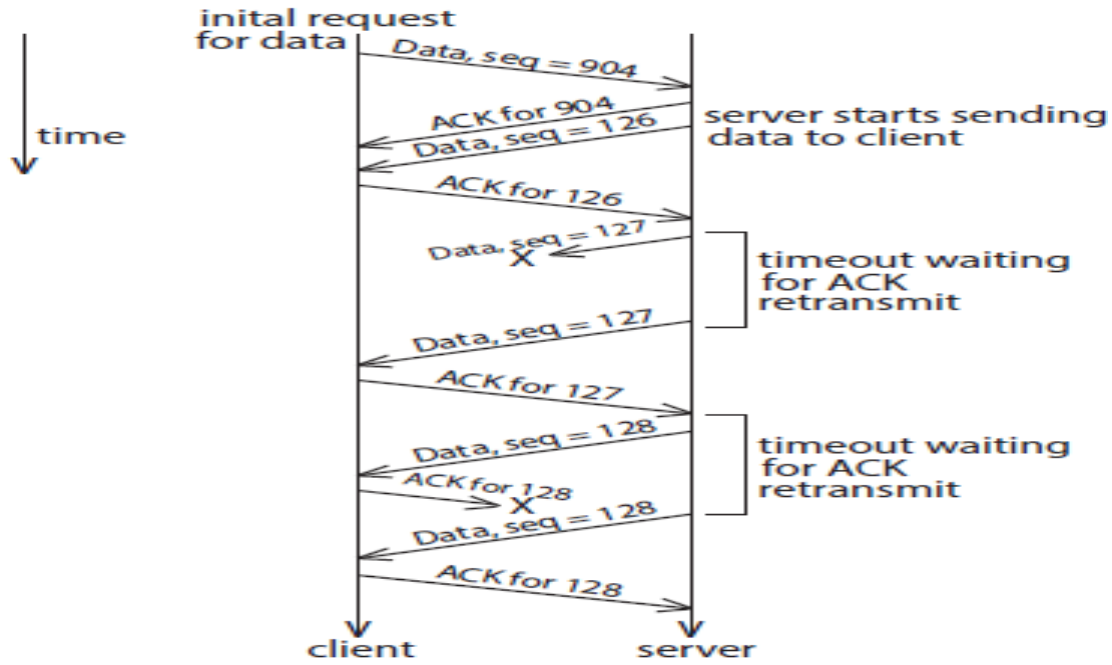
8.3.1.2 Transmission Control Protocol

TCP is a transport protocol that is both reliable and connection-oriented. In addition to specifying port numbers to identify sending and receiving processes on different hosts, TCP provides an abstraction that allows a sending process on one host to send an in-order, uninterrupted byte stream across the network to a receiving process on another host. It accomplishes these things through the following mechanisms:

- Whenever a host sends a packet, the receiver must send an **acknowledgment packet**, or ACK, to notify the sender that the packet was received. If the ACK is not received before a timer expires, the sender will send that packet again.

- TCP introduces **sequence numbers** into the TCP header of every packet. These numbers allow the receiver to (1) put packets in order before sending data up to the requesting process and (2) be aware of packets missing from the byte stream.
- TCP connections are initiated with a series of control packets between the sender and the receiver (often called a *three-way handshake*) and closed gracefully with control packets responsible for tearing down the connection. These control packets allow both the sender and the receiver to set up and remove state. In Figure demonstrates a possible exchange using TCP (with connection setup and tear-down omitted). After the connection has been established, the client sends a request packet to the server with the sequence number 904.Unlike the server in the UDP example, the server must then send an ACK packet back to the client. Next, the server starts sending its own stream of data packets starting with a different sequence number. The client sends an ACK packet for each data packet it receives. Unfortunately, the data packet with the sequence number 127 is lost, and no ACK packet is sent by the client. The sender timeout waiting for the ACK packet, so it must resend data packet 127. Later in the connection, the server sends the data packet with the sequence number128, but the ACK is lost. Since the server does not receive the ACK it must resend data packet 128. The client then receives a duplicate packet. Because the client knows that it previously received a packet with that sequence number, it throws the duplicate away. However, it must send another ACK back to the server to allow the server to continue.

In the actual TCP specification, an ACK isn't required for each and every packet. Instead, the receiver can send a cumulative ACK to ACK a series of packets. The server can also send numerous data packets sequentially before waiting for ACKs, to take advantage of network throughput.TCP also helps regulate the flow of packets through mechanisms called flow control and congestion control. Flow control involves preventing the sender from overrunning the capacity of the receiver.



For example, the receiver may Example of a TCP data transfer with dropped packets. have a slower connection or may have slower hardware components (like as lower network card or processor). Flow-control state can be returned in the ACK packets of the receiver to alert the sender to slow down or speed up. **Congestion control** attempts to approximate the state of the networks (and generally the routers) between the sender and the receiver. If a router becomes overwhelmed with packets, it will tend to drop them.

Dropping packets results in ACK timeouts, which results in more packets saturating the network. To prevent this condition, the sender monitors the connection for dropped packets by noticing how many packets are not acknowledged. If there are too many dropped packets, the sender will slow down the rate at which it sends them. This helps ensure that the TCP connection is being fair to other connections happening at the same time. By utilizing a reliable transport protocol like TCP, a distributed system does not need extra logic to deal with lost or out-of-order packets. However, TCP is slower than UDP.

8.4 Network and Distributed Operating Systems

In this section, we describe the two general categories of network-oriented operating systems: network operating systems and distributed operating sys terms. Network operating systems are simpler to

implement but generally more difficult for users to access and use than are distributed operating systems, which provide more features.

8.4.1 Network Operating Systems

A network operating system provides an environment in which users can access remote resources (implementing resource sharing) by either logging in to the appropriate remote machine or transferring data from the remote machine to their own machines. Currently, all general-purpose operating systems, and even embedded operating systems such as Android and iOS, are network operating systems.

8.4.1.1 Remote Login

An important function of a network operating system is to allow users to log in remotely. The Internet provides the ssh facility for this purpose. To illustrate, suppose that a user at Westminster College wishes to compute onkristen.cs.yale.edu, a computer located at Yale University. To do so, the user must have a valid account on that machine. To log in remotely, the user issues the command ssh kristen.cs.yale.edu This command results in the formation of an encrypted socket connection between the local machine at Westminster College and the kristen.cs.yale.edu computer.

After this connection has been established, the networking software creates a transparent, bidirectional link so that all characters entered by the user are sent to a process on kristen.cs.yale.eduand all the output from that process is sent back to the user. The process on the remote machine asks the user for a login name and a password. Once the correct information has been received, the process acts as a proxy for the user, who can compute on the remote machine just as any local user can.

8.4.1.2 Remote File Transfer

Another major function of a network operating system is to provide a mechanism for remote fil transfer from one machine to another. In such an environment, each computer maintains its own local file system. If a user at onsite (say, Kurt at albion.edu) wants to access a file owned by Becca located on another computer (say, at colby.edu), then the file must be copied explicitly from the computer at Colby in Maine to the computer at Albion in Michigan. The communication is one-directional and individual, such that other user sat those sites wishing to transfer a file, say Sean at colby.edu to Karen atalbion.edu, must likewise issue a set of commands. The Internet provides a mechanism for such a transfer with the file transfer protocol (FTP) and the more private secure file transfer protocol (SFTP).

Suppose that user Carla at wesleyan.edu wants to copy a file that is owned by Owen at kzoo.edu. The user must first invoke the sftp program by executingsftp owen@kzoo.edu The program then asks the user for a login name and a password. Once thecorrect information has been received, the user can use a series of commands to upload files, download files, and navigate the remote file system structure. Some of these commands are:

• get—Transfer a file from the remote machine to the local machine.

• put—Transfer a file from the local machine to the remote machine.

• ls or dir—List files in the current directory on the remote machine.

• cd—Change the current directory on the remote machine.

There are also various commands to change transfer modes (for binary or ASCII

files) and to determine connection status.

8.4.1.3 Cloud Storage

Basic cloud-based storage applications allow users to transfer files much as with FTP. Users can upload files to a cloud server, download files to the local computer, and share files with other cloud-service users via a web link or other sharing mechanism through a graphical interface. Common examples include Dropbox and Google Drive. An important point about SSH, FTP, and cloud-based storage applications is that they require the user to change paradigms. FTP, for example, requires the user to know a command set entirely different from the normal operating system commands. With SSH, the user must know appropriate commands on the remote system. For instance, a user on a Windows machine who connects remotely to a UNIX machine must switch to UNIX commands for the duration of the SSH session. In networking, a **session**s a complete round of communication, frequently beginning with a login to authenticate and ending with a logoff to terminate the communication.)With cloud-based storage applications, users may have to log into the cloud service (usually through a web browser) or native application and then use a series of graphical commands to upload, download, or share files. Obviously, users would find it more convenient not to be required to use a different set of commands. Distributed operating systems are designed to address this problem.

8.4.2 Distributed Operating Systems

In a distributed operating system, users access remote resources in the someway they access local resources. Data and process migration from one site to another is under the control of the distributed operating system. Depending on the goals of the system, it can implement data migration, computation migration, process migration, or any combination thereof.

8.4.2.1 Data Migration

Suppose a user on site A wants to access data (such as a file) that reside at site B. The system can transfer the data by one of two basic methods. One approach to **data migration**s to transfer the entire file to site A. From that point on, all access to the file is local. When the user no longer needs access to the file, copy of the file (if it has been modified) is sent back to site B. Even if only a modest change has been made to a large file, all the data must be transferred. This mechanism can be thought of as an automated FTP system.

This approach was used in the Andrew file system, but it was found to be too inefficient. The other approach is to transfer to site A only those portions of the file that are actually *necessary* for the immediate task. If another portion is required later, another transfer will take place. When the user no longer wants to access the file, any part of it that has been modified must be sent back to site B. (Note the similarity to demand paging.) Most modern distributed systems use this approach.

Whichever method is used, data migration includes more than the me retransfer of data from one site to another. The system must also perform various data translations if the two sites involved are not directly compatible (for instance, if they use different character-code representations or represent integers with a different number or order of bits).

8.4.2.2 Computation Migration

In some circumstances, we may want to transfer the computation, rather than the data, across the system; this process is called **computation migration**. For example, consider a job that needs to access various large files that reside at different sites, to obtain a summary of those files. It would be more efficient to access the files at the sites where they reside and return the desired results to the site that initiated the computation. Generally, if the time to transfer the data is longer than the time to execute the remote command, the remote command should be used.

Such a computation can be carried out in different ways. Suppose that process P wants to access a file at site A. Access to the file is carried out at site A and could be initiated by an RPC. An RPC uses network protocols to execute routine on a remote system. Process P invokes a predefined procedure at site A. The procedure executes appropriately and then returns the results to P.Alternatively, process P can send a message to site A. The operating system at site A then creates a new process Q whose function is to carry out the designated task. When process Q completes its execution, it sends the needed result back to P via the message system. In this scheme, process P may execute concurrently with process Q. In fact, it may have several processes running concurrently on several sites.

Either method could be used to access several files (or chunks of files) residing at various sites. One RPC might result in the invocation of another RPCor even in the transfer of messages to another site. Similarly, process Q could, during the course of its execution, send message to another site, which in turn would create another process. This process might either send a message back to Q or repeat the cycle.

8.4.2.3 Process Migration

A logical extension of computation migration is **processes migration**. When process is submitted for execution, it is not always executed at the site at which it is initiated. The entire process, or parts of it, may be executed at different sites. This scheme may be used for several reasons:

- **Load balancing**. The processes (or sub processes) may be distributed across the sites to even the workload.
- **Computation speedup**. If a single process can be divided into a number of sub processes that can run concurrently on different sites or nodes, then the total process turnaround time can be reduced.
- **Hardware preference**. The process may have characteristics that make it more suitable for execution on some specialized processor (such as matrix inversion on a GPU) than on a microprocessor.
- **Software preference**. The process may require software that is available at only a particular site, and either the software cannot be moved, or it is less expensive to move the process.

- **Data access**. Just as in computation migration, if the data being used in the computation are numerous, it may be more efficient to have a process run remotely (say, on a server that hosts a large database) than to transfer all the data and run the process locally.

We use two complementary techniques to move processes in a computer network. In the first, the system can attempt to hide the fact that the process has migrated from the client. The client then need not code her program explicitly to accomplish the migration. This method is usually employed for achieving load balancing and computation speedup among homogeneous systems, as they do not need user input to help them execute programs remotely. The other approach is to allow (or require) the user to specify explicitly how the process should migrate. This method is usually employed when the process must be moved to satisfy a hardware or software preference.

You have probably realized that the World Wide Web has many aspects of a distributed computing environment. Certainly, it provides data migration(between a web server and a web client). It also provides computation migration. For instance, a web client could trigger a database operation on a web server. Finally, with Java, JavaScript, and similar languages, it provides a form of process migration: Java applets and JavaScript scripts are sent from the server to the client, where they are executed. A network operating system provides most of these features, but a distributed operating system makes them seamless and easily accessible. The result is a powerful and easy-to-use facility—one of the reasons for the huge growth of the WorldWideWeb.

8.5 Design Issues in Distributed Systems

The designers of a distributed system must take a number of design challenges into account. The system should be robust so that it can withstand failures. The system should also be transparent to users in terms of both file location and user mobility. Finally, the system should be scalable to allow the addition of more computation power, more storage, or more users. We briefly introduce these issues here. In the next section, we put them in context when we describe the designs of specific distributed file systems.

8.5.1 Robustness

A distributed system may suffer from various types of hardware failure. The failure of a link, a host, or a site and the loss of a message are the most common types. To ensure that the system is robust, we must detect any of these failures, reconfigure the system so that computation can continue, and recover when the failure is repaired. system can be **fault tolerant** in that it can tolerate a certain level of failure

and continue to function normally. The degree of fault tolerance depends on the design of the distributed system and the specific fault. Obviously, more fault tolerance is better. We use the term ***fault tolerance*** in a broad sense.

Communication faults, certain machine failures, storage-device crashes, and decays of storage media should all be tolerated to some extent. **fault-tolerant system** should continue to function, perhaps in a degraded form, when faced with such failures. The degradation can affect performance, functionality, or both. It should be proportional, however, to the failures that caused it. A system that grinds to a halt when only one of its components fails is certainly not fault tolerant. Unfortunately, fault tolerance can be difficult and expensive to implement.

At the network layer, multiple redundant communication paths and network devices such as switches and routers are needed to avoid a communication failure. A storage failure can cause loss of the operating system, applications, or data. Storage units can include redundant hardware components that automatically take over from each other in case of failure. In addition, RAID systems can ensure continued access to the data even in the event of one or more storage device failures (Section 11.8).

8.5.1.1 Failure Detection

In an environment with no shared memory, we generally cannot differentiate among link failure, site failure, host failure, and message loss. We can usually detect only that one of these failures has occurred. Once a failure has-been detected, appropriate action must be taken. What action is appropriate depends on the particular application.

To detect link and site failure, we use a **heartbeat** procedure. Suppose that sites A and B have a direct physical link between them. At fixed intervals, the sites send each other an *I-am-up* message. If site does not receive this message within a predetermined time period, it can assume that site B has failed, that the link between and B has failed, or that the message from B has been lost. At this point, site A has two choices. It can wait for another time period to receive an *I-am-up* message from B, or it can send an *Are-you-up?* message to B. If time goes by and site still has not received an *I-am-up* message, or if site A has sent an *Area-you-up?* message and has not received a reply, the procedure can be repeated. Again, the only conclusion that site A can draw safely is that some type of failure has occurred. Site A can try to differentiate between link failure and site failure by sending an *Are-you-up?* message to B by another route (if one exists). If and when B receives this message, it immediately replies positively. This

positive reply tells A that B is up and that the failure is in the direct link between them. Since we do not know in advance how long it will take the message to travel from A t o Band back, we must use a time-out scheme. At the time A sends the *Are-you-up?* message, it specifies a time interval during which it is willing to wait for the reply from B. If A receives the reply message within that time interval, then it can safely conclude that B is up. If not, however (that is, if a time-out occurs),then A may conclude only that one or more of the following situations has occurred:

- Site B is down.
- The direct link (if one exists) from A to B is down.
- The alternative path from A to B is down.
- The message has been lost. (Although the use of a reliable transport protocol such as TCP should eliminate this concern.)Site A cannot, however, determine which of these events has occurred.

8.5.1.2 Reconfiguration

Suppose that site A has discovered, through the mechanism just described, that a failure has occurred. It must then initiate a procedure that will allow the system to reconfigure and to continue its normal mode of operation.

- If a direct link from Ato B has failed, this information must be broadcast to every site in the system, so that the various routing tables can be updated accordingly.
- If the system believes that a site has failed (because that site can no longer be reached), then all sites in the system must be notified, so that they will no longer attempt to use the services of the failed site. The failure of a site that serves as a central coordinator for some activity (such as deadlock detection) requires the election of a new coordinator. Note that, if the site has not failed (that is, if it is up but cannot be reached), then we may have the undesirable situation in which two sites serve as the coordinator. When the network is partitioned, the two coordinators (each for its own partition) may initiate conflicting actions. For example, if the coordinators are responsible for implementing mutual exclusion, we may have a situation in which two processes are executing simultaneously in their critical sections.

8.5.1.3 Recovery from Failure

When a failed link or site is repaired, it must be integrated into the system gracefully and smoothly.

- Suppose that a link between A and B has failed. When it is repaired, both A and B must be notified. We can accomplish this notification by continuously repeating the heartbeat procedure described in Section 19.5.1.1.
- Suppose that site B has failed. When it recovers, it must notify all other sites that it is up again. Site B then may have to receive information from the other sites to update its local tables. For example, it may need routing table information, a list of sites that are down, undelivered messages, a transaction log of unexecuted transactions, and mail. If the site has not failed but simply cannot be reached, then it still needs this information.

8.5.2 Transparency

Making the multiple processors and storage devices in a distributed system **transparent** to the users has been a key challenge to many designers. Ideally distributed system should look to its users like a conventional, centralized system. The user interface of a transparent distributed system should not distinguish between local and remote resources. That is, users should be able to access remote resources as though these resources were local, and the distributed system should be responsible for locating the resources and for arranging for the appropriate interaction.

Another aspect of transparency is user mobility. It would be convenient to allow users to log into any machine in the system rather than forcing them to use a specific machine. A transparent distributed system facilitates user mobility by bringing over a user's environment (for example, home directory) to wherever he logs in. Protocols like LDAP provide an authentication system for local, remote, and mobile users. Once the authentication is complete, facilities like desktop virtualization allow users to see their desktop sessions at remote facilities.

8.5.3 Scalability

Still another issue is **scalability**—the capability of a system to adapt to increased service load. Systems have bounded resources and can become completely saturated under increased load. For example, with respect to a file system, saturation occurs either when a server's CPU runs at a high utilization rate or when disks' I/O requests overwhelm the I/O subsystem. Scalability is a relative property, but it can be measured accurately A scalable system reacts more gracefully to increased load than does a non-

scalable one. First, its performance degrades more moderately; and second, its resources reach saturated state later. Even perfect design however cannot accommodate an ever-growing load. Adding new resources might solve the problem, but it might generate additional indirect load on other resources (for example, adding machines to a distributed system can clog the network and increase service loads). Even worse, expanding the system can call for expensive design modifications. A scalable system should have the potential to grow without these problems.

In a distributed system, the ability to scale up gracefully is of special importance, since expanding a network by adding new machines or interconnecting two networks is commonplace. In short, a scalable design should withstand high service load, accommodate growth of the user community, and allow simple integration of added resources.

Scalability is related to fault tolerance, discussed earlier. A heavily loaded component can become paralyzed and behave like a faulty component. In addition, shifting the load from a faulty component to that component's backup can saturate the latter. Generally, having spare resources is essential for ensuring reliability as well as for handling peak loads gracefully. Thus, the multiple resources in a distributed system represent an inherent advantage, giving the system a greater potential for fault tolerance and scalability. However, inappropriate design can obscure this potential. Fault-tolerance and scalability considerations call for a design demonstrating distribution of control and data.

Scalability can also be related to efficient storage schemes. For example, many cloud storage providers use **compression** or **deduplication** to cut down on the amount of storage used. *Compression* reduces the size of a file. For example, a zip archive file can be generated out of a file (or files) by executing a zip command, which runs a lossless compression algorithm over the data specified.(***Lossless compression*** allows original data to be perfectly reconstructed from compressed data.) The result is a file archive that is smaller than the uncompressed file.

To restore the file to its original state, a user runs some sort of unzip command over the zip archive file. *Deduplication* seeks to lower data storage requirements by removing redundant data. With this technology, only one instance of data is stored across an entire system (even across data owned by multiple users). Both compression and deduplication can be performed at the file level or the block level, and they can be used together. These technique scan be automatically built into a distributed

system to compress information without users explicitly issuing commands, thereby saving storage space and possibly cutting down on network communication costs without adding user complexity.

8.6 Network Topology

In tech, network topology architecture refers to an overall view of any organization's network infrastructure. The terms network topology and network architecture are often used separately. Let's know what these terms stand for.

Network Topology:

- Network topologies give us an overview of logical and physical network layouts containing links and nodes.
- The **physical topology** refers to the configuration of computers, cables, or other peripherals, etc.
- The **logical topology** allows us to pass information between workstations.
- The different types of network topologies are:
  - Bus topology
  - Mesh topology
  - Star topology
  - Ring topology

Network Architecture:

- The network architecture tells us a detailed picture of resources and network layers.
- In other words, it shows us the overall design of a computer network.
- It presents the logical and structural layout of networking systems and the related hardware devices such as routers, switches, etc.

Here are various types of network topology architectures. We are going to discuss the characteristics of the following network topology architectures:

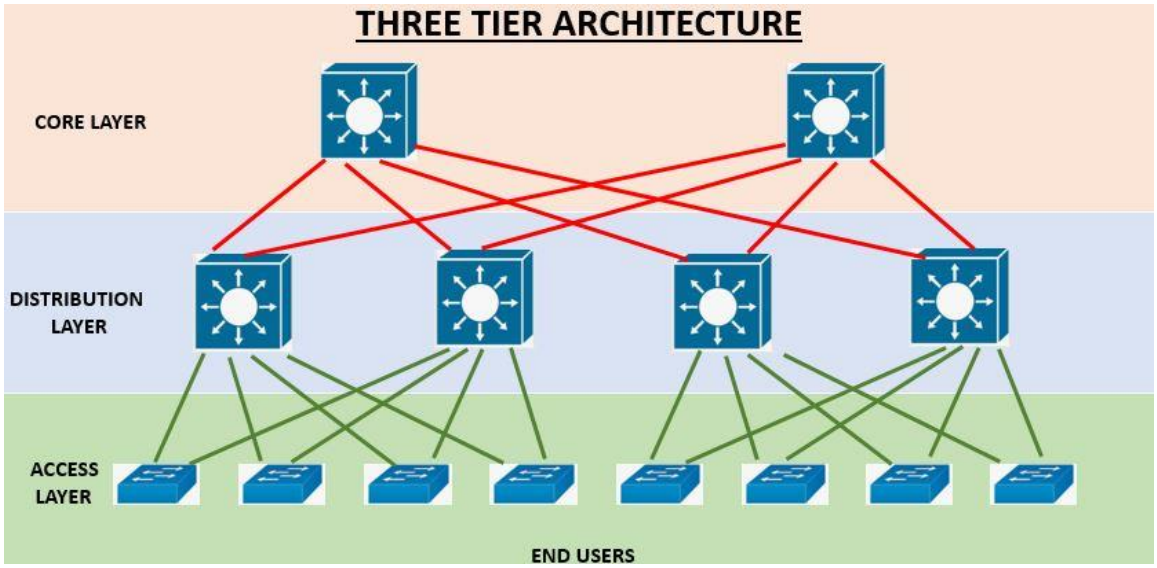
- Three-Tier Architecture
- Two-Tier Architecture

- Spine Leaf Architecture
- WAN Architecture
- SOHO Architecture
- On-Premise/Cloud Architecture

1. Three-Tier Architecture

According to Cisco, networks have been divided into layers or tiers for better understanding. The three-tier architecture is one of the oldest and classic networking models. As the name suggests, the three-tier architecture consists of the following 3 layers:

- Access Layer (bottom layer)
- Distribution Layer (middle layer)
- Core Layer (Topmost layer)



Access Layer:

- The access layer is the lowest layer in the 3-tier architecture.
- It is also called as **workstation layer**.
- It is the closest layer to the end users.
- It consists of **access switches**.

- These switches connect users to the network.

Distribution Layer:

- It is the middle layer in the three-tier architecture.
- The distribution layer is also, sometimes, referred to as the **aggregation layer**.
- It performs quality of service and security work.
- It consists of **multilayer switches**.
- It moves the traffic from the access layer to the core layer.
- It aggregates LAN and WAN links.

Core Layer:

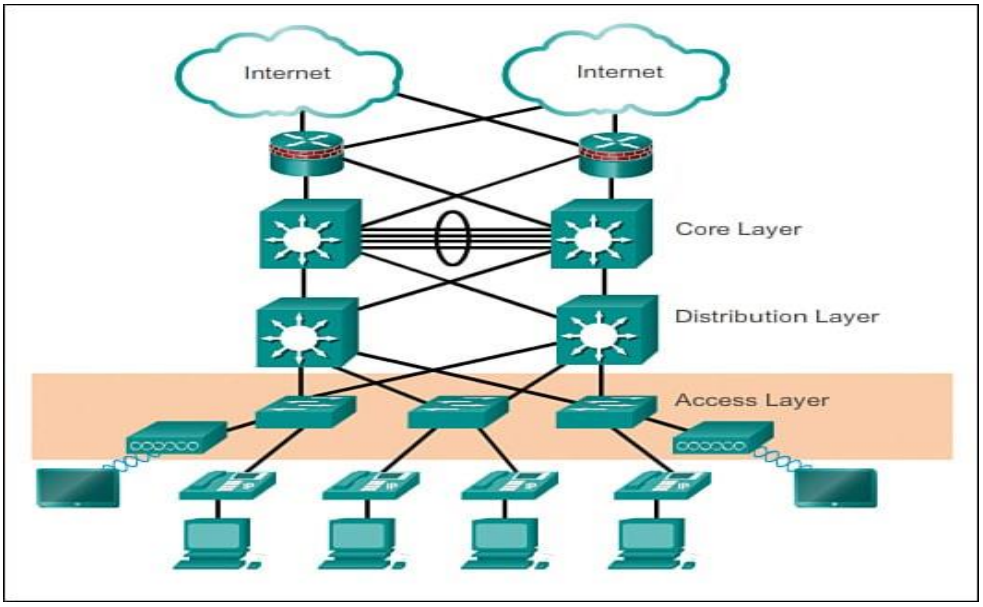
- It is the topmost layer in the three-tier architecture.
- The Core layer also has another name which is the **backbone layer**.
- It connects distribution layer devices.
- It performs high-speed transport of traffic.
- It is reliable and fault-tolerant.

2. Two-Tier Architecture:

The two-tier architecture is more popular architecture than three-tier architecture these days.

- It has a **collapsed core**. It is called so because it has a blended or collapsed distribution layer and core layer.
- Therefore, the two-tier architecture consists of only 2 layers:
    1. Access Layer
    2. Collapsed Core Layer
- It is therefore simpler.

Spine and leaf are the most popular two-tier architecture.



3. Spine Leaf Architecture

- Spine Leaf architecture is a two-layer or two-tier architecture.
- It is mostly used in **data centers**
- It has **low latency**.
- It consists of two layers:
    1. Spine Layer
    2. Leaf Layer

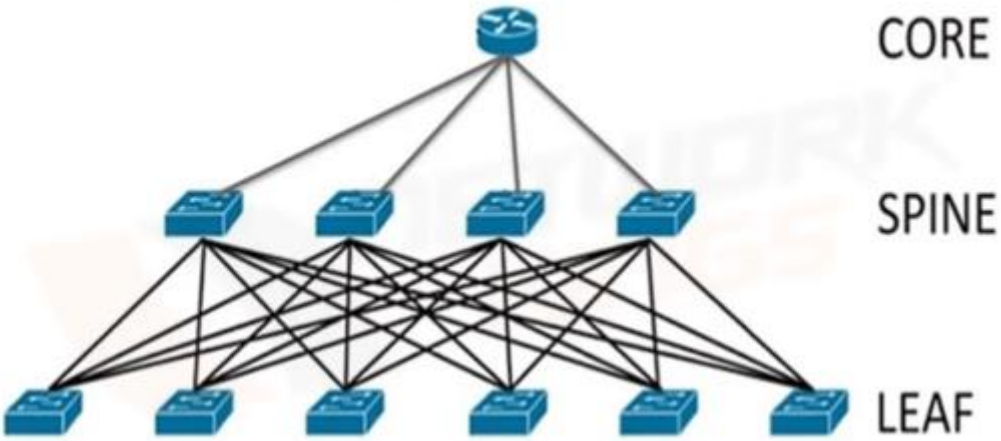
Spine Layer:

- The spine layer is the top layer.
- The Spine layer consists of very intelligent devices such as **Cisco Nexus 9000** devices.
- These devices have ACI Controller intelligence inside them.

Leaf Layer:

- It is the bottom layer in the spine leaf architecture.
- It consists of access switches.

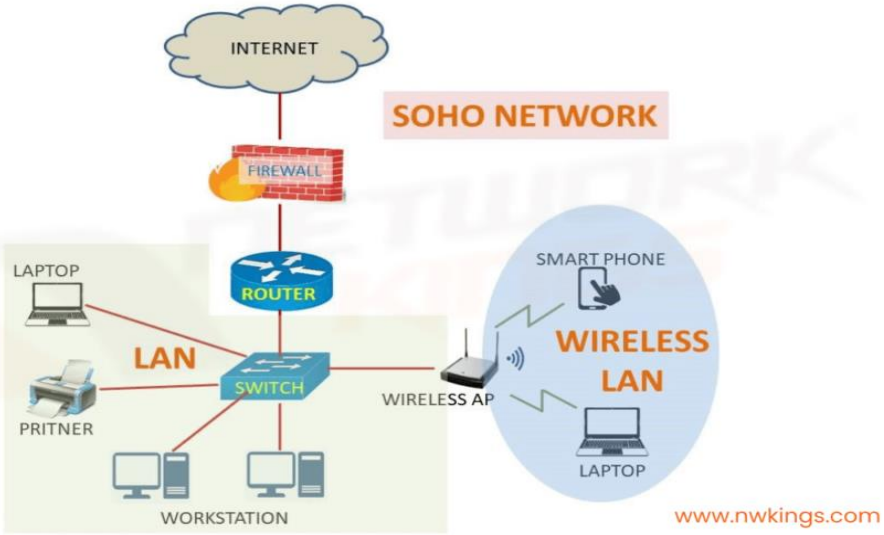
- Each leaf is connected to every spine device.



4. Small Office/Home Office (SOHO) Architecture:

- The SOHO architecture consists of the simplest architecture.
- As the name suggests, it is mostly used in homes and/or small enterprises.
- This type of architecture consists of **three** components:
  1. A small switch
  2. A router
  3. Connected access devices such as printers, PCs, etc.
- Usually, a **single device** is used that acts as both a switch and router.
- The devices are hardwired into this router.

This router also acts as a **firewall**.



5. Wide Area Network (WAN) Architecture:

Imagine this. You have a SOHO network at home and you've connected multiple access points. These access points are making wide-area network connections out to multiple **Internet Service Providers** (ISPs)There are two types of connections formed in the WAN architecture:

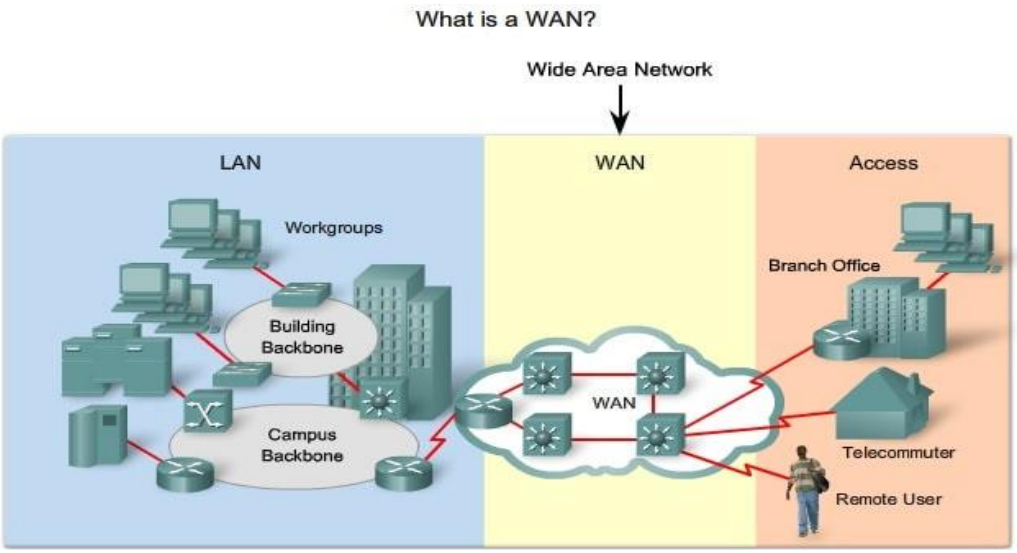
- **The Primary Connection:**

First is the WAN connection formed by the access point using Digital Subscriber Loop (DSL).

- **Emergency Connection:**

The second is a low-cost WAN connection using an asymmetric DSL, meaning there is a difference between download and upload speeds. This is the connection to reach out to cloud resources.

These connections are referred to as **dual-homed configurations**. Such a connection is very strong because if one ISP connection fails, the second one takes the charge.



In today's time, better **client connectivity technologies** are available in the market. These are:

MPLS:

- **Multiprotocol Label Switching (MPLS)** is a transportation technique for high-performance telecommunication networks.
- It transfers data from one network node to another node on short path labels.
- MPLS can carry packets of various network protocols, therefore, called multiprotocol.

Metro-Ethernet:

- Metro Ethernet network is mostly used to connect clients to a large service network.
- Metro Ethernet provides multiple configuration options such as point-2-point, point-2-multipoint, multipoint-2-multipoint, etc.

Internet VPN:

- It consists of the following:
  - Dynamic Multipoint VPN (DMVPN)
  - Site-to-Site VPN
  - Client VPN
- **DMVPN** is dynamic, meaning it can build VPN connections when required and it can break them when not needed.

- Site-to-Site VPN links allow the creation of VPN links when sending protected data over a non-trusted network such as the Internet.
- **Client VPN** allows remote access to corporate resources.

6. On-Premises and Cloud Architecture:

Cloud technologies have developed virtual service models.

- **SaaS:**

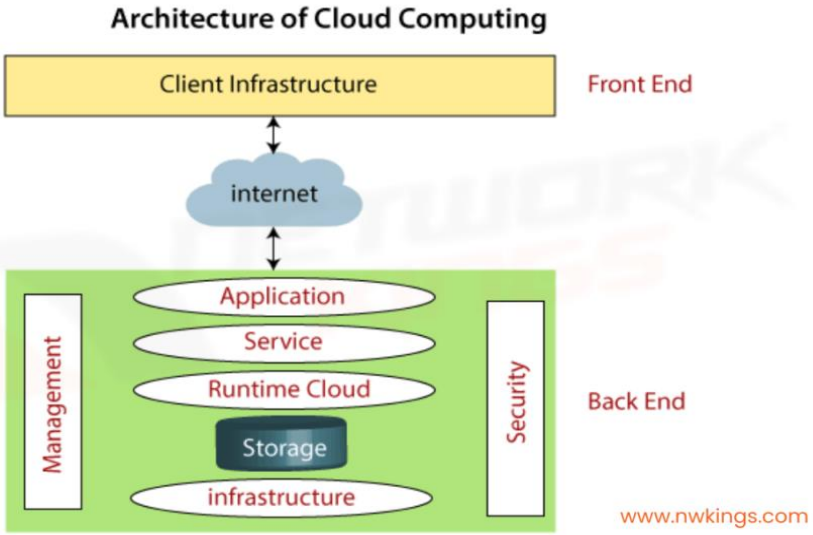
Cloud architecture is more of an as-a-service model than a network topology. For example, if you're using Google Docs on the cloud, you're not aware of its network topology. This refers to as **Software-as-a-Service** (SaaS).

- **PaaS:**

If you're working with **Platform-as-a-Service** (PaaS), you might be familiar with the cloud topology. You can access any development resource using PaaS such as Operating Systems to test out any application.

- **IaaS:**

When you're setting up a cloud-based network topology, you're using Infrastructure-as-a-Service (IaaS). Networks, servers, and firewalls are organized in the topology as virtualized components.



8.7 Check Your Progress

1. This set of Operating System Multiple Choice Questions & Answers (MCQs) focuses on "Distributed Operating System".

1. In distributed system, each processor has its own _____

a) local memory

b) clock

c) both local memory and clock

d) none of the mentioned

2. If one site fails in distributed system then _____

a) the remaining sites can continue operating

b) all the sites will stop working

c) directly connected sites will stop working

d) none of the mentioned

3. Network operating system runs on _____

a) server

b) every system in the network

c) both server and every system in the network

d) none of the mentioned

4. Which technique is based on compile-time program transformation for accessing remote data in a distributed-memory parallel system?

a) cache coherence scheme

b) computation migration

c) remote procedure call

d) message passing

5. Logical extension of computation migration is _____

a) process migration

b) system migration

c) thread migration

d) data migration

6. In distributed systems, link and site failure is detected by _____

a) polling

b) handshaking

c) token passing

d) none of the mentioned

7. Internet provides _____ for remote login.

a) telnet

b) http

c) ftp

d) rpc

8.8 Summary

- A distributed system is a collection of processors that do not share memory or a clock. Instead, each processor has its own local memory, and the processors communicate with one another through various communication lines, such as high-speed buses and the Internet. The processors in distributed system vary in size and function.
- A distributed system provides the user with access to all system resources. Access to a shared resource can be provided by data migration, computation migration, or process migration. The access can be specified by the user or implicitly supplied by the operating system and applications. Protocol stacks, as specified by network layering models, add information to a message to ensure that it reaches its destination.
- A naming system (such as DNS) must be used to translate from a host name to a network address, and another protocol (such as ARP) may be needed to translate the network number to a network device address (an Ethernet address, for instance).
- If systems are located on separate networks, routers are needed to pass packets from source network to destination network.
- The transport protocols UDP and TCP direct packets to waiting processes through the use of unique system-wide port numbers. In addition, the TCP protocol allows the flow of packets to become a reliable, connection-oriented byte stream.

- There are many challenges to overcome for a distributed system to work correctly. Issues include naming of nodes and processes in the system, fault tolerance, error recovery, and scalability. Scalability issues include handling increased load, being fault tolerant, and using efficient storage schemes, including the possibility of compression and/or deduplication.
- A DFS is a file-service system whose clients, servers, and storage devices are dispersed among the sites of a distributed system. Accordingly, service activity has to be carried out across the network; instead of a single centralized data repository, there are multiple independent storage devices.
- There are two main types of DFS models: the client–server model and the cluster-based model. The client-server model allows transparent file sharing among one or more clients. The cluster-based model distributes the files among one or more data servers and is built for large-scale parallel data processing.
- Ideally, a DFS should look to its clients like a conventional, centralized file system (although it may not conform exactly to traditional file-system interfaces such as POSIX). The multiplicity and dispersion of its servers and storage devices should be transparent. A transparent DFS facilitates client mobility by bringing the client's environment to the site where the client logs in.
- There are several approaches to naming schemes in a DFS. In the simplest approach, files are named by some combination of their host name and local name, which guarantees a unique system-wide name. Another approach, popularized by NFS, provides a means to attach remote directories to local directories, thus giving the appearance of a coherent directory tree.
- Requests to access a remote file are usually handled by two complementary methods. With remote service, requests for accesses are delivered to the server. The server machine performs the accesses, and the results are forwarded back to the client. With caching, if the data needed to satisfy the access request are not already cached, then a copy of the data is brought from the server to the client. Accesses are performed on the cached copy. The problem of keeping the cached copies consistent with the master file is the cache-consistency problem.

8.9 Keywords

1. **Confidentiality:** protection against disclosure to unauthorized individuals.

2. **Integrity**: protection against alteration/corruption

3. **Availability**: protection against interference with the means to access the resources.

4. **Detecting failures:** to manage the presence of failures cannot be detected but may be suspected.

5. **Masking failures:** hiding failure not guaranteed in the worst case.

6. **Concurrency:** Where applications/services process concurrency, it will affect a conflict in operations with one another and produce inconsistence results. Each resource must be designed to be safe in a concurrent environment.

7. **Cost:** Better price / performance as long as everyday hardware is used for the component computers – Better use of existing hardware

8.10 Self –Assessment Test

1. Why would you design a system as a distributed system? List some advantages of distributed systems.

2. List some disadvantages or problems of distributed systems that local only systems do not show (or at least not so strong)

3. List three properties of distributed systems

4. Give a definition of middleware and show in a small diagram where it is positioned.

Why would it be a bad idea for routers to pass broadcast packets between networks? What would be the advantages of doing so?

5**.** Discuss the advantages and disadvantages of caching name translations for computers located in remote domains.

6. What are two formidable problems that designers must solve to implement network system that has the quality of transparency?

7. Is it always crucial to know that the message you have sent has arrived at its destination safely? If your answer is "yes," explain why. If your answer is "no," give appropriate examples.

8. What implications does your answer have for recovery in distributed systems?

8.11 Answer to Check Your Progress

1. C

2. A

3. A

4. B

5. A

6. B

7. A

8.12 Suggested Material/ Reference Book

15. OperatingSystemConcepts,5th Edition,SilberschatzA.,GalvinP.B.,JohnWileyand Sons.
16. SystemsProgrammingandOperatingSystems,2nd RevisedEdition,DhamdhereD.M.,TataMcGrawHillPublishingCompanyLtd.,NewDelhi.
17. OperatingSystems,MadnickS.E.,DonovanJ.T.,TataMcGrawHillPublishingCompany Ltd.,New Delhi.
18. OperatingSystems-AModernPerspective,GaryNutt,PearsonEducationAsia,2000.
19. OperatingSystems,HarrisJ.A.,TataMcGrawHillPublishingCompanyLtd.,New Delhi, 2002.
20. Peterson and Davie (2012)] and [Kurose and Ross (2017)] provide generaloverviews of computer networks. The Internet and its protocols are describedin [Comer (2000)]. Coverage of TCP/IP can be found in [Fall and Stevens (2011)]and [Stevens (1995)]. UNIX network programming is described thoroughly in[Steven et al. (2003).
21. Ethernet and WiFi standards and speeds are evolving quickly. Current IEEE802.3 Ethernet standards can be found at http://standards.ieee.org/about/get/802/802.3.html. Current IEEE 802.11 Wireless LAN standards can be found athttp://standards.ieee.org/about/get/802/802.11.html.
22. Sun's network file system (NFS) is described by [Callaghan (2000)]. Information about OpenAFS is available from http://www.openafs.org.*

Course: MCA-32	Writer: Ritu
Lesson: 9	Vetter:

Memory Management-1

Structure

9.0 Objectives

Memory is a very important resource of the computer system to be managed by the operating system. The objectives of this lesson are to get the students familiar with the various concepts of memory management in particular:

(a) Segmentation

(b) Paging

(c) Virtual memory

9.1 Introduction

Memory management module is concerned with (a) Keeping track of whether each location is allocated or unallocated, to which process and how much, (b) If memory is to be shared by more than one process concurrently, it must be determined which process' request should be satisfied, (c) Once it is decided to allocate memory, the specific locations must be selected and allocated. Memory status information is updated, and (d) Handling the de-allocation/reclamation of memory. After the process holding memory is finished, memory locations held by it are declared free by changing the status information. In order to accomplish these, there are varieties of memory management systems. They are:

1. Contiguous, real memory management system such as:

- Single, contiguous memory management system
- Fixed partitioned memory management system

- Variable Partitioned memory management system

2. Non-Contiguous, real memory management system

- Paged memory management system
- Segmented memory management system
- Combined memory management system

3. Non-Contiguous, virtual memory management system

- Virtual memory management system

9.2 Contiguous Memory Management: In Contiguous Memory Management each program occupies a single contiguous block of storage locations.

9.2.1 Single Contiguous Memory Management

In this scheme, the physical memory is divided into two contiguous areas. One of them is permanently allocated to the resident portion of the OS. The remaining memory is allocated to user processes, which are loaded and executed one at a time, in response to user commands. This process is run to completion and then the next process is brought in memory.

9.2.2 Fixed Partitioned Memory Management System

In this scheme, memory is divided into number of contiguous regions called partitions, could be of different sizes. But once decided, they could not be changed. Partitions are fixed at the time of system generation, a process of setting the OS to specific requirements. There are two forms of memory partitioning (i) Fixed Partitioning and (ii) Variable Partitioning.

In fixed partitioning the main memory is divided into fixed number of partitions during system startup. The number and sizes of individual partitions are decided by the factors like capacity of the available physical memory, desired degree of multiprogramming, and the typical sizes of processes most frequently run on a given installation. The number of partitions represents an upper limit on degree of multiprogramming.

On request for partitions due to (1) creations of new processes or (2) reactivations of swapped-out processes, the memory manager attempts to satisfy these requests from the pool of free partitions.

Common obstacles faced by it are: (1) All partitions are allocated, or (2) No free partition is

large enough to accommodate the incoming process i.e. fragmentation. It refers to the unused memory that the memory management system cannot allocate. It is of two types: External and Internal. External Fragmentation is waste of memory between partitions caused by scattered non-contiguous free space. It occurs when total available memory space is enough to satisfy the request for a process to be allocated, but it is not continuous. It is severe in variable size partitioning schemes. Internal fragmentation is waste of memory within a partition caused by difference between size of partition and the process allocated. It refers to the amount of memory, which is not being used and is allocated along with a process request. It is severe in fixed partitioning schemes.

The main problem with fixed partitioned memory management system is determining the best region size to minimize the problem of fragmentation. It is difficult to achieve in fixed partitioning because in it the number of partitions and their sizes are decided statically and with a dynamic set of job to run there is no one right partition of memory.

9.2.3 Variable Partitioned Memory Allocation

In variable partitions, the number of partitions and their sizes are variable as they are not defined at the time of system generation. Starting with the initial state of the system, partitions are created dynamically to fit the needs of each requesting process. When a process departs, the memory manager returns the vacated space to the pool of free memory areas from which partition allocations are made. The OS obviously needs to keep track of both partitions and free memory. Once created, a partition is defined by its base address and size. Free areas of memory are produced upon termination of partitions and as leftovers in the partition creation process. For allocation and for partition creation purpose, the OS must keep track of the starting address and size of each free area of memory. The highly dynamic nature of both the number and the attributes of free areas suggest the use of some sort of a linked list to describe them within the free memory itself. Common algorithms for selection of a free area of memory are (a) First fit, (b) Best fit, (c) Worst fit, and (d) Next Fit.

First fit is faster because it terminates as soon as a free block large enough to house a new partition is found but it does not minimize wasted memory for a given allocation. Best fit searches the entire free list to find the smallest free block large enough to hold a partition being

created. Best fit is slower, and it tends to produce small leftover free blocks that may be too small for subsequent allocations.

Neither algorithm has been shown to be superior to the other in terms of wasted memory. Next fit is a modification of first fit whereby the pointer to the free list is saved following an allocation and used to begin the search for the subsequent allocation as opposed to always starting from the beginning of the free list. The idea is to reduce the search by avoiding examination of smaller blocks that tend to be created at the beginning of the free list as a result of previous allocations. Worst fit allocates the largest free block to reduce the rate of production of small holes. Simulation studies indicate that worst fit allocation is not very effective in reducing wasted memory in the processing of a series of requests.

9.3 Noncontiguous memory management

In Non-Contiguous Memory Management a program is divided into several blocks that may be placed throughout main storage in pieces not necessarily adjacent to one another. It is done in various ways broadly categorized as

(a) Non-Contiguous, real memory management system &

(b) Non-Contiguous, virtual memory management system.

9.3.1 Segmentation

Segments are formed at program translation time by grouping together logically related items. Programs are collections of subroutines, stacks, functions etc. Each of these components is of variable length and are logically related entities. All segments of all programs do not have to be of the same length. There is a maximum segment length. Although different segments may be placed in separate, noncontiguous areas of physical memory, items belonging to a single segment must be placed in contiguous areas of physical memory.

Segmentation is mapping of user's view onto physical memory. A logical address space is a collection of segments. Each segment has a name and length. User specifies each address by segment name or number and offset within segment. Segments are numbered and are referenced by segment number. For relocation purposes, each segment is compiled to begin at its own virtual address 0. An individual item within a segment is then identifiable by its offset relative

to the beginning of the enclosing segment. Thus, logical address consists of <segment no., offset>. To simplify processing, segment names are usually mapped to (virtual) segment numbers. This mapping is static, and systems programs in the course of preparation of process images may perform it.

9.3.1.1 Address Translation Since physical memory in segmented systems generally retains its linear-array organization, some address translation mechanism is needed to convert a two- dimensional virtual-segment address into its one-dimensional physical equivalent. In segmented systems, items belonging to a single segment reside in one contiguous area of physical memory. With each segment compiled as if starting from the virtual address zero, segments are generally individually reloadable. As a result, different segments of the same process need not occupy contiguous areas of physical memory.

When requested to load a segmented process, the OS attempts to allocate memory for the supplied segments. Using logic similar to that used for dynamic partitioning, it may create a separate partition to suit the needs of each particular segment. The base (obtained during partition creation) and size (specified in the load module) of a loaded segment are recorded as a tuple called the segment descriptor. All segment descriptors of a given process are collected in a table called the segment descriptor table (SDT). Two dimensional user defined address is mapped to one dimensional physical address by segment descriptor table. Each entry of this table has segment base and segment limit. Segment base contains the starting physical address of the segment and segment limit specifies the length of the segment.

Figure 1 illustrates a sample placement of the segments into physical memory, and the resulting SDT formed by the Operating System. With the physical base address of each segment defined, the process of translation of a virtual, two- component address into its physical equivalent basically follows the mechanics of based addressing. The segment number provided in the virtual address is used to index the segment descriptor table and to obtain the physical base address of the related segment. Adding the offset of the desired item to the base of its enclosing segment then produces the physical address.

The size of a SDT is related to the size of the virtual address space of a process.

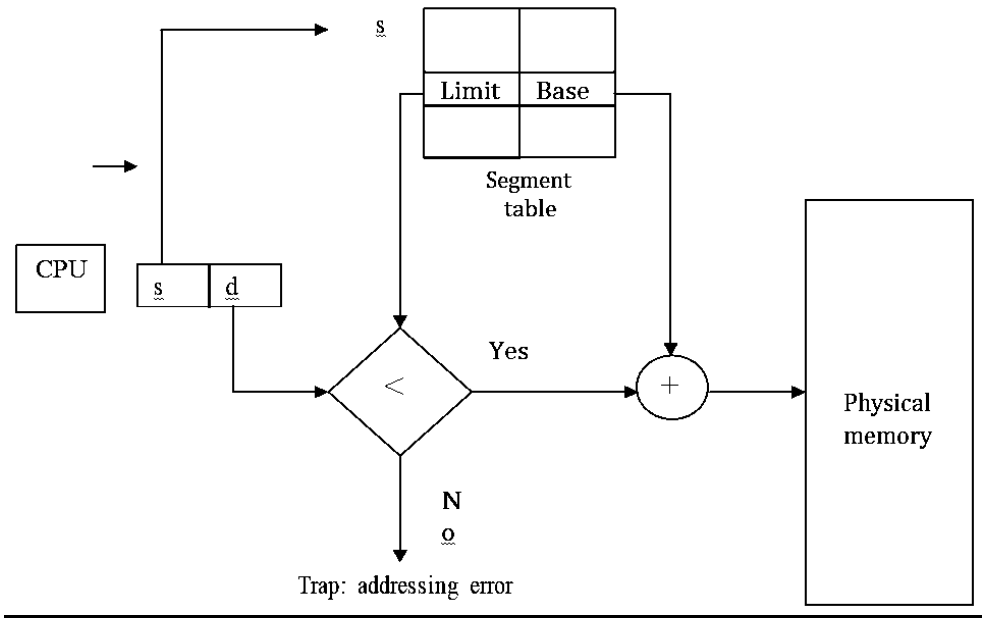
Given their potential size, SDT are not kept in registers. Being a collection of logically related

items, the SDTs themselves are often treated as special types of segments. Their accessing is usually facilitated by means of a dedicated hardware register called the segment descriptor table base register (SDTBR), which is set to point to the base of the running process's SDT.

Since the size of an SDT may vary from a few entries to several thousand, another dedicated hardware register, called the segment descriptor table limit register (SDTLR), is provided to mark the end of the SDT pointed to by the SDTBR. In this way, an SDT need contain only as many entries as there are segments actually defined in a given process. Attempts to access nonexistent segments may be detected and dealt with as nonexistent-segment exceptions. Mapping each virtual address requires two physical memory references for a single virtual (program) reference, as follows:

- Memory reference to access the segment descriptor in the SDT
- Memory reference to access the target item in physical memory

In other words, segmentation may cut the effective memory bandwidth in half by making the effective virtual-access time twice as long as the physical memory access time.



Hardware support for Segmentation

Figure 1 – Address translation in segmented systems

9.3.1.2 Segment Descriptor Caching

As performance of segmented systems is dependent on the address translation process, system designers often provide some hardware accelerators to speed the translation. Memory references expended on mapping may be avoided by keeping segment descriptors in registers. However to keep an entire SDT of the running process in register is very costly, hence most frequently used segment descriptors are kept in registers. In this way, most of the memory references may be mapped with the aid of registers. The rest may be mapped using the SDT in memory, as usual. This scheme is dependent on the OS's ability to select the proper segment descriptors for storing into registers. In order to provide the intuitive motivation for one possible implementation of systematic descriptor selection, let us investigate the types of segments referenced by the executing process. Memory references may be functionally categorized as accesses to (i) Instructions, (ii) Data, and (iii) Stack. A typical instruction execution sequence consists of a mixture of the outline types of memory references.

In fact, completion of a single stack manipulation instruction, such as a push of a datum from memory onto stack, may require all three types of references. Thus the working space of a process normally encompasses one each of code, data, and stack segments. Therefore, keeping the current code, data, and stack segment descriptors in registers may accelerate address translation. Depending on its type, a particular memory reference may then be mapped using the appropriate register. But can we know the exact type of each memory reference as the processor is making it? The answer is yes, with the proper hardware support. Namely, in most segmented machines the CPU emits a few status bits to indicate the type of each memory reference. The memory management hardware uses this information to select the appropriate mapping register. Register-assisted translation of virtual to physical addresses is illustrated in Figure 2.

As indicated, the CPU status lines are used to select the appropriate segment descriptor register (SDR). The size field of the selected segment descriptor is used to check whether the intended reference is within the bounds of the target segment. If so, the base field is added with the offset to produce the physical address. By making the choice of the appropriate segment register implicit in the type of memory reference being made, segment typing may eliminate the need to keep track of segment numbers during address translations.

Though segment typing is certainly useful, it may become restrictive at times. For example, copying an instruction sequence from one segment into another may confuse the selector logic into believing that source and target segments should be of type data rather than code.

Using the so-called segment override of type prefixes, which allows the programmer to explicitly indicate the particular segment descriptor register to be used for mapping the memory reference in question, may alleviate this problem.

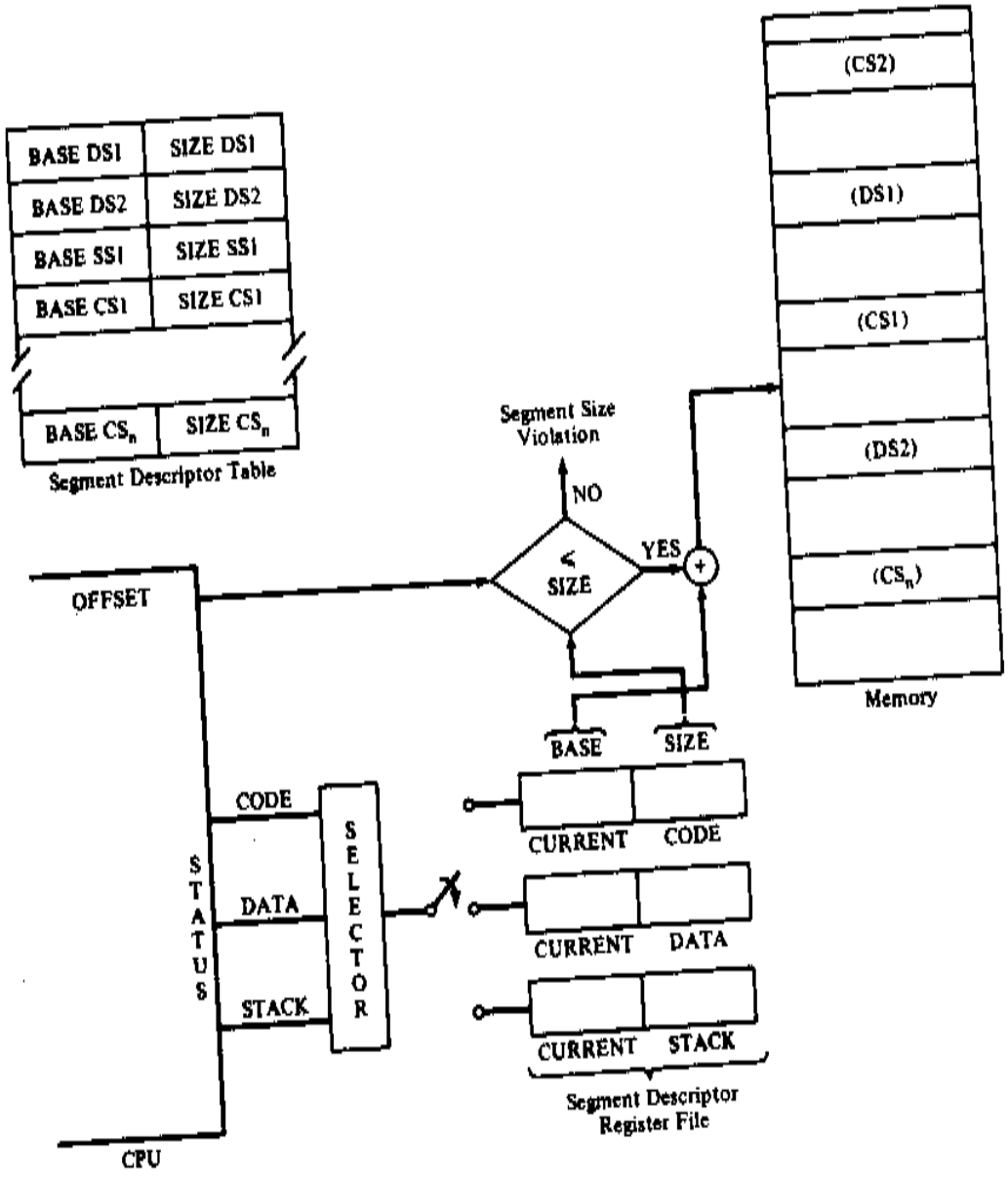


Figure 2–Segment-descriptor cache registers

Segment descriptor registers are initially loaded from the SDT. Whenever the running process makes an intersegment reference, the corresponding segment descriptor is loaded into the appropriate register from the SDT. For example, an intersegment JUMP or CALL causes the segment descriptor of the target (code) segment to be copied from the SDT to the code segment descriptor register. When segment typing is used as described, segment descriptor caching becomes deterministic as opposed to probabilistic. Segment descriptors stored in the three segment descriptor registers; define the current working set of the executing process. Since membership in the working set of segments of a process changes with time, segment descriptor registers are normally included in the process state. Upon each process switch, the contents of the SDRs of the departing process are stored with the rest of its context. Before dispatching the new running process, the OS loads segment descriptor registers with their images recorded in the related PCB.

9.3.1.3 Protection

The base-limit form of protection is obviously the most natural choice for segmented systems. The legal address space of a process is the collection of segments defined by its SDT. Except for shared segments. Placing different segments in disjoint areas of memory enforces separation of distinct address space. An interesting possibility in segmented systems is to provide protection within the address space of a single process, in addition to the more usually protection between different processes. Given that the type of each segment is defined commensurate with the nature of information stored in its constituent elements, access rights to each segment can be defined accordingly. For instance, though both reading and writing of stack segments may be necessary, accessing of code segments can be permitted in execute-only or perhaps in the read-only mode. Data segments can be read-only, write-only, or read-write. Thus, segmented systems may be able to prohibit some meaningless operations. Additional examples include prevention of stack growth into the adjacent code or data areas, and other errors resulting from mismatching of segment types and intended references to them. An important observation is that access rights to different portions of a single address space may vary in accordance with the type of information stored therein. Due to the grouping of logically related items, segmentation is one of the rare memory-management schemes that allow such finely grained delineation of access rights.

The mechanism for enforcement of declared access rights in segmented systems is usually coupled with the address translation hardware. Typically, access-rights bits are included in segment descriptors. In the course of address mapping, the intended type of reference is checked against the access rights for the segment in question. Any mismatch results in abortion of the memory reference in progress, and a trap to the OS.

9.3.1.4 Sharing Shared objects are usually placed in separate, dedicated segments. A shared segment may be mapped, via the appropriate SDTs, to the virtual-address spaces of all processes that are authorized to reference it. The deliberate use of offsets and of bases addressing facilitate sharing since the virtual offset of a given item is identical in all processes that share it. The virtual number of a shared segment, on the other hand, need not be identical in all address spaces of which it is a member. These points are illustrated in Figure 3, where a code segment EMACS is assumed to be shared by three processes P1, P2 & P3. The relevant portions of the SDTs of the participating processes P1, P2 & P3 are SDT1, SDT2, and SDT3 respectively, and shown. As indicated, the segment EMACS is assumed to have different virtual numbers in the three address spaces of which it is part. The placement of access-rights bits in segment descriptor tables is also shown. Figure 3 illustrates the fact that different processes can have different access rights to the same shared segment. For example, whereas processes P1 and P2 can execute only the shared segment EMACS, process P3 is allowed both reading and writing Figure 3 also illustrates the ability of segmented systems to conserve memory by sharing the code of programs executed by many users. In particular, each participating process can execute the shared code from EMACS using its own private data segment. Assuming there is an editor, this means that a single copy of it may serve the entire user population of a time-sharing system. Naturally, execution of EMACS on behalf of each user is stored in a private data segment of its corresponding process. For example, users 1, 2, and 3 can have their respective texts buffers stored in data segments DATA1, DATA2, and DATA3. Depending on which of the three processes is active at a given time, the hardware data segment descriptor register points to data segment DATA1, DATA2, or DATA3, and the code segment descriptor register points to EMACS in all cases. Of course, the current instruction to be executed by the particular process is indicated by the program counter, which is saved and restored as a part of each process's state. In segmented systems, the program counter register

usually contains offsets of instructions within the current code segment. This facilitates sharing by making all code self-references relative to the beginning of the current code segment.

When coupled with segment typing, this feature makes it possible to assign different virtual segment numbers to the same (physical) shared segment in virtual-address spaces of different processes of which it is a part. Alternatively, the problem of making direct self- reference in shared routines restricts the type of code that may safely be shared. As described, sharing is encouraged in segmented systems.

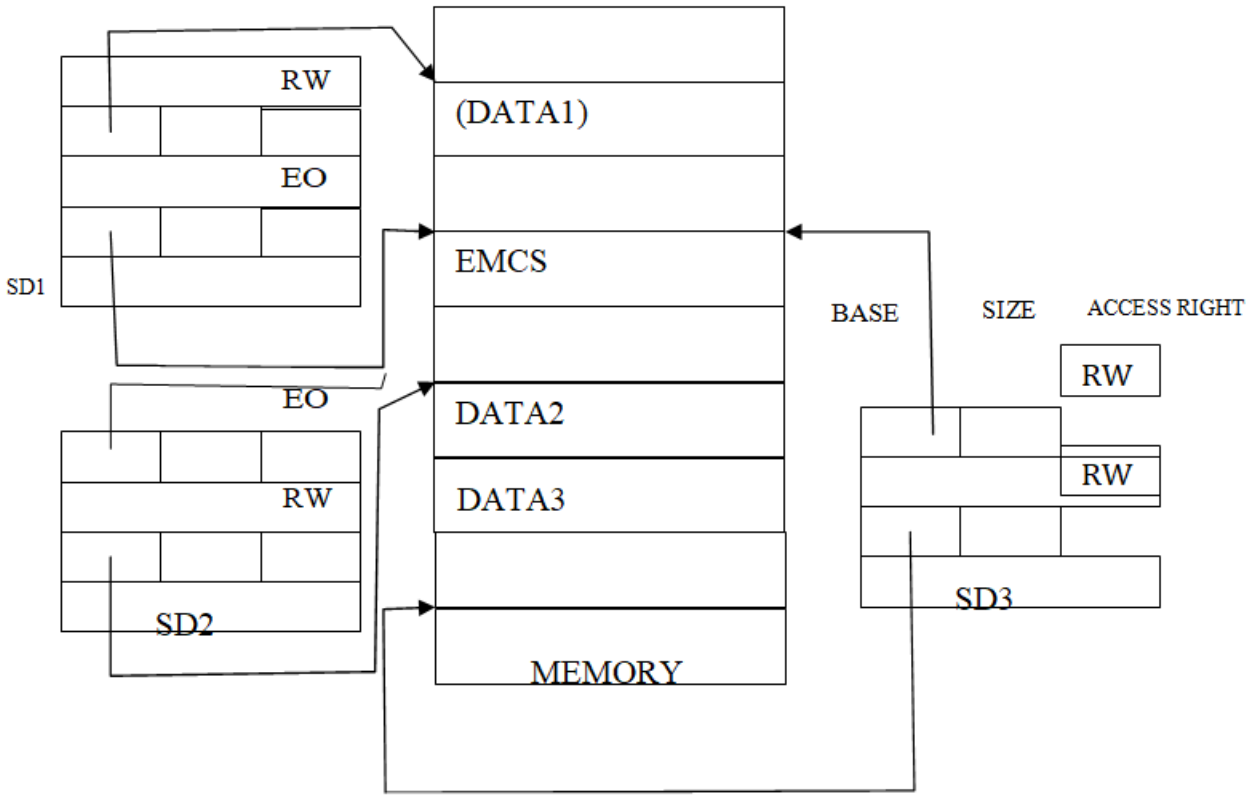


Figure 3 – Sharing in segmented systems

This presents some problems in systems that also support swapping, which is normally done to increase processor utilization. For example, a shared segment may need to maintain its memory residence while being actively used by any of the processes authorized to reference it. Swapping in this case opens up the possibility that a participating process may be swapped out while its shared segment remains resident. When such a process is swapped back in, the construction of its SDT must take into consideration the fact that the shared segment may already be resident.

In other words, the OS must keep track of shared segments and of processes that access them. When a participating process is loaded in memory, the OS is expected to identify the location of the shared segment in memory, if any, and to ensure its proper mapping from all virtual address spaces of which it is a part.

4.3.2 Paging

In it, the physical memory is conceptually divided into a number of fixed-size slots, called page frames. The virtual-address space of a process is also split into fixed-size blocks of the same size, called pages. Memory management module identifies sufficient number of unused page frames for loading of the requesting process's pages. An address translation mechanism is used to map virtual pages to their physical counterparts. Since each page is mapped separately, different page frames allocated to a single process need not occupy contiguous areas of physical memory.

Figure 4 demonstrates the basic principle of paging. It illustrates a sample 16 MB system where virtual and physical addresses are assumed to be 24 bits long each. The page size is assumed to be 4096 bytes. Thus, the physical memory can accommodate 4096 page frames of 4096 bytes each. After reserving 1 MB of physical memory for the resident portion of the Operating System, the remaining 3840 page frames are available for allocation to user processes. The addresses are given in hexadecimal notation. Each page is 1000H bytes long, and the first user-allocatable page frame starts at the physical address 100000H. The virtual-address space of a sample user process that is 14,848 bytes (3A00H) long is divided into four virtual pages numbered from 0 to 3. A possible placement of those pages into physical memory is depicted in Figure 4. The mapping of virtual addresses to physical addresses in paging systems is performed at the page level. Each virtual address is divided into two parts: the page number and the offset within that page. Since pages and page frames have identical sizes, offsets within each are identical and need not be mapped. So each 24-bit virtual address consists of a 12-bit page number (high-order bits) and a 12-bit offset within the page.

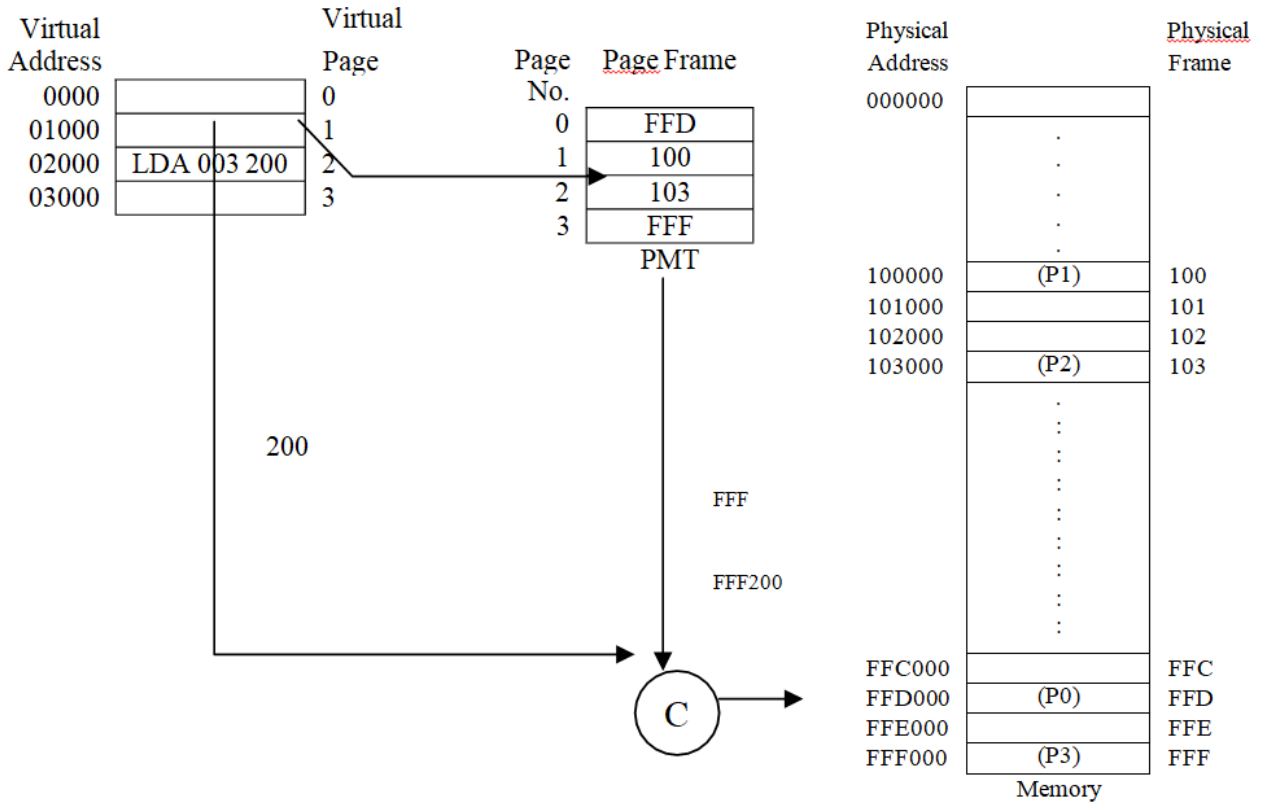


Figure 4 – Paging

Address translation is performed with the help of the page-map table (PMT), constructed at process-loading time. As indicated in figure 4, there is one PMT entry for each virtual page of a process. The value of each entry is the number of the page frame in the physical memory where the corresponding virtual page is placed. Since offsets are not mapped, only the page frame number need be stored in a PMT entry. E.g., virtual page 0 is assumed to be placed in the physical page frame whose starting address is FFD000H (16,764,928 decimal). With each frame being 1000H bytes long, the corresponding page frame number is FFDH, as indicated on the right-hand side of the physical memory layout in Figure 4. This value is stored in the first entry of the PMT. All other PMT entries are filled with page frame numbers of the region where the corresponding pages are actually loaded. The logic of the address translation process in paged systems is illustrated in Figure 4 on the example of the virtual address 03200H. The virtual address is split by hardware into the page number 003H, and the offset within that page(200H).

The page number is used to index the PMT and to obtain the corresponding physical frame number,

i.e. FFF. This value is then concatenated with the offset to produce the physical address, FFF200H, which is used to reference the target item in memory.

The OS keeps track of the status of each page frame by using a memory-map table (MMT). Format of an MMT is illustrated in Figure 5, assuming that only the process depicted in Figure 5 and the OS are resident in memory.

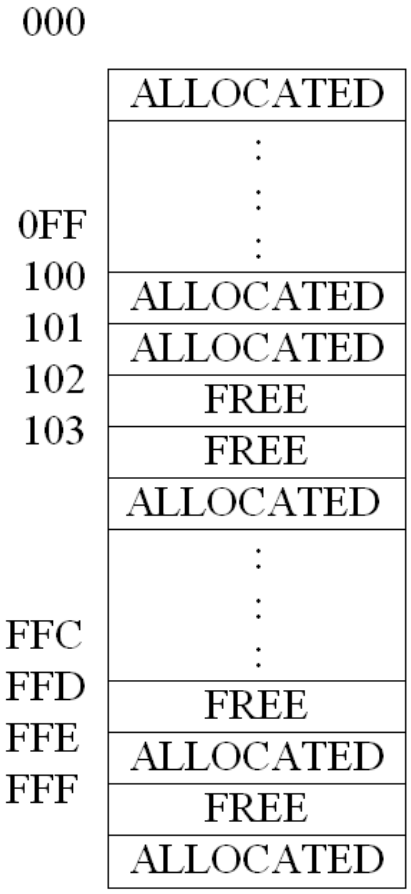


Figure 5 – Memory-map table (MMT)

Each entry of the MMT described the status of page frame as FREE or ALLOCATED. The number of MMT entries i.e. ***f*** is computed as ***f*** = ***m/p*** where ***m*** is the size of the physical memory, and ***p*** is page size. Both ***m*** and ***p*** are usually an integer power of base 2, thus resulting in f being an integer. When requested to load a process of size ***s***, the OS must allocate n free page frames, so that ***n*** = Round(***s/p***) where ***p*** is the page size. The OS allocates memory in terms

of an integral number of page frames. If the size of a given process is not a multiple of the page size, the last page frame may be partly unused resulting into page fragmentation.

After selecting n free page frames, the OS loads process pages into them and constructs the page-map table of the process. Thus, there is one MMT per system, and as many PMTs as there are active processes.

When a process terminates or becomes swapped out, memory is deallocated by releasing the frame holdings of the departing process to the pool of free page frames.

9.3.2.1 Page Allocation

The efficiency of the memory allocation algorithm depends on the speed with which it can locate free page frames. To facilitate this, a list of free pages is maintained instead of the static-table format of the memory map assumed earlier. In that case, n free frames may be identified and allocated by unlinking the first n nodes of the free list. De-allocation of memory in systems without the free list consists of marking in the MMT as FREE all frames found in the PMT of the departing process a time consuming operation. Frames identified in the PMT of the departing process can be linked to the beginning of the freed list. Linking at the beginning is the fastest way of adding entries to an unordered singly linked list. Since the time complexity of deallocation is not significantly affected by the choice of data structure of free pages, the free-list approach has a performance advantage as its time complexity of deallocation is not significantly affected by the choice of data structure of free pages, and is not affected by the variation of memory utilization.

9.3.2.2 Hardware Support for Paging

Hardware support for paging, concentrates on saving the memory necessary for storing of the mapping tables, and on speeding up the mapping of virtual to physical addresses. In principle, each PMT must be large enough to accommodate the maximum size allowed for the address space of a process in a given system. In theory, this may be the entire physical memory. So in a 16 MB system with 256-byte pages, the size of a PMT should be 64k entries. Individual PMT entries are page numbers that are 16 bits long in the sample system, thus requiring 128 KB of physical memory to store a PMT. With one PMT needed for each active process, the total PMT storage can consume a significant portion of physical memory. Since the actual address space of a process may be well

below its allowable maximum, it is reasonable to construct each PMT with only as many entries as its related process has pages. This may be accomplished by means of a dedicated hardware page-map table limit register (PMTLR). A PMTLR is set to the highest virtual page number defined in the PMT of the running process. Accessing of the PMT of the running process may be facilitated by means of the page-map table base register (PMTBR), which points to the base address of the PMT of the running process.

The respective values of these two registers for each process are defined at process-loading time and stored in the related PCB. Upon each process switch, the PCB of the new running process provides the values to be loaded in the PMTBR and PMTLR registers. Even with the assistance of these registers, address translations in paging systems still require two memory references; one to access the PMT for mapping, and the other to reference the target item in physical memory. To speed it up, a high-speed associative memory for storing a subset of often-used page-map table entries is used. This memory is called the translation look aside buffer (TLB), or mapping cache.

Associative memories can be searched by contents rather than by address. So, the main-memory reference for mapping can be substituted by a TLB reference. Given that the TLB cycle time is very small, the memory-access overhead incurred by mapping can be significantly reduced. The role of the cache in the mapping process is depicted in Figure 6. As indicated, the TLB entries contain pairs of virtual page numbers and the corresponding page frame numbers where the related pages are stored in physical memory. The page number is necessary to define each particular entry, because a TLB contains only a subset of page-map table entries. Address translation begins by presenting the page-number portion of the virtual address to the TLB. If the desired entry is found in the TLB, the corresponding page frame number is combined with the offset to produce the physical address.Alternatively, if the target entry is not in TLB, the PMT in memory must be accessed to complete the mapping. This process begins by consulting the PMTLR to verify that the page number provided in the virtual address is within the bounds of the related process's address space. If so, the page number is added to the contents of the PMTBR to obtain the address of the corresponding PMT entry where the physical page frame number is stored.

This value is then concatenated with the offset portion of the virtual address to produce the

physical memory address of the desired item.

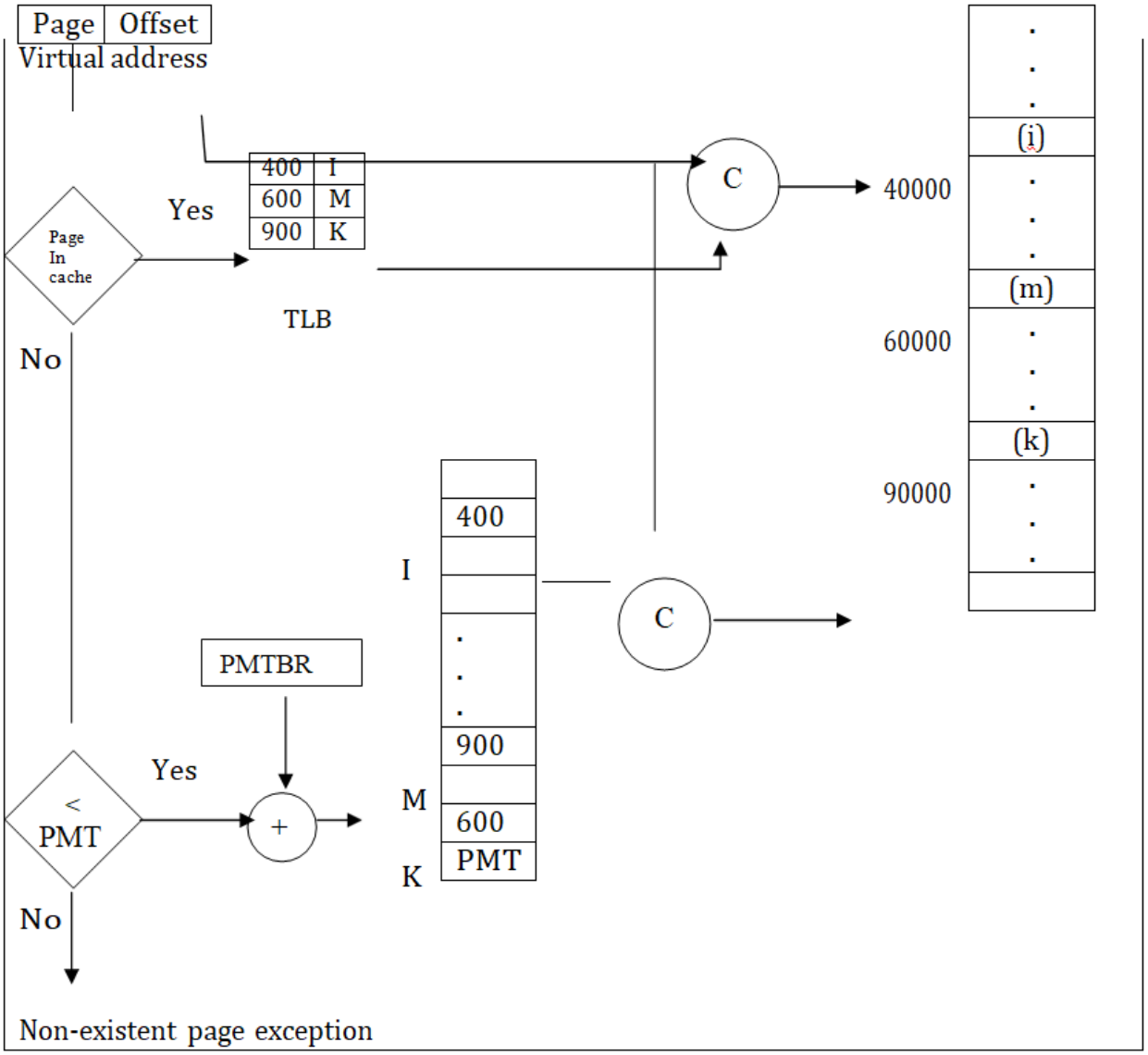


Figure 6 – Translation-look aside buffer (TLB)

Figure 6 demonstrates that the overhead of TLB search is added to all mappings, regardless of whether they are eventually completed using the TLB or the PMT in main memory. In order for the TLB to be effective, it must satisfy a large portion of all address mappings. Given the generally small size of a TLB because of the high price of associative memories, only the PMT entries most likely to be needed ought to reside in the TLB.

The effective memory-access time, t_{eff} in systems with run-time address translation is the sum of

the address translation time, t_{TR} and the subsequent access time needed to fetch the target item from memory, t_M. So $t_{eff} = t_{TR} + t_{M.}$ With TLB used to assist in address translation, t_{TR} becomes

$T_{TR} = h\ t_{TLB} + (1 - h)\ (t_{TLB} + t_M) = t_{TLB} + (1 - h)t_{M;}$ where h is the TLB hit ratio, that is, the ratio of address translations that are contained the TLB over all translations, and thus 0□h□1;t_{TLB} is the TLB access time; and t_M is the main-memory access time. Therefore, effective memory-access time in systems with a TLB is $t_{eff} = t_{TLB} + (2 - h)t_{M;}$ It is observed that the hardware used to accelerate address translations in paging systems (TLB) is managed by means of probabilistic algorithms, as opposed to the deterministic mapping-register typing described in relation to segmentation. The reason is that the mechanical splitting of a process's address space into fixed-size chunks produces pages. As a result, a page, unlike a segment, in general does not bear any relationship to the logical entities of the underlying program. For example, a single page may contain a mixture of data, stack, and code. This makes typing and other forms of deterministic loading of TLB entries extremely difficult, in view of the stringent timing restrictions imposed on TLB manipulation.

9.3.2.3 Protection and Sharing

Unless specifically declared as shared, distinct address spaces are placed in disjoint areas of physical memory. Memory references of the running process are restricted to its own address space by means of the address translation mechanism, which uses the dedicated PMT. The PMTLR is used to detect and to abort attempts to access any memory beyond the legal boundaries of a process. Modifications of the PMTBR and PMTLR registers are usually possible only by means of privileged instructions, which trap to the OS if attempted in user mode. By adding the access bits to the PMT entries and appropriate hardware for testing these bits, access to a given page may be allowed only in certain programmer-defined modes such as read-only, execute-only, or other restricted forms of access. This feature is much less flexible in paging systems than segmentation. The primary difference is that paging is supposed to be entirely transparent to programmers. Mechanical splitting of an address space into pages is performed without any regard for the possible logical relationships between the items under consideration. Since there is no notion of typing, code and data may be mixed within one page. As we shall see, specification of the access rights in paging systems is useful for pages shared by several processes, but it is of much less value inside the boundaries of a given address space.

Protection in paging systems may also be accomplished by means of the protection keys. In principle, the page size should correspond to the size of the memory block protected by the single key. This allows pages belonging to a single process to be scattered throughout memory- a perfect match for paged allocation. By associating access-rights bits with protection keys, access to a given page may be restricted when necessary.

Sharing of pages is quite straightforward with paged memory management. A single physical copy of a shared page can be easily mapped into as many distinct address spaces as desired. Since each such mapping is performed via a dedicated entry in the PMT of the related process, different processes may have different access rights to the shared page. Given that paging is transparent to users, sharing at the page level must be recognized and supported by systems programs. Systems programs must ensure that virtual offsets of each item within a shared page are identical in all participating address spaces.

Like data, shared code must have the same within-page offsets in all address spaces of which it is a part. As usual, shared code that is not executed in mutually exclusive fashion must be reentrant. In addition, unless the shared code is position-independent, it must have the same virtual page numbers in all processes that invoke it. This property must be preserved even in cases when the shared code spans several pages.

9.3.3 Virtual Memory Under Virtual Memory all processes execute code written in terms of virtual addresses that are translated by the memory management hardware into the appropriate physical address. Each process thinks it has access to the whole physical memory of the machine. This solves the relocation problem – no rewriting of addresses is ever necessary, and the protection problem because a process can no longer express the idea of accessing another process's memory. Virtual memory allows execution of partially loaded processes. As a consequence, virtual address spaces of active processes in a virtual-memory system can exceed the capacity of the physical memory. This is accomplished by maintaining an image of the entire virtual-address space of a process on secondary storage, and by bringing its sections into main memory when needed. The OS decides which sections to bring in, when to bring them in, and where to place them. Thus, virtual-memory systems provide for automatic migration of portions of address spaces between secondary and primary storage.

Due to the ability to execute a partially loaded process, a process may be loaded into a space of arbitrary size resulting into the reduction of external fragmentation. Moreover, the amount of space in use by a given process may be varied during its memory residence. As a result, the OS may speed up the execution of important processes by allocating them more real memory. Alternatively, by reducing the real-memory holdings of resident processes, the degree of multi-programming can be increased by using the vacated space to activate more processes.

The speed of program execution in virtual-memory systems is bounded from above by the execution speed of the same program run in a non-virtual memory management system. That is due to delays caused by fetching of missing portions of program's address space at run-time.

Virtual memory provides execution of partially loaded programs. But an instruction can be completed only if all code, data, and stack locations that it references reside in physical memory. When there is a reference for an out-of- memory item, the running process must be suspended to fetch the target item from disk. So what is the performance penalty?

An analysis of program behavior provides an answer to the question. Most programs consist of alternate execution paths, some of which do not span the entire address space. On any given run, external and internal program conditions cause only one specific execution path to be followed. Dynamic linking and loading exploits this aspect of program behavior by loading into memory only those procedures that are actually referenced on a particular run. Moreover, many programs tend to favor specific portions of their address spaces during execution. So it is reasonable to keep in memory only those routines that make up the code of the current pass. When another pass over the source code commences, the memory manager can bring new routines into the main memory and return those of the previous pass back to disk.

9.3.3.1 Principles of Operation

Virtual memory can be implemented as an extension of paged or segmented memory management or as a combination of both. Accordingly, address translation is performed by means of PMT, SDT, or both. The process of address mapping in virtual-memory systems is more formally defined as follows. Let the virtual-address space be V = {0, 1, … , v-1}, and the physical memory space by M = {0, 1, ….m-1}. The OS dynamically allocates real memory to portions of the virtual-address space. The

address translation mechanism must be able to associate virtual names with physical locations. At any time the mapping hardware must realize the function f: V □ M such that

r if item x is in real memory at location r

f(x) =

missing-item exception if item x is not in real memory

Thus, the additional task of address translation hardware in virtual systems is to detect whether the target item is in real memory or not. If the referenced item is in memory, the process of address translation is completed.

We present the operation of virtual memory assuming that paging is the basic underlying memory-management scheme. The detection of missing items is rather straightforward. It is usually handled by adding the presence indicator, a bit, to each entry of PMTs. The presence bit, when set, indicates that the corresponding page is in memory; otherwise the corresponding virtual page is not in real memory. Before loading the process, the OS clears all the presence bits in the related PMT. As and when specific pages are brought into the main memory, its presence bit is reset.

A possible implementation is illustrated in Figure 7. The presented process's virtual space is assumed to consisting of only six pages. As indicated, the complete process image is present in secondary memory. The PMT contains an entry for each virtual page of the related process. For each page actually present in real memory, the presence bit is set (IN), and the PMT points to the physical frame that contains the corresponding page. Alternatively, the presence bit is cleared (OUT), and the PMT entry is invalid

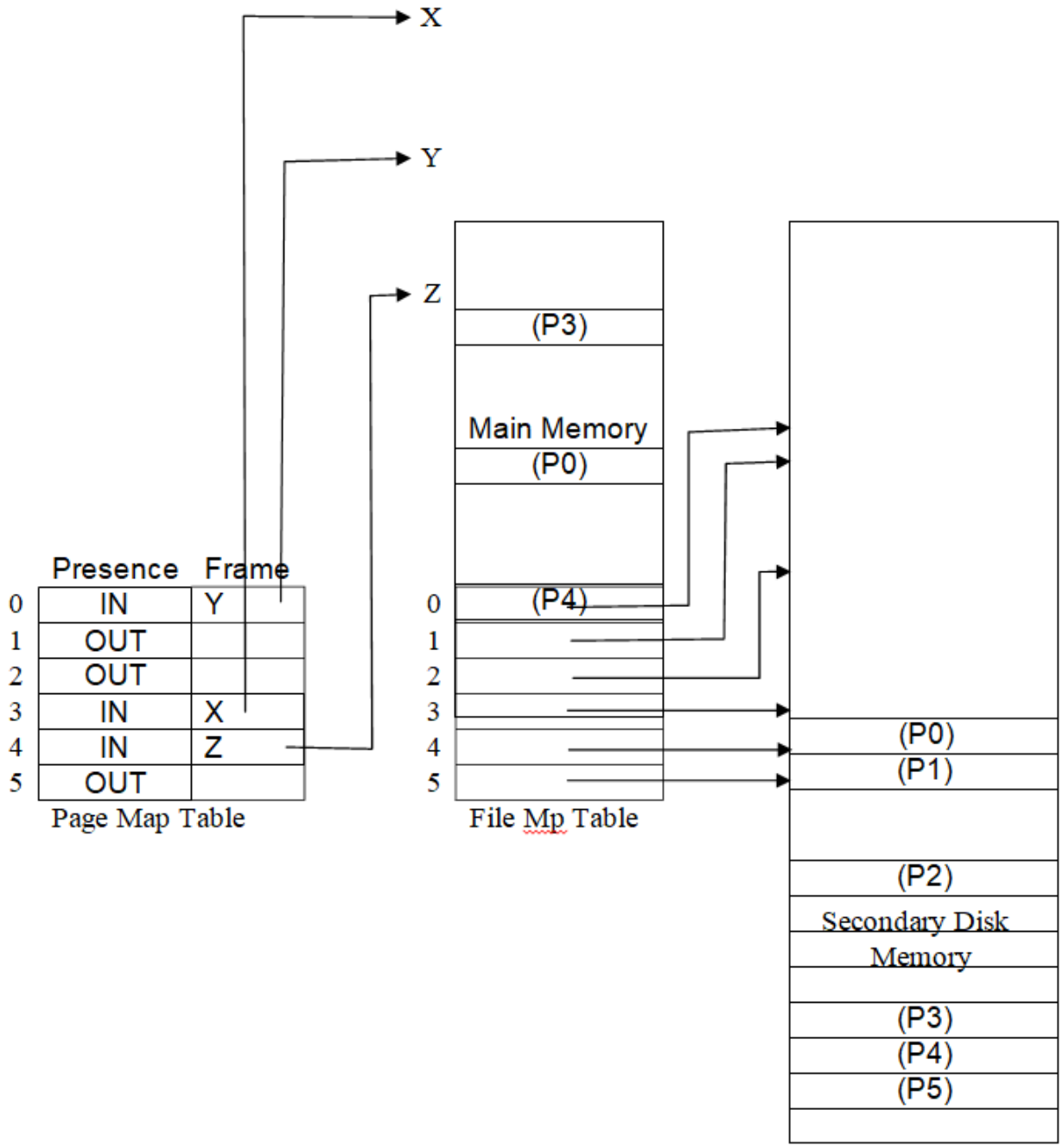


Figure 7: Virtual Memory

The address translation hardware checks the presence bit during the mapping of each memory reference if the bit is set, the mapping is completed as usual.

However, if the corresponding presence bit in the PMT is reset, the hardware generates a missing-item exception known as page fault. When the running process experiences a page fault, it must be

suspended until the missing page is brought into main memory.

The disk address of the faulted page is usually provided in the file-map table (FMT). This table is parallel to the PMT. Thus, when processing a page fault, the OS uses the virtual page number provided by the mapping hardware to index the FMT and to obtain the related disk address. A possible format and use of the FMT is depicted in Figure 7.

9.3.3.2 Management of Virtual Memory

The implementation of virtual memory requires maintenance of one PMT per active process. Given that the virtual-address space of a process may exceed the capacity of real memory, the size of an individual PMT can be much larger in a virtual than in a real paging system with identical page sizes. The OS maintains one MMT or a free-frame list to keep track of Free/allocated page frames.

A new component of the memory manager's data structures is the FMT. FMT contains secondary-storage addresses of all pages. The memory manager used the FMT to load the missing items into the main memory. One FMT is maintained for each active process. Its base may be kept in the control block of the related process. An FMT has a number of entries identical to that of the related PMT. A pair of page-map table base and page-map length registers may be provided in hardware to expedite the address translation process and to reduce the size of PMT for smaller processes. As with paging, the existence of a TLB is highly desirable to reduce the negative effects of mapping on the effective memory bandwidth.

The allocation of only a subset of real page frames to the virtual-address space of a process requires the incorporation of certain policies into the virtual-memory manager. We may classify these policies as follows:

1. Allocation policy: How much real memory to allocate to each active process
2. Fetch policy: Which items to bring and when to bring them from secondary storage into the main memory
3. Replacement policy: When a new item is to be brought in and there is no free real memory, which item to evict in order to make room.
4. Placement policy: Where to place an incoming item

9.3.3.3 Program Behavior

By minimizing the number of page faults, the effective processor utilization, effective disk I/O bandwidth, and program turnaround times may be improved. It is observed that there is a strong tendency of programs to favor subsets of their address spaces during execution. This phenomenon is known as locality of reference. Both temporal and spatial locality of reference has been observed.

(a) Spatial locality suggests that once an item is referenced, there is a high probability that it or its neighboring items are going to be referenced in the near future.

(b) Temporal locality is the tendency for a program to reference the same location or a cluster several times during brief intervals of time. Temporal locality of reference is exhibited by program loops.

Both temporal and spatial locality of reference is dynamic properties in the sense that the identity of the particular pages that compose the actively used set varies with time. As observed, the executing program moves from one locality to another in the course of its execution. Statistically speaking, the probability that a particular memory reference is going to be made to a specific page is a time- varying function. It increases when pages in its current locality are being referenced, and it decreases otherwise. The evidence also suggests that the executing program moves slowly from one locality to another. Locality of reference basically suggests that a significant portion of memory references of the running process may be made to a subset of its pages. These findings may be utilized for implementation of replacement and allocation policies.

9.3.3.4 Replacement Policies

If a page fault is there, then it is to be brought into the main memory necessitating creation of a room for it. There are two options for this situation:

- The faulted process may be suspended until availability of memory.
- A page may be removed to make room for the incoming one.

Suspending a process is not an acceptable solution. Thus, removal is commonly used to free the memory needed to load the missing items. A replacement policy decides the victim page for eviction. In virtual memory systems all pages are kept on the secondary storage. As and when

needed, some of those pages are copied into the main memory. While executing, the running process may modify its data or stack areas, thus making some resident pages different from their disk images (dirty page). So it must be written back to disk in place of its obsolete copy. When a page that has not been modified (clean page) during its residence in memory is to be evicted, if can simply be discarded. Tracking of page modifications is usually performed in hardware by adding a written-into bit called as dirty bit, to each entry of the PMT. It indicates whether the page is dirty or clean.

9.3.3.5 Replacement Algorithms

First-In-First-Out (FIFO):

The FIFO algorithm replaces oldest pages i.e. the resident page that has spent the longest time in memory. To implement the FIFO page-replacement algorithm, the memory manager must keep track of the relative order of the loading of pages into the main memory. One way to accomplish this is to maintain a FIFO queue of pages.

FIFO fails to take into account the pattern of usage of a given page; FIFO tends to throw away frequently used pages because they naturally tend to stay longer in memory. Another problem with FIFO is that it may defy intuition by increasing the number of page faults when more real pages are allocated to the program. This behavior is known as Belady's anomaly.

Least Recently Used (LRU):

The LRU algorithm replaces the least recently used resident page. LRU algorithm performs better than FIFO because it takes into account the patterns of program behavior by assuming that the page used in the most distant past is least likely to be referenced in the near future. The LRU algorithm belongs to a larger class of stack replacement algorithms. A stack algorithm is distinguished by the property of performing better, or at least not worse, when more real memory is made available to the executing program. Stack algorithms therefore do not suffer from Belady's anomaly. The implementation of the LRU algorithm imposes too much overhead to be handled by software alone. One possible implementation is to record the usage of pages by means of a structure similar to the stack. Whenever a resident page is referenced, it is removed from its current stack position and placed at the top of the stack. When a page eviction is in order, the page at the bottom of the stack is removed from memory.

Maintenance of the page-referencing stack requires it's updating for each page reference, regardless of whether it results in a page fault or not. So the overhead of searching the stack, moving the reference page to the top, and updating the rest of the stack accordingly must be added to all memory references. But the FIFO queue needs to be updated only when page faults occur-overhead almost negligible in comparison to the time required for processing of a page fault.

Optimal (OPT):

The algorithm by Belady, removes the page to be reference in the most distant future i.e. page out the page that will be needed the furthest in the future. This is impossible (halting problem), but provides an interesting benchmark. Since it requires future knowledge, the OPT algorithm is not realizable. Its significance is theoretical, as it can serve as a yardstick for comparison with other algorithms.

Approximations-Clock:

One popular algorithm combines the relatively low overhead of FIFO with tracking of the resident-page usage, which accounts for the better performance of LRU. This algorithm is sometimes referred to as Clock, and it is also known as not recently used (NRU). The algorithm makes use of the referenced bit, which is associated with each resident page. The referenced bit is set whenever the related page is reference and cleared occasionally by software. Its setting indicates whether a given page has been referenced in the recent past. How recent this past is depends on the frequency of the referenced-bit resetting. The page-replacement routine makes use of this information when selecting a victim for removal.

The algorithm is usually implemented by maintaining a circular list of the resident pages and a pointer to the page where it left off. The algorithm works by sweeping the page list and resetting the presence bit of the pages that it encounters. This sweeping motion of the circular list resembles the movement of the clock hand, hence the name clock. The clock algorithm seeks and evicts pages not recently used in order to free page frames for allocation to demanding processes. When it encounters a page whose reference bit is cleared, which means that the related page has not been referenced since the last sweep, the algorithm acts as follows:

(1) If the page is modified, it is marked for clearing & scheduled for writing to disk.

(2) If the page is not modified, it is declared non-resident, and the page frames that it occupies are feed.

The algorithm continues its operation until the required numbers of page frames are freed. The algorithm may be invoked at regular time intervals or when the number of free page frames drops below a specified threshold.

Other approximations and variations on this theme are possible. Some of them track page usage more accurately by means of a reference counter that counts the number of sweeps during which a given page is found to be un-referenced. Another possibility is to record the states of referenced bits by shifting them occasionally into related bit arrays. When a page is to be evicted, the victim is chosen by comparing counters or bit arrays in order to find the least frequently reference page. The general idea is to devise an implementable algorithm that bases its decisions on measured page usage and thus takes into account the program behavior patterns.

9.3.3.6 Allocation Policies

The allocation policy must compromise among conflicting requirements such as:

(a) Reduced page-fault frequency,

(b) Improved turn-around time,

(c) Improved processor utilization, etc.

Giving more real pages to a process will result in reduced page-fault frequency and improved turnaround time. But it reduces the number of active processes that may coexist in memory at a time resulting into the lower processor utilization factor. On the other hand, if too few pages are allocated to a process, its page- fault frequency and turnaround times may deteriorate.

Another problem caused by under-allocation of real pages may be encountered in systems that opt for restarting of faulted instructions. If fewer pages are allocated to a process than are necessary for execution of the restartable instruction that causes the largest number of page faults in a given architecture, the system might fault continuously on a single instruction and fail to make any real progress.

Consider a two-address instruction, such as Add @X, @Y, where X and Y are virtual addresses and @ denotes indirect addressing. Assuming that the operation code and operand addresses are

encoded in one word each, this instruction need three words for storage. With the use of indirect addressing, eight memory references are needed to complete execution of this instruction: three to fetch the instruction words, two to fetch operand addresses, two to access the operands themselves (indirect addressing), and one to store the result. In the worst case, six different pages may have to reside in memory concurrently in order to complete execution of this instruction: two if the instruction crosses a page boundary, two holding indirect addresses, and two holding the target operands. A likely implementation of this instruction calls for the instruction to be restarted after a page fault. If so, with fewer than six pages allocated to the process that executes it, the instruction may keep faulting forever. In general, the lower limit on the number of pages imposed by the described problem is architecture-dependent. In any particular implementation, the appropriate bound must be evaluated and built into the logic of the allocation routine.

While we seem to have some guidance as to the minimal number of pages, the reasonable maximum remains elusive. It is also unclear whether a page maximum should be fixed for a given system or determined on an individual basis according to some specific process attributes. Should the maximum be defined statically or dynamically, in response to system resource utilization and availability, and perhaps in accordance with the observable behavior of the specific process?

From the allocation module's point of view, the important conclusion is that each program has a certain threshold regarding the proportion of real to virtual pages, below which the number of page faults increases very quickly. At the high end, there seems to be a certain limit on the number of real pages, above which an allocation of additional real memory results in little or in moderate performance improvement. Thus, we want to allocate memory in such a way that each active program is between these two extremes.

Being program-specific, the upper and lower limits should probably not be fixed but derived dynamically on the basis of the program faulting behavior measured during its execution. When resource utilization is low, activating more processes may increase the degree of multiprogramming. However, the memory manager must keep track of the program behavior when doing so. A process that experiences a large number of page faults should be either

allocated more memory or suspended otherwise. Likewise, a few pages may be taken away from a process with a low page-fault rate without great concern. In addition, the number of pages allocated to a process may be influenced by its priority (higher priority may indicate that shorter turnaround time is desirable), the amount of free memory, fairness, and the like.

Thrashing: Although the complexity and overhead of memory allocation should be within a reasonable bound, the use of oversimplified allocation algorithms has the potential of crippling the system throughput. If real memory is over-allocated to the extent that most of the active programs are above their upper page-fault rate thresholds, the system may exhibit a behavior known as thrashing. With very frequent page faults, the system spends most of its time shuttling pages between main memory and secondary memory. Although the disk I/O channel may be overloaded by this activity, but processor utilization is reduced.

One way of introducing thrashing behavior is dangerously logical and simple. After observing a low processor utilization factor, the OS may attempt to improve it by activating more processes. if no free pages are available, the holdings of the already-active processes may be reduced. This may drive some of the processes into the high page-fault zone. As a result, the processor utilization may drop while the processes are awaiting their pages to be brought in. In order to improve the still-decreasing processor utilization, the OS may decide to increase the degree of multi-programming even further. Still more pages will be taken away from the already-depleted holdings of the active processes, and the system is hopelessly on its way to thrashing. It is obvious that global replacement strategies are susceptible to thrashing. Thus a good design must make sure that the allocation algorithm is not unstable and inclined toward thrashing. Knowing the typical patterns of program behavior, we want to ensure that no process is allocated too few pages for its current needs. Too few pages may lead to thrashing, and too many pages may unduly restrict the degree of multi-programming and processor utilization.

Page-Fault Frequency (PFF)

This policy uses an upper and lower page-fault frequency threshold to decide for allocation of new page frames. The PFF parameter P may be defined as: P = 1/T Where T is the critical inter-page fault time. P is usually measured in number of page faults per millisecond. The PFF algorithm may be implemented as follows:

1. The OS defines a system-wide (or per-process) critical page-fault frequency, P.
2. The OS measures the virtual (process) time and stores the time of the most recent page fault in the related process control block.

When a page fault occurs, the OS acts as follows:

- If the last page fault occurred less than T = 1/P ms ago, the process is operating above the PFF threshold, and a new page frame is added from the pool to house the needed page.
- Otherwise, the process is operating below the PFF threshold P, and a page frame occupied by a page whose reference bit and written-into bit are not set is freed to accommodate the new page.
- The OS sweeps and resets referenced bits of all resident pages. Pages that are found to be unused, unmodified, and not shared since the last sweep are released, and the freed page frames are returned to the pool for future allocations.

For completeness, some policies need to be employed for process activation and deactivation to maintain the size of the pool of free page frames within desired limits.

9.3.3.7 Hardware Support and Considerations

Virtual memory requires:

(1) instruction interruptibility and restart ability,

(2) a collection of page status bits associated with each page descriptor,

(3) And if based on paging - a TLB to accelerate address translations.

Choice of the page size can have a significant impact on performance of a virtual-memory system. In most implementations, one each of the following bits is provided in every page descriptor:

- Presence bit, used to aid detection of missing items by the mapping hardware
- Written-into (modified) bit, used to reduce the overhead incurred by the writing of unmodified replaced pages to disk
- Referenced bit, used to aid implementation of the replacement policy

An important hardware accelerator in virtual-memory systems is the TLB. Although system architects and hardware designers primarily determine the details of the TLB operation, the management of TLB is of interest because it deals with problems quite similar to those discussed in the more general framework of virtual memory. TLB hardware must incorporate allocation and replacement policies so as to make the best use of the limited number of mapping entries that the TLB can hold. An issue in TLB allocation is whether to devote all TLB entries to the running process or to distribute them somehow among the set of active processes. The TLB replacement policy governs the choice of the entry to be evicted when a miss occurs and another entry needs to be brought in.

Allocation of all TLB entries to the running process can lead to relatively lengthy initial periods of "loading" the TLB whenever a process is scheduled. This can lead to the undesirable behavior observed in some systems when an interrupt service routine (ISR) preempts the running process. Since a typical ISR is only a few hundred instructions long, it may not have enough time to load the TLB. This can result in slower execution of the interrupt service routine due to the need to reference PMT in memory while performing address translations. Moreover, when the interrupted process is resumed, its performance also suffers from having to load the TLB all over again. One way to combat this problem is to use multi-context TLBs that can contain and independently manage the PMT entries of several processes. With a multi-context TLB, when a process is scheduled for execution, it may find some of its PMT entries left over in the TLB from the preceding period of activity. Management of such TLBs requires the identity of the corresponding process to be associated with each entry, in order to make sure that matches are made only with the TLB entries belonging to the process that produced the addresses to be mapped. Removal of TLB entries is usually done after each miss. If PMT entries of several processes are in the buffer, the victim may be chosen either locally or globally. Understandably, some preferential treatment is usually given to holdings of the running process. In either case, least recently used is a popular strategy for replacement of entries.

The problem of maintaining consistency between the PMT entries and their TLB copies in the presence of frequent page moves must also be tackled by hardware designers. Its solution

usually relies on some specialized control instructions for TLB flushing or for it selective invalidation. Another hardware-related design consideration in virtual-memory systems is whether I/O devices should operate with real or virtual addresses.

A hardware/software consideration involved in the design of paged systems is the choice of the page size. Primary factors that influence this decision are

(1) Memory utilization and cost.

(2) Page-transport efficiency.

Page-transport efficiency refers to the performance cost and overhead of fetching page from the disk or, in a diskless workstation environment, across the network. Loading of a page from disk consists of two basic components: the disk-access time, and the page-transfer time. Head positioning delays generally exceed disk- memory transfer times by order of magnitude. Thus, total page-transfer time tends to be dominated by the disk positioning delay, which is independent of the page size.

Small page size reduces page breakage, and it may make better use of memory by containing only a specific locality of reference. On the other hand, small pages may result in excessive size of mapping tables in virtual systems with large virtual-address spaces. Page-transport efficiency is also adversely affected by small page sizes, since the disk-accessing overhead is imposed for transferring a relatively small group of bytes.

Large pages tend to reduce table fragmentation and to increase page-transport efficiency. This is because the overhead of disk accessing is amortized over a larger number of bytes whenever a page is transferred between disk and memory. On the negative side, it may impact memory utilization by increasing page breakage and by spanning more than one locality of reference. If multiple localities contained in a single page have largely dissimilar patterns of reference, the system may experience reduced effective memory utilization and wasted I/O bandwidth.

9.3.3.8 Protection and Sharing

The frequent moves of items between main and secondary memory may complicate

the management of mapping tables in virtual systems. When several arties share an item in real memory, the mapping tables of all involved processes must point to it. If the shared item is selected

for removal, all concerned mapping tables must be updated accordingly. The overhead involved tends to outweigh the potential benefit of removing shared items. Many systems simplify the management of mapping tables by fixing the shared objects in memory. An interesting possibility provided by large virtual-address spaces is to treat the OS itself as a shared object. As such, the OS is mapped as a part of each user's virtual space. To reduce table fragmentation, dedicated mapping registers are often provided to access a single physical copy of the page-map table reserved for mapping references to the OS. One or more status bits direct the mapping hardware to use the public or private mapping table, as appropriate for each particular memory reference. In this scheme, different users have different access rights to portions of the OS. Moreover, the OS-calling mechanism may be simplified by avoiding expensive mode switches between users and the OS code. With the protection mechanism provided by mapping, a much faster CALL instruction, or its variant, may be used to invoke the OS

.9.3.4 Segmentation and Paging

It is also possible to implement virtual memory in the form of demand segmentation inheriting the benefits of sharing and protection provided by segmentation. Moreover, their placement policies are aided by explicit awareness of the types of information contained in particular segments. For example, a "working set" of segments should include at least one each of code, data, and stack segments. As with segmentation, inter-segment references alert the OS to changes of locality. However, the variability of segment sizes and the within-segment memory contiguity requirement complicate the management of both main and secondary memories. Placement strategies are quite complex in segmented systems. Moreover, allocation and deallocation of variable-size storage areas to hold individual segments on disk imposes considerably more overhead than handling of pages that are usually designed to fit in a single disk block.

On the other hand, paging is very easy for the management of main and secondary memories, but it is inferior with regard to protection and sharing. The transparency of paging necessitates the use of probabilistic replacement algorithms which virtually no guidance from users, they are forced to operate mainly on the basis of their observations of program behavior. Both segmented and paged implementations of virtual memory have their advantages/disadvantages. Some systems combine the two approaches in order to enjoy the benefits of both. One approach is to use segmentation from the

user's point of view but to divide each segment into pages of fixed size for purposes of allocation. In this way, the combined system retains most of the advantages of segmentation. At the same time, the problems of complex segment placement and management of secondary memory are eliminated by using paging. The principle of address translation in combined segmentation and paging systems is shown in Figure 8. Both segment descriptor tables and PMT are required for mapping. Instead of containing the base and limit of the corresponding segment, each entry of the SDT contains the base address and size of the PMT to be used for mapping of the related segment's pages. The presence bit in each PMT entry indicates availability of the corresponding page in the real memory. Access rights are recorded as a part of segment descriptors, although they may be placed or refined in the entries of the PMT. Each virtual address consists of three fields: segment number, page number, and offset within the page. When a virtual address is presented to the mapping hardware, the segment number is used to locate the corresponding PMT. Provided that the issuing process is authorized to make the intended type of reference to the target segment; the page number is used to index the PMT. If the presence bit is set, obtaining the page-frame address from the PMT and combining this with the offset part of the virtual address complete the mapping. If the target page is absent from real memory, the mapping hardware generates a page-fault exception, which is processed.

At both mapping stages, the length fields are used to verify that the memory references of the running process lie within the confines of it address space. Many variations of this powerful scheme are possible. For example, the presence bit may be included with entries of the SDT. It may be cleared when no pages of the related segment are in real memory. When such a segment is referenced, bringing several of list pages into main memory may process the segment fault. In general, page re-fetching has been more difficult to implement in a way that performs better than demand paging. One of the main reasons for this is the inability to predict the use of previously un-referenced pages. However, referencing of a particular segment increases the probability of its constituent pages being referenced.

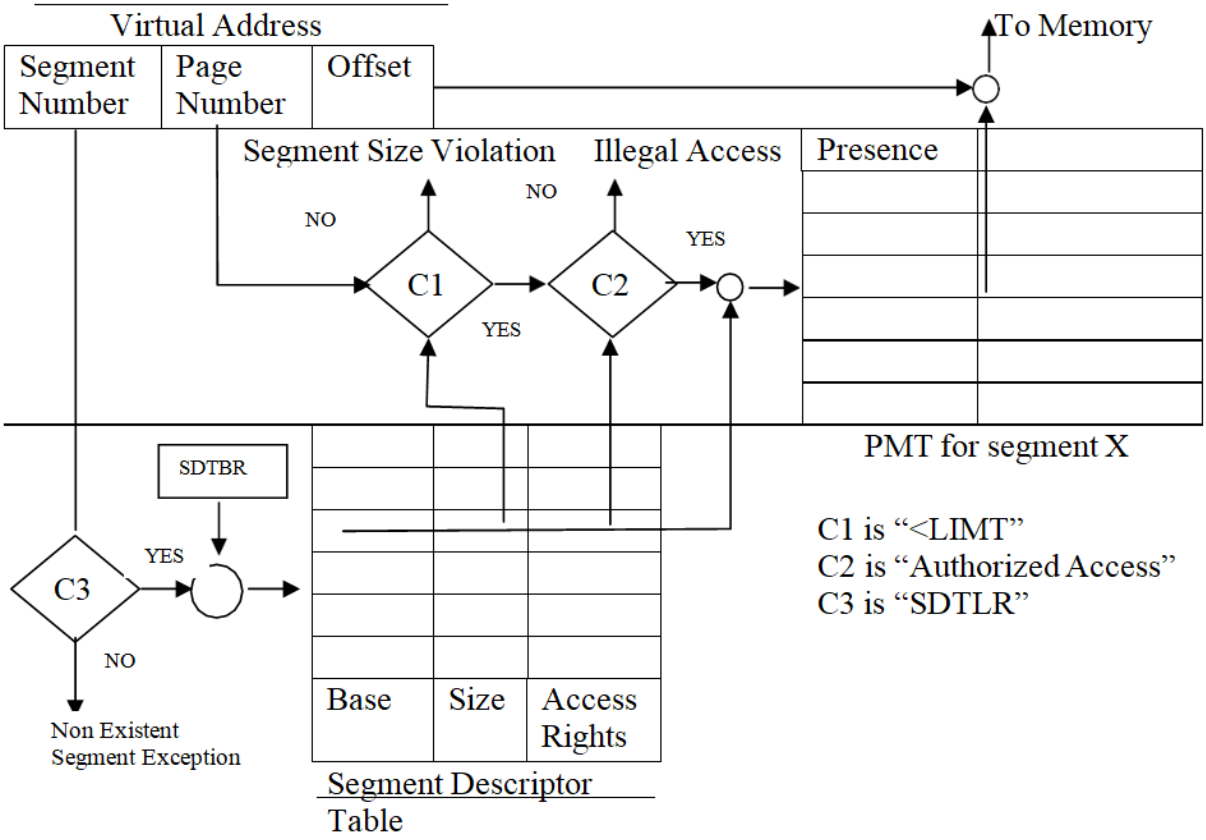


Figure 8 – Segmentation and paging

While the combination of segmentation and paging is certainly appealing, it requires two memory accesses to complete the mapping of each virtual address resulting into the reduction of the effective memory bandwidth by two-thirds.

9.4 Check your progress

1. What is Address Binding?
   a) going to an address in memory
   b) locating an address with the help of another address
   c) binding two addresses together to form a new address in a different memory space
   d) a mapping from one address space to another
2. If the process can be moved during its execution from one memory segment to another, then binding must be _____
   a) delayed until run time

b) preponed to compile time

c) preponed to load time

d) none of the mentioned

3. The _____ swaps processes in and out of the memory.

a) Memory manager

b) CPU

c) CPU manager

d) User

4. In a system that does not support swapping _____

a) the compiler normally binds symbolic addresses (variables) to relocatable addresses

b) the compiler normally binds symbolic addresses to physical addresses

c) the loader binds relocatable addresses to physical addresses

d) binding of symbolic addresses to physical addresses normally takes place during execution

5. CPU fetches the instruction from memory according to the value of _____

a) program counter

b) status register

c) instruction register

d) program status word

6. A memory buffer used to accommodate a speed differential is called _____

a) stack pointer

b) cache

c) accumulator

d) disk buffer

7. Which one of the following is the address generated by CPU?

a) physical address

b) absolute address

c) logical address

d) none of the mentioned

Run time mapping from virtual to physical address is done by _____

a)Memorymanagementunit

b)CPU

c)PCI

d) None of the mentioned

9.5 Summary

Segmentation allows breaking of the virtual address space of a single process into separate entities that may be placed in noncontiguous areas of physical memory. As a result, the virtual-to-physical address translation at instruction execution time in such systems is more complex, and some dedicated hardware support is necessary to avoid a drastic reduction in effective memory bandwidth. Since average segment sizes are usually smaller than average process sizes, segmentation can reduce the impact of external fragmentation on the performance of systems with dynamically partitioned memory. Other advantages of segmentation include dynamic relocation, finely grained protection both within and between address spaces, ease of sharing, and facilitation of dynamic linking and loading.

No doubt segmentation reduces the impact of fragmentation and offers superior protection and sharing by dividing each process's address space into logically related entities that may be placed into non-contiguous areas of physical memory. But paging simplifies allocation and de-allocation of memory by dividing address spaces into fixed-sized chunks. Execution-time translation of virtual to physical addresses, usually assisted by hardware, is used to bridge the gap between contiguous virtual addresses and non-contiguous physical addresses where different pages may reside.

It is very desirable to execute a process whose logical address space is larger than the available physical address space and the option is virtual memory. Virtual memory removes the restriction on the size of address spaces of individual processes that is imposed by the capacity of the physical memory installed in a given system. In addition, virtual memory provides for dynamic migration of portions of address spaces between primary and secondary memory in accordance with the relative frequency of usage.

If the total memory requirement is larger than the available physical memory, then memory

management system has to create the house for new pages by replacing some pages from the memory. A number of page replacement policies have been proposed such as FIFO, LRU, NRU, etc with their merits and demerits. FIFO implementation is easy but suffers from Belady anomaly. Optimal replacement requires future knowledge. LRU is n approximation of optimal but difficult to implement. After page replacement, there is the need for frame allocation policy. An improper allocation policy may result into thrashing.

It is also possible to implement virtual memory in the form of demand segmentation inheriting the benefits of sharing and protection provided by segmentation but placement strategies are complex, allocation and deallocation of variable-size storage areas to hold individual segments on disk imposes more overhead. On the other hand, paging is very easy for the management of main and secondary memories, but it is inferior with regard to protection and sharing. Some systems combine the two approaches in order to enjoy the benefits of both.

9.6 Keywords

External Fragmentation is waste of memory between partitions caused by scattered non-contiguous free space.

Internal fragmentation is waste of memory within a partition caused by difference between size of partition and the process allocated.

Page: The virtual-address space of a process is divided into fixed-size blocks of the same size, called pages.

Page fault: The phenomenon of not finding a referenced page in the memory is known a page fault.

TLB (Translation Look aside Buffer): It is a high-speed associative memory, used to speed up memory access, by for storing a subset of often-used page- map table entries.

PMT (Page Map Table): It is a table used to translate a virtual address into actual physical address in paging system.

Locality of Reference: There is a strong tendency of programs to favor subsets of their address spaces during execution. This phenomenon is known as locality of reference.

9.7 Self Assessment Questions (SAQ)

1. Define segmentation and write a detailed note on the address translation mechanism in segmentation.
2. Write a detailed mote on sharing in segmentation. How the access rights are implementation in sharing in segmentation.
3. What is the basic difference between paging and segmentation? Which one is better and why?
4. What is Table Look aside Buffer (TLB)? How is it used to speed up the memory access? Explain.
5. Write short notes on following:
    (a) Thrashing
    (b) Page fault frequency
    (c) Sharing in virtual memory
6. Differentiate between following:
    a. Dirty page and clean page
    b. Logical Address and Physical Address
    c. Spatial and temporal locality of reference
    d. Segmentation and paging
7. What is the common drawback of all the real memory management techniques? How is it overcome in virtual memory management schemes?
8. What extra hardware do we require for implementing demand paging and demand segmentation?
9. Show that LRU page replacement policy possesses the stack property.
10. Differentiate between internal and external fragmentation.
11. What do you understand by thrashing? What are the factors causing it?
12. Compare FIFO page replacement policy with LRU page replacement on the basis of

overhead.

9.8 Answer to check your progress

1. a mapping from one address space to another
2. delayed until run time
3. Memory manager
4. the compiler normally binds symbolic addresses (variables) to relocatable addresses
5. program counter
6. cache
7. logical address
8. Memory management unit

9.9 Suggested Readings / Reference Material

1. Operating System Concepts, 5th Edition, Silberschatz A., Galvin P.B., John Wiley & Sons.

Systems Programming & Operating Systems, 2nd Revised Edition, Dhamdhere D.M., Tata McGraw Hill Publishing Company Ltd., New Delhi.

2. Operating Systems, Godbole A.S., Tata McGraw Hill Publishing Company Ltd., New Delhi.

3. Operating Systems, Madnick S.E., Donovan J.T., Tata McGraw Hill Publishing Company Ltd., New Delhi.

4. Operating Systems-A Modern Perspective, Gary Nutt, Pearson Education Asia, 2000.

5. Operating Systems, Harris J.A., Tata McGraw Hill Publishing Company Ltd., New Delhi, 2002.

Course: MCA-32	Writer: Ritu
Lesson: 10	Vetter:

Memory Management II

Structure

10.0 Learning Objectives

The objective of this lesson is to make the students familiar with the following concepts of

- Hardware Basics
- The CPU
- Address Binding
- Logical and physical address space

10.1 Introduction

A computer is an electronic device, operating under the control of instructions stored in its own memory that can accept data (input), process the data according to specified rules, produce information (output), and store the information for future use. All physical components that make up a computer are known

as computer hardware. It includes all components that we can see and touch i.e. processor, input devices like keyboard, mouse, output devices like visual display unit (VDU), printer, speaker, connecting wires, casing, storage devices etc. The Address Binding refers to the mapping of computer instructions and data to physical memory locations. Both logical and physical addresses are used in computer memory. It assigns a physical memory region to a logical pointer by mapping a physical address to a logical address known as a virtual address. It is also a component of computer memory management that the OS performs on behalf of applications that require memory access.

10.2 Hardware Basics

An operating system has to work closely with the hardware system that acts as its foundations. The operating system needs certain services that can only be provided by the hardware. In order to fully understand the Linux operating system, you need to understand the basics of the underlying hardware. This chapter gives a brief introduction to that hardware: the modern PC.

When the ``Popular Electronics'' magazine for January 1975 was printed with an illustration of the Altair 8080 on its front cover, a revolution started. The Altair 8080, named after the destination of an early Star Trek episode, could be assembled by home electronics enthusiasts for a mere $397. With its Intel 8080 processor and 256 bytes of memory but no screen or keyboard it was puny by today's standards. Its inventor, Ed Roberts, coined the term ``personal computer'' to describe his new invention, but the term PC is now used to refer to almost any computer that you can pick up without needing help.

By this definition, even some of the very powerful Alpha AXP systems are PCs. Enthusiastic hackers saw the Altair's potential and started to write software and build hardware for it. To these early pioneers it represented freedom; the freedom from huge batch processing mainframe systems run and guarded by an elite priesthood. Overnight fortunes were made by college dropouts fascinated by this new phenomenon, a computer that you could have at home on your kitchen table. A lot of hardware appeared, all different to some degree and software hackers were happy to write software for these new machines. Paradoxically it was IBM who firmly cast the mould of the modern PC by announcing the IBM PC in 1981 and shipping it to customers early in 1982. With its Intel 8088 processor, 64K of memory (expandable to 256K), two floppy disks and an 80 character by 25 lines Colour Graphics Adapter (CGA) it was not very powerful by today's standards but it sold well. It was followed, in 1983, by the IBM PC-XT which had the luxury of a 10Mbyte hard drive. It was not long before IBM PC

clones were being produced by a host of companies such as Compaq and the architecture of the PC became a de-facto standard. This de-facto standard helped a multitude of hardware companies to compete together in a growing market which, happily for consumers, kept prices low. Many of the system architectural features of these early PCs have carried over into the modern PC. For example, even the most powerful Intel Pentium Pro based system starts running in the Intel 8086's addressing mode. When Linus Torvalds started writing what was to become Linux, he picked the most plentiful and reasonably priced hardware, an Intel 80386 PC.

Looking at a PC from the outside, the most obvious components are a system box, a keyboard, a mouse and a video monitor. On the front of the system box are some buttons, a little display showing some numbers and a floppy drive. Most systems these days have a CD ROM and if you feel that you have to protect your data, then there will also be a tape drive for backups. These devices are collectively known as the peripherals.

PC

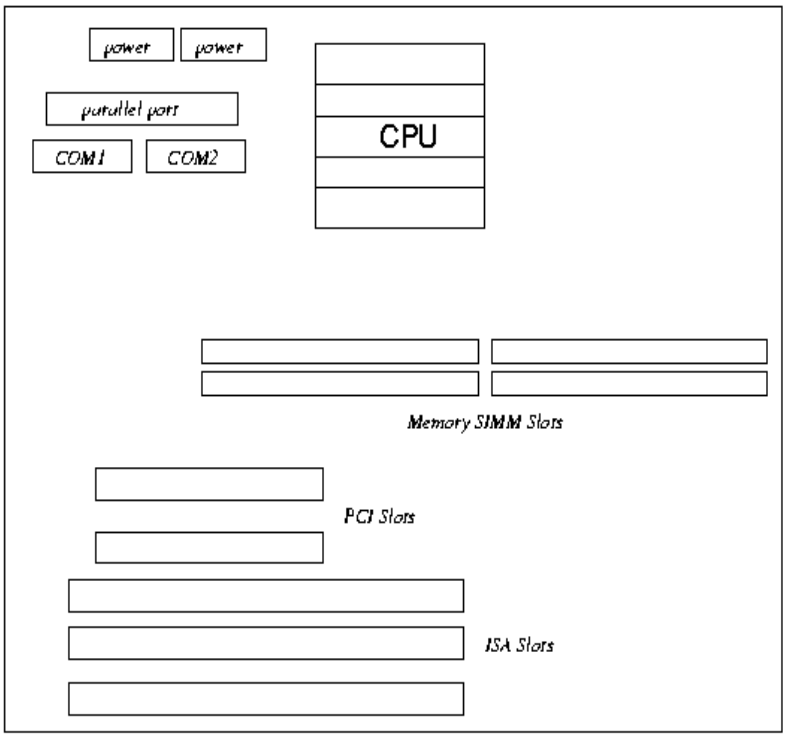


Figure 1.1: A typical PC motherboard.

Although the CPU is in overall control of the system, it is not the only intelligent device. All of the peripheral controllers, for example the IDE controller, have some level of intelligence. Inside the PC (Figure) you will see a motherboard containing the CPU or microprocessor, the memory and a number of slots for the ISA or PCI peripheral controllers. Some of the controllers, for example the IDE disk controller may be built directly onto the system board.

Hardware – any physical device or equipment used in or with a computer system (anything you can see and touch).

10.2.1 External hardware

- External hardware devices (peripherals) – any hardware device that is located outside the computer.
- Input device – a piece of hardware device which is used to enter information to a computer for processing.
- Examples: keyboard, mouse, track pad (or touchpad), touch screen, joystick, microphone, light

- Pen, webcam, speech input, etc.
- Output device – a piece of hardware device that receives information from a computer.

- Examples: monitor, printer, scanner, speaker, display screen (tablet, Smartphone), projector, head phone, etc.



10.2.2 Internal hardware

- Internal hardware devices (or internal hardware components) – any piece of hardware device that is located inside the computer.
- Examples: CPU, hard disk drive, ROM, RAM, etc.

10.2.3 Computer software

- Software – a set of instructions or programs that tells a computer what to do or how to perform a specific task (computer software runs on hardware).
- Main types of software – systems software and application software.

10.2.4 Application software

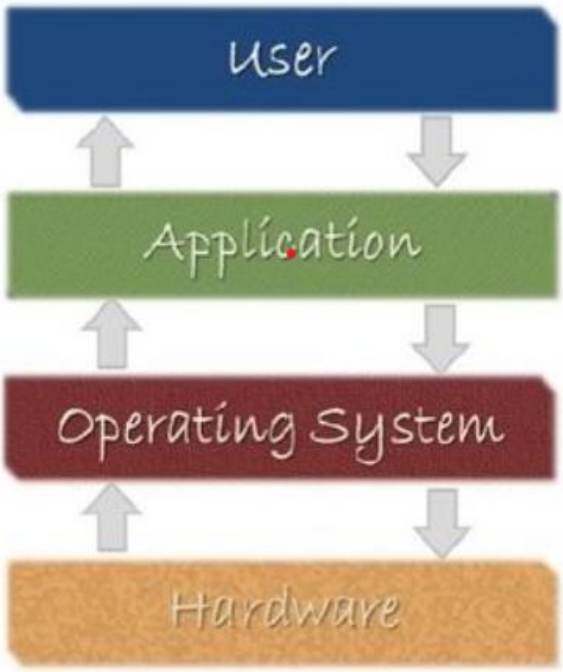
- Application software – a computer program that provides users with tools to accomplish a specific task.

- Examples of application software: word processing, spreadsheets, presentation, database management, Internet browsers, email programs, media players, accounting, pronunciation, translation, desktop publishing, enterprise, etc.

10.2.5 System Software

System software – it is designed to run a computer's hardware and application software, and make the computer system available for use. It serves as the interface between hardware, application software, and the user.

- Main functions of system software – allocating system resources, managing storage space, storing and retrieval of files, providing security, etc.
- Main types of systems software – operating system, device driver, utility software, programming software, etc.



10.2.6 Operating system(OS) – software that controls and coordinates the computer hardware devices and runs other software and applications on a computer. It is the main part of system software and a computer will not function without it.

- Main functions of an operating system – booting the computer, managing system resources (CPU, memory, storage devices, printer, etc.), managing files, handling input and output, executing and providing services for application software, etc.
- Examples of operating system: Microsoft Windows, Apple iOS, Android OS, macOS, Linux, etc.

10.2.7 Device driver – a software program that is designed to control a particular hardware device that is attached to a computer.

- The main purpose of device driver – it acts as a translator between the hardware device and operating systems or applications that use it.
- It instructs computer on how to communicate with the device by translating the operating system's instructions into a language that a device can understand in order to perform the necessary task.
- Examples of device driver: printer driver, display driver, USB driver, sound card driver, motherboard driver, ROM driver, etc.

10.2.8 Utility software – a type of system software that helps set up, analyze, configure, strengthen, maintain a computer and performs a very specific task (e.g. antivirus software, backup software, memory tester, screen saver, etc.

10.3 Fragmentation

The process of dividing a computer file, such as a data file or an executable program file, into fragments that are stored in different parts of a computer's storage medium, such as its hard disc or RAM, is known as fragmentation in computing. When a file is fragmented, it is stored on the storage medium in non-contiguous blocks, which means that the blocks are not stored next to each other.

10.3.1 Cause of Fragmentation

This can happen when a file is too large to fit into a single contiguous block of free space on the storage medium, or when the blocks of free space on the medium are insufficient to hold the file. Because the system must search for and retrieve individual fragments from different locations in order to open the file, fragmentation can cause problems when reading or accessing the file.

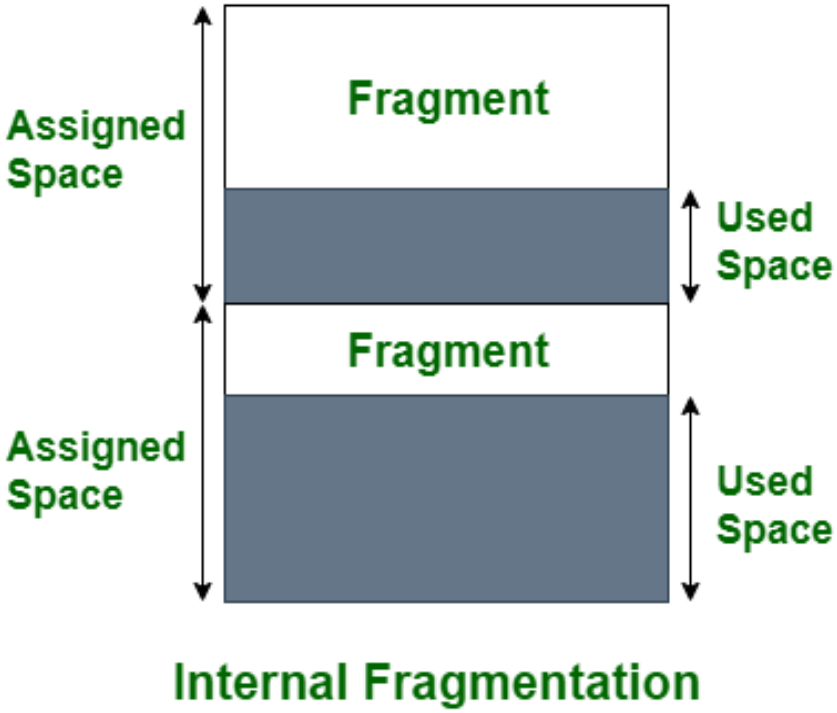
10.3.2 Effect of Fragmentation

This can reduce system performance and make it more difficult to access the file. It is generally best to defragment your hard disc on a regular basis to avoid fragmentation, which is a process that rearranges the blocks of data on the disc so that files are stored in contiguous blocks and can be accessed more quickly.

10.3.3 Types of Fragmentation

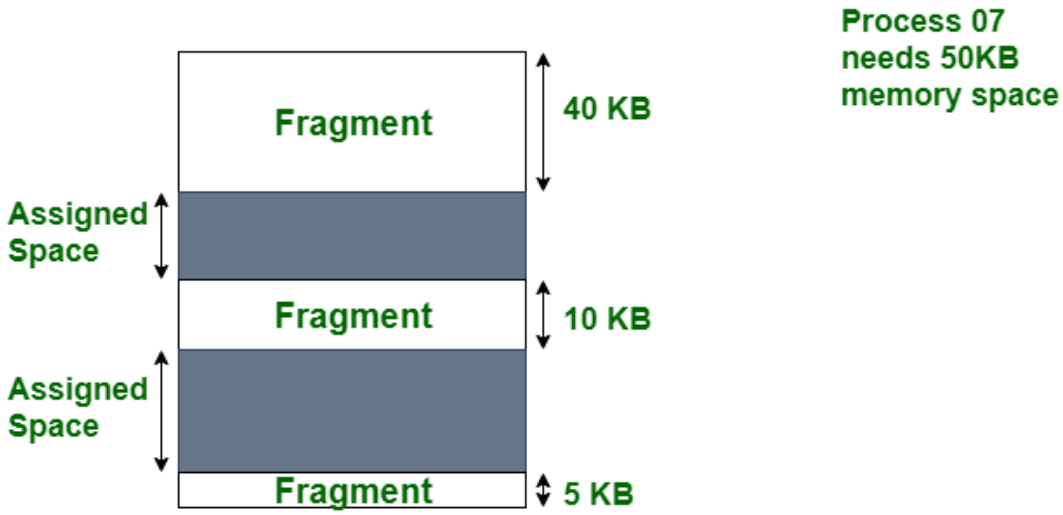
There are two main types of fragmentation:

10.3.3.1 Internal fragmentation: Internal fragmentation occurs when there is unused space within a memory block. For example, if a system allocates a 64KB block of memory to store a file that is only 40KB in size, that block will contain 24KB of internal fragmentation. When the system employs a fixed-size block allocation method, such as a memory allocator with a fixed block size, this can occur.



10.3.3.2 External fragmentation: External fragmentation occurs when a storage medium, such as a hard disc or solid-state drive, has many small blocks of free space scattered throughout it. This can

happen when a system creates and deletes files frequently, leaving many small blocks of free space on the medium. When a system needs to store a new file, it may be unable to find a single contiguous block of free space large enough to store the file and must instead store the file in multiple smaller blocks. This can cause external fragmentation and performance problems when accessing the file.



Fragmentation can also occur at various levels within a system. File fragmentation, for example, can occur at the file system level, in which a file is divided into multiple non-contiguous blocks and stored on a storage medium. Memory fragmentation can occur at the memory management level, where the system allocates and deallocated memory blocks dynamically. Network fragmentation occurs when a packet of data is divided into smaller fragments for transmission over a network.

10.3.4 How Fragmentation affects the system?

- **Slower Performance:** Fragmentation slows down the read and writes speed of the disk, as the disk head has to move to different locations to access the fragments of a file. This, in turn, increases the access time and reduces the overall speed of the system, causing slowdowns and lag in applications.
- **Disk Space Wasting:** Fragmentation can also lead to disk space being wasted, as fragments may occupy more space than required. This can result in a shortage of disk space, causing the system to become unstable and vulnerable to crashes or errors.

- **Data Loss:** In severe cases, fragmentation can also cause the system to run out of disk space, leading to data loss. This can be particularly detrimental to users who rely on their systems to store critical information and data.
- **Increased Risk of System Crashes:** The more fragmented a disk becomes, the more prone it is to crashes and errors. This can cause the system to become unstable and vulnerable to data loss, crashes, and other issues.
- **Reduced Battery Life:** Fragmentation can also negatively impact the battery life of laptops and other mobile devices. This is because fragmentation requires the disk to work harder, which in turn puts additional strain on the battery.

10.3.5 Role of Fragmentation in the Performance of an Operating System

Fragmentation is a critical issue that can significantly impact the performance of an operating system. The primary role of fragmentation is to slow down the read and write speed of the disk. As the disk head has to move to different locations to access the fragments of a file, the access time increases, reducing the overall speed of the system. This can lead to a decrease in system performance, causing slowdowns and lag in applications. Moreover, fragmentation can also lead to disk space being wasted, as fragments may occupy more space than required. This can result in a shortage of disk space, causing the system to become unstable and vulnerable to crashes or errors. In severe cases, fragmentation can also cause the system to run out of disk space, leading to data loss.

To maintain optimal performance, it is essential to regularly defragment the disk. Defragmentation reorganizes the fragments of a file and allocates contiguous disk space to store the file. This helps to improve the read and write speed of the disk, reducing access time and increasing the overall speed of the system. By regularly defragmenting the disk, the performance of the operating system can be improved and maintained, ensuring a smooth and efficient user experience.

Advantages:

There are several potential benefits to fragmentation:

- It makes better use of storage space on a computer’s hard disc or other storage medium. When a file is fragmented, it can be stored in the medium’s available free-space blocks, even if those

blocks are not contiguous. This is especially useful if the medium contains a large number of small blocks of free space that would otherwise be wasted.

- It can improve system performance by reducing the amount of seeking required by the system's hard disc drive (HDD) or solid-state drive (SSD). When a file is stored on the medium in contiguous blocks, the HDD or SSD must move its read/write head to different locations on the disc to access the various blocks.
- It can enable a system to store larger files than it could otherwise. If the system does not have a single contiguous block of free space large enough to store a specific file, the file can be fragmented and stored in multiple smaller blocks.

However, it is generally recommended to minimize fragmentation whenever possible, as it can have a negative impact on system performance and make accessing and managing files more difficult.

Disadvantages:

- It can degrade system performance, particularly when reading or accessing fragmented files. When a system needs to access a fragmented file, it must search for and retrieve the various fragments from various locations on the storage medium, which can take longer than accessing a contiguous file. This can reduce system performance and make it more difficult to access the file.
- It can make managing and organizing files on a system more difficult. When a file is fragmented, it is stored on the storage medium in multiple non-contiguous blocks, making it more difficult to locate and access the file.
- It has the potential to shorten the lifespan of a storage medium, such as a hard disc or solid-state drive. When a system writes data to a storage medium, it can cause the medium to wear out. Data stored in non-contiguous blocks can cause more wear and tear on the medium than data stored in contiguous blocks, potentially reducing the medium's lifetime.
- It can consume more system storage space. When a file is fragmented, the system must store additional information on the storage medium about the location of each fragment. This can consume more space and limit the amount of space available for other files.

10.4 The CPU

The CPU, or rather microprocessor, is the heart of any computer system. The microprocessor calculates, performs logical operations and manages data flows by reading instructions from memory and then executing them. In the early days of computing the functional components of the microprocessor were separate (and physically large) units. This is when the term *Central Processing Unit* was coined. The modern microprocessor combines these components onto an integrated circuit etched onto a very small piece of silicon. The terms *CPU*, *microprocessor* and *processor* are all used interchangeably in this book. Microprocessors operate on binary data; that is data composed of ones and zeros. These ones and zeros correspond to electrical switches being either on or off. Just as 42 is a decimal number meaning ``4 10s and 2 units", a binary number is a series of binary digits each one representing a power of 2. In this context, a power means the number of times that a number is multiplied by itself. 10 to the power 1 (10^1) is 10, 10 to the power 2 (10^2) is 10x10, 10^3 is 10x10x10 and so on. Binary 0001 is decimal 1, binary 0010 is decimal 2, binary 0011 is 3, binary 0100 is 4 and so on. So, 42 decimalsis 101010 binary or (2 + 8 + 32 or $2^1 + 2^3 + 2^5$). Rather than using binary to represent numbers in computer programs, another base, hexadecimal is usually used. In this base, each digital represents a power of 16. As decimal numbers only go from 0 to 9 the numbers 10 to 15 are represented as a single digit by the letters A, B, C, D, E and F. For example, hexadecimal E is decimal 14 and hexadecimal 2A is decimal 42 (two 16s) + 10). Using the C programming language notation (as I do throughout this book) hexadecimal numbers are prefaced by ``*0x*"; hexadecimal 2A is written as *0x2A*.Microprocessors can perform arithmetic operations such as add, multiply and divide and logical operations such as ``is X greater than Y?".

The processor's execution is governed by an external clock. This clock, the system clock, generates regular clock pulses to the processor and, at each clock pulse, the processor does some work. For example, a processor could execute an instruction every clock pulse. A processor's speed is described in terms of the rate of the system clock ticks. A 100 MHz processor will receive 100,000,000 clock ticks every second. It is misleading to describe the power of a CPU by its clock rate as different processors perform different amounts of work per clock tick. However, all things being equal, a faster clock speed means a more powerful processor. The instructions executed by the processor are very simple; for example, ``read the contents of memory at location X into register Y". Registers are the microprocessor's internal storage, used for storing data and performing operations on it. The operations performed may cause the processor to stop what it is doing and jump to another instruction somewhere

else in memory. These tiny building blocks give the modern microprocessor almost limitless power as it can execute millions or even billions of instructions a second.

The instructions have to be fetched from memory as they are executed. Instructions may themselves reference data within memory and that data must be fetched from memory and saved there when appropriate. The size, number and type of register within a microprocessor is entirely dependent on its type. An Intel 4086 processor has a different register set to an Alpha AXP processor; for a start, the Intel's are 32 bits wide and the Alpha AXP's are 64 bits wide. In general, though, any given processor will have a number of general-purpose registers and a smaller number of dedicated registers. Most processors have the following special purpose, dedicated, registers:

10.4.1 Program Counter (PC)

This register contains the address of the next instruction to be executed. The contents of the PC are automatically incremented each time an instruction is fetched,

10.4.2 Stack Pointer (SP)

Processors have to have access to large amounts of external read/write random access memory (RAM) which facilitates temporary storage of data. The stack is a way of easily saving and restoring temporary values in external memory. Usually, processors have special instructions which allow you to push values onto the stack and to pop them off again later. The stack works on a last in first out (LIFO) basis. In other words, if you push two values, x and y, onto a stack and then pop a value off of the stack then you will get back the value of y.

Some processor's stacks grow upwards towards the top of memory whilst others grow downwards towards the bottom, or base, of memory. Some processor's support both types, for example ARM.

10.4.3 Processor Status (PS)

Instructions may yield results; for example ``is the content of register X greater than the content of register Y?" will yield true or false as a result. The PS register holds this and other information about the current state of the processor. For example, most processors have at least two modes of operation, kernel (or supervisor) and user. The PS register would hold information identifying the current mode.

All systems have a memory hierarchy with memory at different speeds and sizes at different points in the hierarchy. The fastest memory is known as cache memory and is what it sounds like - memory that is used to temporarily hold, or cache, contents of the main memory. This sort of memory is very fast but expensive, therefore most processors have a small amount of on-chip cache memory and more system based (on-board) cache memory. Some processors have one cache to contain both instructions and data, but others have two, one for instructions and the other for data. The Alpha AXP processor has two internal memory caches; one for data (the D-Cache) and one for instructions (the I-Cache). The external cache (or B-Cache) mixes the two together. Finally there is the main memory which relative to the external cache memory is very slow. Relative to the on-CPU cache, main memory is positively crawling.

The cache and main memories must be kept in step (coherent). In other words, if a word of main memory is held in one or more locations in cache, then the system must make sure that the contents of cache and memory are the same. The job of cache coherency is done partially by the hardware and partially by the operating system. This is also true for a number of major system tasks where the hardware and software must cooperate closely to achieve their aims.

10.4.5 Buses

The individual components of the system board are interconnected by multiple connection systems known as buses. The system bus is divided into three logical functions; the address bus, the data bus and the control bus. The address bus specifies the memory locations (addresses) for the data transfers. The data bus holds the data transferred. The data bus is bidirectional; it allows data to be read into the CPU and written from the CPU. The control bus contains various lines used to route timing and control signals throughout the system. Many flavours of bus exist, for example ISA and PCI buses are popular ways of connecting peripherals to the system.

10.4.6 Controllers and Peripherals

Peripherals are real devices, such as graphics cards or disks controlled by controller chips on the system board or on cards plugged into it. The IDE disks are controlled by the IDE controller chip and the SCSI disks by the SCSI disk controller chips and so on. These controllers are connected to the CPU and to each other by a variety of buses. Most systems built now use PCI and ISA buses to connect together the

main system components. The controllers are processors like the CPU itself, they can be viewed as intelligent helpers to the CPU. The CPU is in overall control of the system.

All controllers are different, but they usually have registers which control them. Software running on the CPU must be able to read and write those controlling registers. One register might contain status describing an error. Another might be used for control purposes; changing the mode of the controller. Each controller on a bus can be individually addressed by the CPU, this is so that the software device driver can write to its registers and thus control it. The IDE ribbon is a good example, as it gives you the ability to access each drive on the bus separately. Another good example is the PCI bus which allows each device (for example a graphics card) to be accessed independently.

10.4.6 Address Spaces

The system bus connects the CPU with the main memory and is separate from the buses connecting the CPU with the system's hardware peripherals. Collectively the memory space that the hardware peripherals exist in is known as I/O space. I/O space may itself be further subdivided, but we will not worry too much about that for the moment. The CPU can access both the system space memory and the I/O space memory, whereas the controllers themselves can only access system memory indirectly and then only with the help of the CPU. From the point of view of the device, say the floppy disk controller, it will see only the address space that its control registers are in (ISA), and not the system memory. Typically a CPU will have separate instructions for accessing the memory and I/O space. For example, there might be an instruction that means ``read a byte from I/O address *0x3f0* into register X". This is exactly how the CPU controls the system's hardware peripherals, by reading and writing to their registers in I/O space. Where in I/O space the common peripherals (IDE controller, serial port, floppy disk controller and so on) have their registers has been set by convention over the years as the PC architecture has developed. The I/O space address *0x3f0* just happens to be the address of one of the serial port's (COM1) control registers.

There are times when controllers need to read or write large amounts of data directly to or from system memory. For example when user data is being written to the hard disk. In this case, Direct Memory Access (DMA) controllers are used to allow hardware peripherals to directly access system memory but this access is under strict control and supervision of the CPU.

10.4.7 Timers

All operating systems need to know the time and so the modern PC includes a special peripheral called the Real Time Clock (RTC). This provides two things: a reliable time of day and an accurate timing interval. The RTC has its own battery so that it continues to run even when the PC is not powered on, this is how your PC always ``knows'' the correct date and time. The interval timer allows the operating system to accurately schedule essential work.

10.5 Address Binding

The Address Binding refers to the mapping of computer instructions and data to physical memory locations. Both logical and physical addresses are used in computer memory. It assigns a physical memory region to a logical pointer by mapping a physical address to a logical address known as a virtual address. It is also a component of computer memory management that the OS performs on behalf of applications that require memory access. The Association of program instruction and data to the actual physical memory locations is called the Address Binding. Let’s consider the following example given below for better understanding. Consider a program P1 has the set of instruction such that I1, I2, I3, I4, and program counter value is 10, 20, 30, 40 respectively.

Program P1

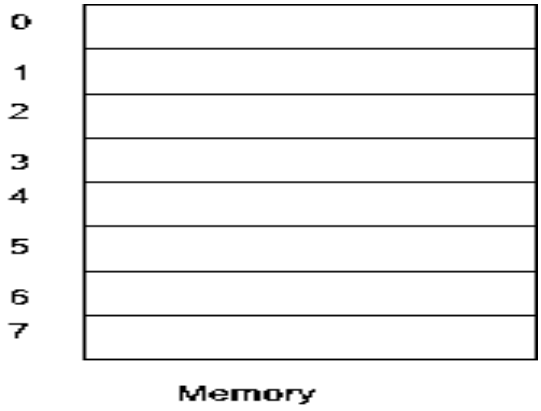
I1 --> 10

I2 --> 20

I3 --> 30

I4 --> 40

Program Counter = 10, 20, 30, 40



10.5.1 Types of Address Binding in Operating System

There are mainly three types of an address binding in the OS. These are as follows:

1. Compile Time Address Binding
2. Load Time Address Binding
3. Execution Time or Dynamic Address Binding

10.5.1.1 Compile Time Address Binding

It is the first type of address binding. It occurs when the compiler is responsible for performing address binding, and the compiler interacts with the operating system to perform the address binding. In other words, when a program is executed, it allocates memory to the system code of the computer. The address binding assigns a logical address to the beginning of the memory segment to store the object code. Memory allocation is a long-term process and may only be modified by recompiling the program.

10.5.1.2 Load Time Address Binding

It is another type of address binding. It is done after loading the program in the memory, and it would be done by the operating system memory manager, i.e., loader. If memory allocation is specified when the program is assigned, no program in its compiled state may ever be transferred from one computer to another. Memory allocations in the executable code may already be in use by another program on the new system. In this case, the logical addresses of the program are not connected to physical addresses until it is applied and loaded into memory.

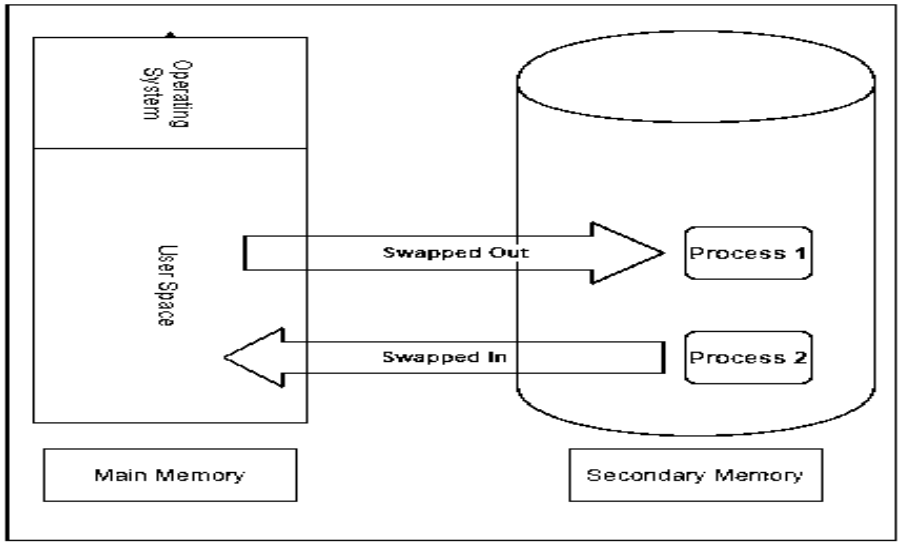
10.5.1.3 Execution Time or Dynamic Address Binding

Execution time address binding is the most popular type of binding for scripts that aren't compiled because it only applies to variables in the program. When a variable in a program is encountered during the processing of instructions in a script, the program seeks memory space for that variable. The memory would assign the space to that variable until the program sequence finished or unless a specific instruction within the script released the memory address connected to a variable.

10.6 Swapping

To increase CPU utilization in multiprogramming, a memory management scheme known as swapping can be used. Swapping is the process of bringing a process into memory and then temporarily copying it

to the disc after it has run for a while. The purpose of swapping in an operating system is to access data on a hard disc and move it to RAM so that application programs can use it. It's important to remember that swapping is only used when data isn't available in RAM. Although the swapping process degrades system performance, it allows larger and multiple processes to run concurrently. Because of this, swapping is also known as memory compaction. The CPU scheduler determines which processes are swapped in and which are swapped out. Consider a multiprogramming environment that employs a priority-based scheduling algorithm. When a high-priority process enters the input queue, a low-priority process is swapped out so the high-priority process can be loaded and executed. When this process terminates, the low priority process is swapped back into memory to continue its execution. Below figure shows the swapping process in operating system:



- Swap-out is a technique for moving a process from RAM to the hard disc.
- Swap-in is a method of transferring a program from a hard disc to main memory, or RAM.

Advantages

- If there is low main memory so some processes may has to wait for much long but by using swapping process do not have to wait long for execution on CPU.
- It utilize the main memory.
- Using only single main memory, multiple process can be run by CPU using swap partition.
- The concept of virtual memory start from here and it utilize it in better way.

- This concept can be useful in priority based scheduling to optimize the swapping process.

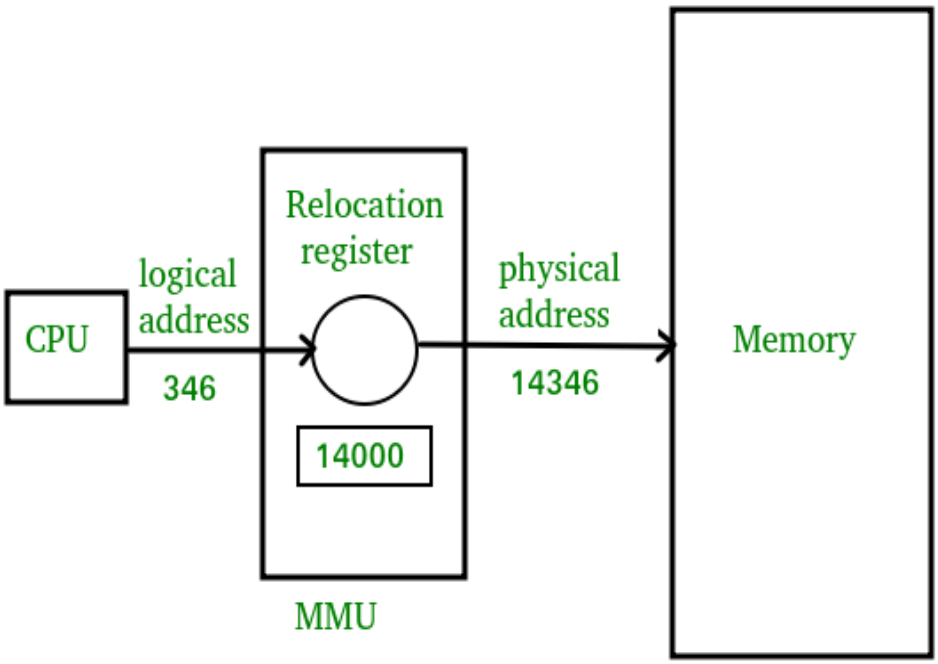
Disadvantages

- If there is low main memory resource and user is executing too many processes and suddenly the power of system goes off there might be a scenario where data get erase of the processes which are took parts in swapping.
- Chances of number of page faults occur
- Low processing performance

Only one process occupies the user program area of memory in a single tasking operating system and remains in memory until the process is completed. When all of the active processes in a multitasking operating system cannot coordinate in main memory, a process is swapped out of main memory so that other processes can enter it.

10.7 Logical and physical address space

- **Logical Address** is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically, therefore, it is also known as Virtual Address. This address is used as a reference to access the physical memory location by CPU. The term Logical Address Space is used for the set of all logical addresses generated by a program's perspective. The hardware device called Memory-Management Unit is used for mapping logical address to its corresponding physical address.
- **Physical Address** identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, the logical address must be mapped to the physical address by MMU before they are used. The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.



Differences between Logical and Physical Address

The basic difference between Logical and physical address is that Logical address is generated by CPU in perspective of a program whereas the physical address is a location that exists in the memory unit.

1. Logical Address Space is the set of all logical addresses generated by CPU for a program whereas the set of all physical address mapped to corresponding logical addresses is called Physical Address Space.
2. The logical address does not exist physically in the memory whereas physical address is a location in the memory that can be accessed physically.
3. Identical logical addresses are generated by Compile-time and Load time address binding methods whereas they differs from each other in run-time address binding method. Please refer this for details.
4. The logical address is generated by the CPU while the program is running whereas the physical address is computed by the Memory Management Unit (MMU).

Parameter	LOGICAL ADDRESS	PHYSICAL ADDRESS

Parameter	LOGICAL ADDRESS	PHYSICAL ADDRESS
Basic	generated by CPU	location in a memory unit
Address Space	Logical Address Space is set of all logical addresses generated by CPU in reference to a program.	Physical Address is set of all physical addresses mapped to the corresponding logical addresses.
Visibility	User can view the logical address of a program.	User can never view physical address of program.
Generation	generated by the CPU	Computed by MMU
Access	The user can use the logical address to access the physical address.	The user can indirectly access physical address but not directly.
Editable	Logical address can be change.	Physical address will not change.
Also called	virtual address.	real address.

Logical Storage Views: viewed by users are a collection of files organized within directories and storage volumes.

- The logical file structure is independent of its physical implementation.
- Logical file structure “ignores”.

Physical storage allocations: records can be stored in separate file locations. Data access methods and Data encoding methods.

Physical Storage Views: a collection of physical storage locations organized as a linear address space.

- The file is subdivided into records.
- The record usually contains information about a single customer, things such as a product in inventory, or an event.
- Records are divided into fields.
- Fields are individual units of data.

10.8 Check Your Progress

1. Which of the following is not an operating system?

   a. Windows

   b. Linux

   c. Oracle

   d. DOS

2. What is the maximum length of the filename in DOS?

   a. 4

   b. 5

   c. 8

   d. 12

3. When was the first operating system developed?

   a. 1948

   b. 1949

   c. 1950

   d. 1951

4. When were MS windows operating systems proposed?

   a. 1994

   b. 1990

c. 1992

d. 1985

5. What is the full name of FAT?

a. File attribute table

b. File allocation table

c. Font attribute table

d. Format allocation table

6. When you delete a file in your computer, where does it go?

a. Recycle bin

b. Hard disk

c. Taskbar

d. None of these

10.9 Summary

The prime responsibility of operating system is to manage the resources of the computer system. In addition to these, Operating System provides an interface between the user and the bare machine. On the basis of their attributes and design objectives, different types of operating systems were defined and characterized with respect to scheduling and management of memory, devices, and files. The primary concerns of a time-sharing system are equitable sharing of resources and responsiveness to interactive requests. Real-time operating systems are mostly concerned with responsive handling of external events generated by the controlled system. Distributed operating systems provide facilities for global naming and accessing of resources, for resource migration, and for distribution of computation. Process scheduling is a very important function of an operating system. Three different schedulers may coexist and interact in a complex operating system: long-term scheduler, medium-term scheduler, and short-term scheduler. Of the presented scheduling disciplines, FCFS scheduling is the easiest to implement but is a poor performer. SRTN scheduling is optimal but unrealizable. RR scheduling is most popular in time- sharing environments, and event-driven and earliest-deadline scheduling are

dominant in real-time and other systems with time-critical requirements. Multiple- level queue scheduling, and its adaptive variant with feedback, is the most general scheduling discipline suitable for complex environments that serve a mixture of processes with different characteristics.

10.10 Keywords

- **Compaction** is to shuffle memory contents and place all free memory together in one block.
- **Program relocatability** refers to the ability to load and execute a given program into an arbitrary place in memory.
- **Contiguous Memory Management**: In this approach, each program occupies a single contiguous block of storage locations.
- **First-fit: This allocates** the first available space that is big enough to accommodate process.
- **Best-fit:** This allocates the smallest hole that is big enough to accommodate process.
- **Worst fit:** This strategy allocates the largest hole.
- **External Fragmentation** is waste of memory between partitions caused by scattered non-contiguous free space.
- **Internal fragmentation** is waste of memory within a partition caused by difference between size of partition and the process allocated

10.11 Self –Assessment Test

1. Explain hardware architecture? What are external and internal hardware in detail?
2. Explain how protection is provided for the hardware resources by the operating system.
3. Define the basic logical structure of computer.
4. Explain what you understand by register explain the various types of registers in CPU
5. What are the system components of an operating system and explain them?
6. What is the difference between static and dynamic address binding?
7. What is the difference between logical address space and physical address space?

10.12 Answer to Check Your Progress

1. C

2. C

3. C

4. D

5. B

6. A

10.13 Suggested Material/ Reference Book

1. Operating System Concepts, 5th Edition, Silberschatz A., Galvin P.B., John Wiley & Sons.

Systems Programming & Operating Systems, 2nd Revised Edition, Dhamdhere D.M., Tata McGraw Hill Publishing Company Ltd., New Delhi.

2 Operating Systems, Godbole A.S., Tata McGraw Hill Publishing Company Ltd., New Delhi.

3. Operating Systems, Madnick S.E., Donovan J.T., Tata McGraw Hill Publishing Company Ltd., New Delhi.

4. Operating Systems-A Modern Perspective, Gary Nutt, Pearson Education Asia, 2000.

5. Operating Systems, Harris J.A., Tata McGraw Hill Publishing Company Ltd., New Delhi, 2002.

Course: MCA-32	Writer: Ritu
Lesson: 11	Vetter:

Paging

Structure

11.2 Translation Look aside Buffer (TLB)

Translation look aside buffer in operating system overcomes the problem the occurs in paging. So, to understand the concept of translation look aside buffer which should have a knowledge about paging and its disadvantages that TLB overcomes.

11.2.1 Paging

- Paging is a memory management scheme that permits the physical address space of a process to be non-contiguous.
- Paging avoids external fragmentation and the need for compaction.
- The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of same size called pages.
- When a process is executed, its pages are loaded into any available memory frames from their source file (a file system or the backing store).
- The backing store is divided into fixed-sized blocks that are the same size as the memory frames or clusters of multiple frames.

Disadvantages of Paging

- Longer memory access time.
- Internal fragmentation to every last page of the process.
- Large memory space is required as page table pre stored in main memory as well.

In Operating System (Memory Management Technique: Paging), for each process page table will be created, which will contain Page Table Entry (PTE). This PTE will contain information like frame number (The address of main memory where we want to refer), and some other useful bits (e.g., valid/invalid bit, dirty bit, protection bit etc). This page table entry (PTE) will tell where in the main memory the actual page is residing. Now the question is where to place the page table, such that overall access time (or reference time) will be less.

The problem initially was to fast access the main memory content based on address generated by CPU (i.e logical/virtual address). Initially, some people thought of using registers to store page table, as they are high-speed memory so access time will be less. The idea used here is, place the page table entries in registers, for each request generated from CPU (virtual address), it will be matched to the appropriate

page number of the page table, which will now tell where in the main memory that corresponding page resides. Everything seems right here, but the problem is register size is small (in practical, it can accommodate maximum of 0.5k to 1k page table entries) and process size may be big hence the required page table will also be big (let's say this page table contains 1M entries), so registers may not hold all the PTE's of Page table. So this is not a practical approach.

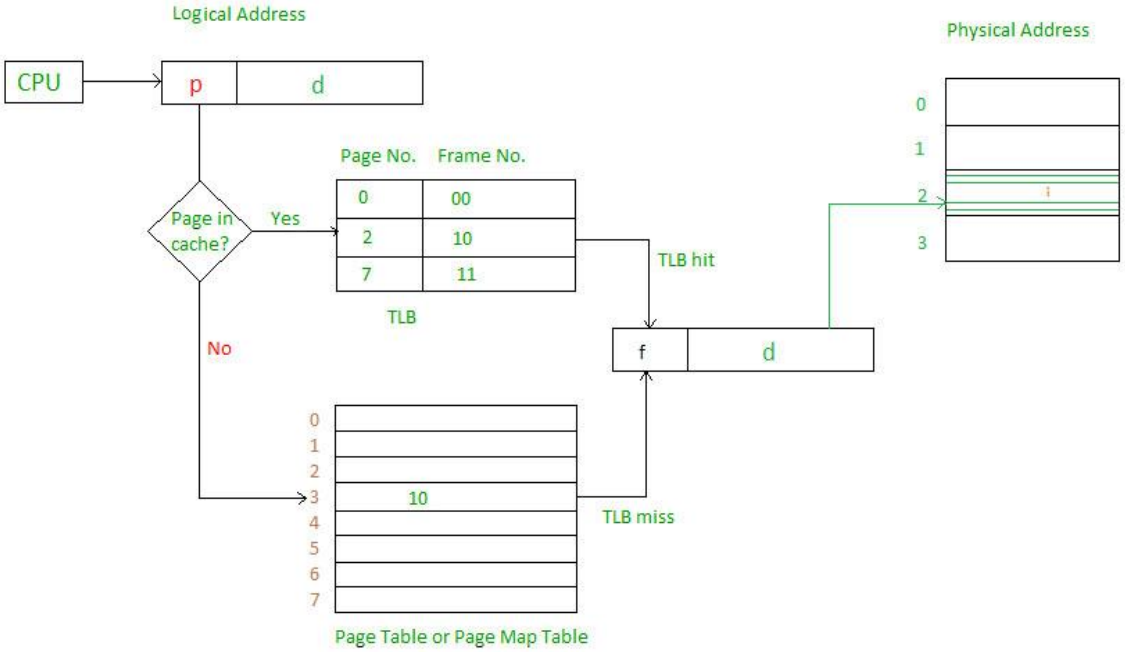
To overcome this size issue, the entire page table was kept in main memory. But the problem here is two main memory references are required:

1) To find the frame number
   To go to the address specified by frame number

To overcome this problem a high-speed cache is set up for page table entries called a Translation Look aside Buffer (TLB). Translation Look aside Buffer (TLB) is nothing but a special cache used to keep track of recently used transactions. TLB contains page table entries that have been most recently used. Given a virtual address, the processor examines the TLB if a page table entry is present (TLB hit), the frame number is retrieved and the real address is formed. If a page table entry is not found in the TLB (TLB miss), the page number is used as index while processing page table. TLB first checks if the page is already in main memory, if not in main memory a page fault is issued then the TLB is updated to include the new page entry.

1. Translation look aside buffer (TLB) is a special, small, fast-lookup hardware cache.
2. The TLB is associative, high-speed, memory.
3. Each entry in the TLB consists of two parts: a key (or tag) and a value.
4. When the associative memory is presented with an item, the item is compared with all keys simultaneously.
5. If the item is found, the corresponding value field is returned.
6. Typically, the number of entries in a TLB is small, often numbering between 64 and 1,0241.
7. The TLB is used with pages tables in the following way
8. The TLB contains only a few of the page-table entries. When a logical address is generated by the CPU, its page number is presented to the TLB.

9. if the page number is found (known as TLB hit), its frame number is immediately available and is used to access memory.



10. If the page number is not found in the TLB (known as TLB miss) a memory reference to the page table must be made.
11. When the frame number is obtained, we can use it to access memory.
12. In addition, we add the page number and frame number to the TLB, so that they will be found quickly on the next reference.
13. If the TLB is already full of entries, the operating system must select one for replacement.
14. Replacement policies range from least recently used (LRU) to random.

Steps in TLB hit:

1. CPU generates virtual (logical) address.
2. It is checked in TLB (present).
3. Corresponding frame number is retrieved, which now tells where in the main memory page lies.

Steps in TLB miss:

CPU generates virtual (logical) address.

1. It is checked in TLB (not present).
2. Now the page number is matched to page table residing in main memory (assuming page table contains all PTE).
3. Corresponding frame number is retrieved, which now tells where in the main memory page lies.
4. The TLB is updated with new PTE (if space is not there, one of the replacement technique comes into picture i.e FIFO, LRU or MFU etc).

Effective memory access time (EMAT): TLB is used to reduce effective memory access time as it is a high speed associative cache.

EMAT = h*(c+m) + (1-h)*(c+2m)

where, h = hit ratio of TLB

m = Memory access time

c = TLB access time

11.3 Demand paging

Every process in the virtual memory contains lots of pages and in some cases; it might not be efficient to swap all the pages for the process at once. Because it might be possible that the program may need only a certain page for the application to run. Let us take an example here, suppose there is a 500 MB application and it may need as little as 100MB pages to be swapped, so in this case, there is no need to swap all pages at once.

The demand paging system is somehow similar to the paging system with swapping where processes mainly reside in the main memory (usually in the hard disk). Thus demand paging is the process that solves the above problem only by swapping the pages on Demand. This is also known as **lazy swapper** (It never swaps the page into the memory unless it is needed). Swapper that deals with the individual pages of a process are referred to as **Pager.** Demand Paging is a technique in which a page is usually brought into the main memory only when it is needed or demanded by the CPU. Initially, only those pages are loaded that are required by the process immediately. Those pages that are never accessed are thus never loaded into the physical memory.

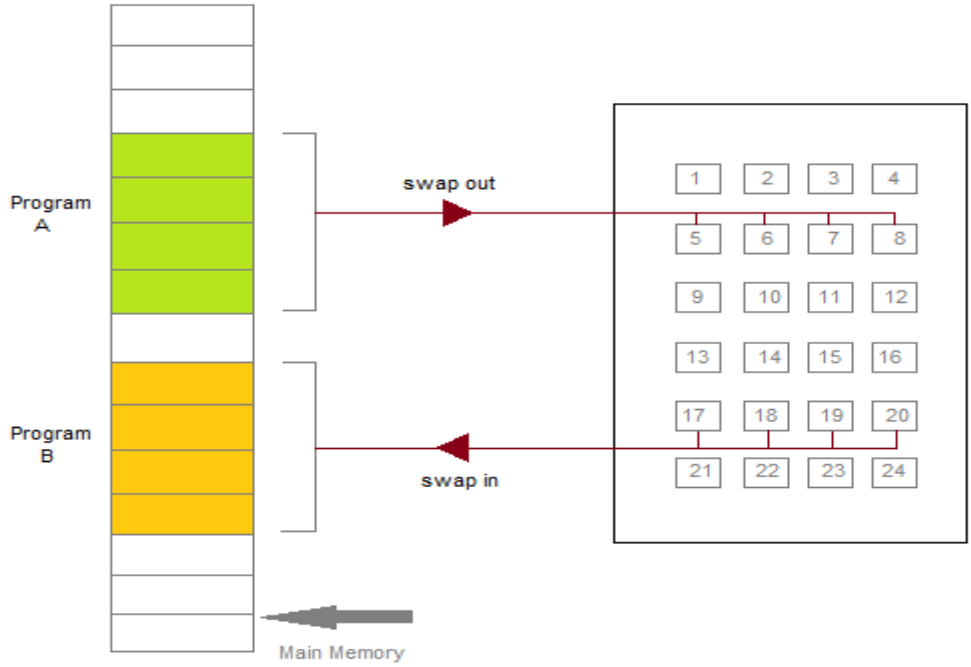


Figure: Transfer of a Paged Memory to the contiguous disk space.

Whenever a page is needed? Make a reference to it;

- If the reference is **invalid** then abort it.
- If the page is Not-in-memory then bring it to memory.

Valid-Invalid Bit

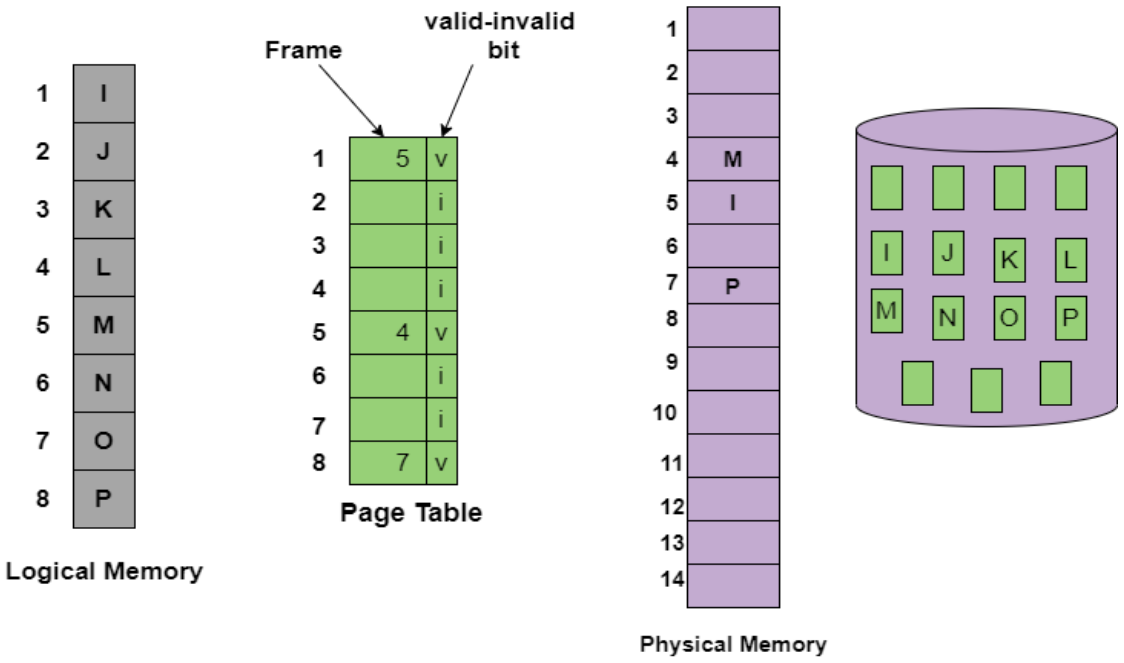
Some form of hardware support is used to distinguish between the pages that are in the memory and the pages that are on the disk. Thus for this purpose Valid-Invalid scheme is used:

- With each page table entry, a valid-invalid bit is associated(where **1** indicates **in the memory** and **0** indicates **not in the memory**)
- Initially, the valid-invalid bit is set to 0 for all table entries.

1. If the bit is set to "**valid**", then the associated page is **both legal and is in memory.**
2. If the bit is set to "**invalid**" then it indicates that the **page is either not valid** or the page is valid **but is currently not on the disk**.
   - For the **pages** that are **brought into the memory**, **the page table is set as usual.**

- But for the **pages** that are **not currently in the memory**, the **page table** is either **simply marked as invalid** or it contains the **address of the page on the disk**.

During the translation of address, if the valid-invalid bit in the page table entry is 0 then it leads to **page fault.**



The above figure is to indicate the page table when some pages are not in the main memory.

How Demand Paging Works?

First of all the components that are involved in the Demand paging process are as follows:

- Main Memory
- CPU
- Secondary Memory
- Interrupt
- Physical Address space
- Logical Address space
- Operating System
- Page Table

1. If a page is not available in the main memory in its active state; then a request may be made to the CPU for that page. Thus for this purpose, it has to generate an interrupt.
2. After that, the Operating system moves the process to the blocked state as an interrupt has occurred.
3. Then after this, the Operating system searches the given page in the Logical address space.
4. And finally with the help of the page replacement algorithms, replacements are made in the physical address space. Page tables are updated simultaneously.
5. After that, the CPU is informed about that update and then asked to go ahead with the execution and the process gets back into its ready state.

When the process requires any of the pages that are not loaded into the memory, a page fault trap is triggered and the following steps are followed,

1. The memory address which is requested by the process is first checked, to verify the request made by the process.
2. If it is found to be invalid, the process is terminated.
3. In case the request by the process is valid, a free frame is located, possibly from a free-frame list, where the required page will be moved.
4. A new operation is scheduled to move the necessary page from the disk to the specified memory location. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)
5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to valid.
6. The instruction that caused the page fault must now be restarted from the beginning.

Advantages of Demand Paging

The benefits of using the Demand Paging technique are as follows:

- With the help of Demand Paging, memory is utilized efficiently.
- Demand paging avoids External Fragmentation.
- Less Input/output is needed for Demand Paging.

- This process is not constrained by the size of physical memory.
- With Demand Paging it becomes easier to share the pages.
- With this technique, portions of the process that are never called are never loaded.
- No compaction is required in demand Paging.

Disadvantages of Demand paging

Drawbacks of Demand Paging are as follows:

- There is an increase in overheads due to interrupts and page tables.
- Memory access time in demand paging is longer.

11.3.1 Pure Demand Paging

In some cases when initially no pages are loaded into the memory, pages in such cases are only loaded when are demanded by the process by generating page faults. It is then referred to as **Pure Demand Paging**.

- In the case of pure demand paging, there is not even a single page that is loaded into the memory initially. Thus pure demand paging causes the page fault.
- When the execution of the process starts with no pages in the memory, then the operating system sets the instruction pointer to the first instruction of the process and that is on a non-memory resident page and then in this case the process immediately faults for the page.
- After that when this page is brought into the memory then the process continues its execution, page fault is necessary until every page that it needs is in the memory.
- And at this point, it can execute with no more faults.
- This scheme is referred to as Pure Demand Paging: means never bring a page into the memory until it is required.

11.4 Page Replacement Algorithm

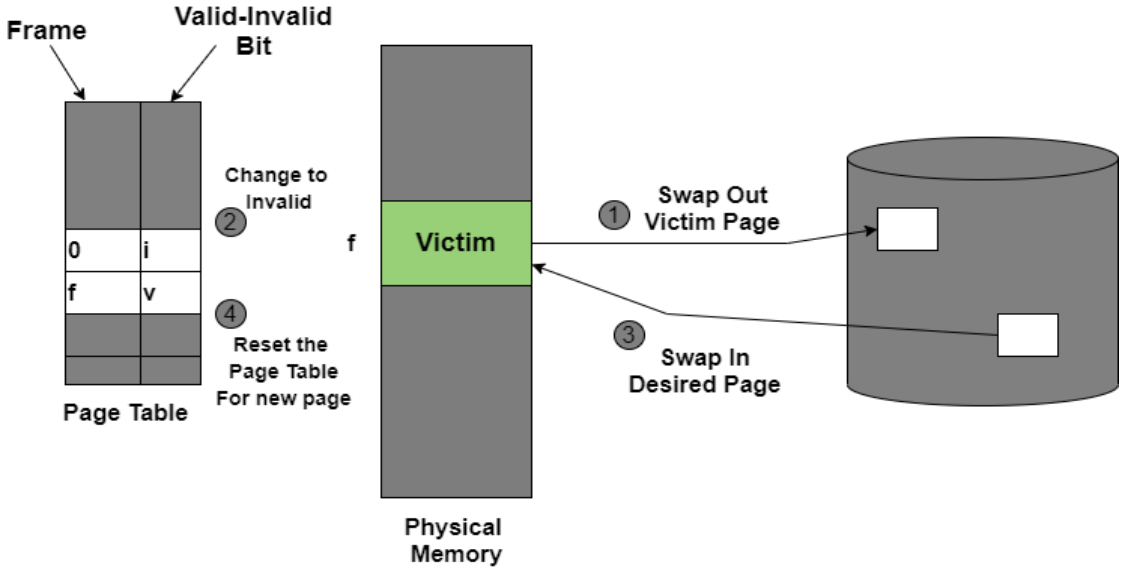
In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.

Page Fault: A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page faults happen. In case of a page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

11.4.1 Basic Page Replacement Algorithm

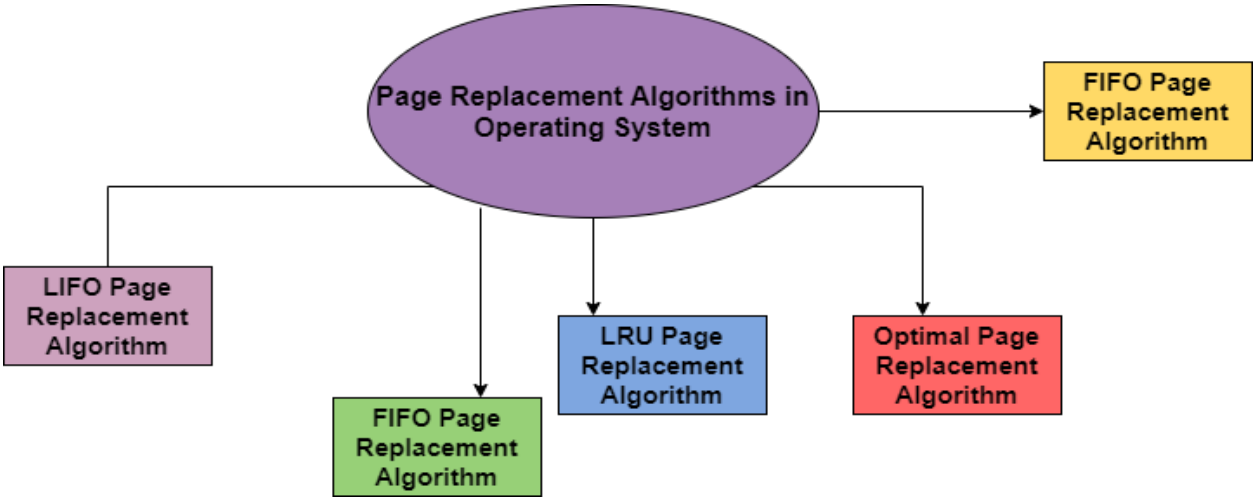
Page Replacement technique uses the following approach. If there is no free frame, then we will find the one that is not currently being used and then free it. A-frame can be freed by writing its content to swap space and then change the page table in order to indicate that the page is no longer in the memory.

1. First of all, find the location of the desired page on the disk.
2. Find a free Frame: a) If there is a free frame, then use it. b) If there is no free frame then make use of the page-replacement algorithm in order to select the victim frame. c) Then after that write the victim frame to the disk and then make the changes in the page table and frame table accordingly.
3. After that read the desired page into the newly freed frame and then change the page and frame tables.
4. Restart the process.



11.4.2 Page Replacement Algorithms:

This algorithm helps to decide which pages must be swapped out from the main memory in order to create a room for the incoming page. This Algorithm wants the lowest page-fault rate. Various Page Replacement algorithms used in the Operating system are as follows;

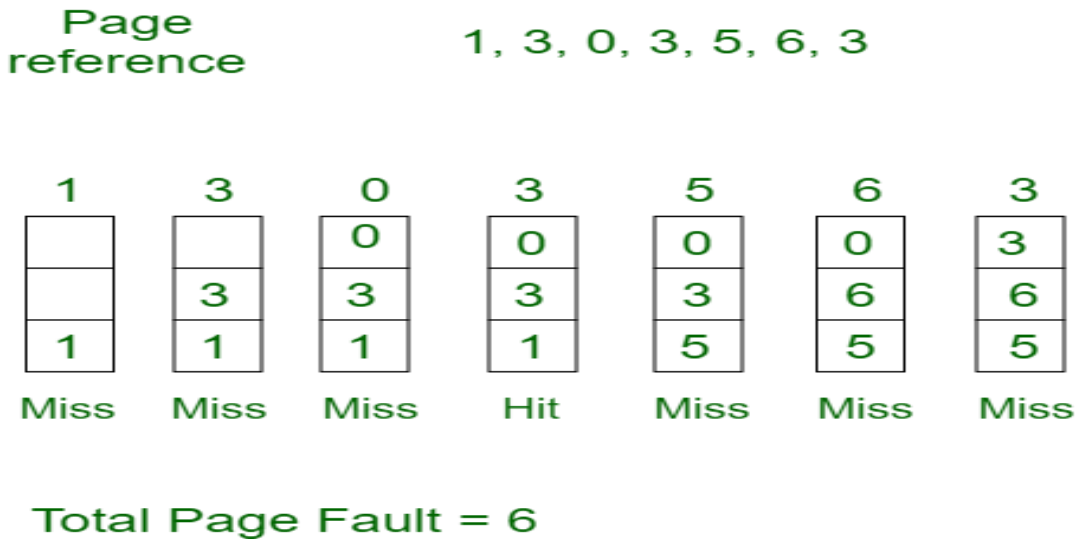


11.4.3 First In First Out (FIFO):

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

- This algorithm is implemented by keeping the track of all the pages in the queue.
- As new pages are requested and are swapped in, they are added to the tail of a queue and the page which is at the head becomes the victim.
- This is not an effective way of page replacement but it can be used for small systems.

Example 1: Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3 page frames. Find the number of page faults.



Initially, all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.** When 3 come, it is already in memory so —> **0 Page Faults.** Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e. 1. —>**1 Page Fault.** 6 comes, it is also not available in memory so it replaces the oldest page slot i.e. 3 —>**1 Page Fault.** Finally, when 3 come it is not available so it replaces 0 **1 page faults.**

Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference strings 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4, and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10-page faults.

Advantages

- This algorithm is simple and easy to use.
- FIFO does not cause more overhead.

Disadvantages

- This algorithm does not make the use of the frequency of **last used time rather** it just replaces the Oldest Page.
- There is an increase in **page faults** as page frames increases.
- The performance of this algorithm is the worst.

11.4.4 Optimal Page replacement

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

- Practical implementation is not possible because we cannot predict in advance those pages that will not be used for the longest time in the future.
- This algorithm leads to less number of page faults and thus is the best-known algorithm

Also, this algorithm can be used to measure the performance of other algorithms.

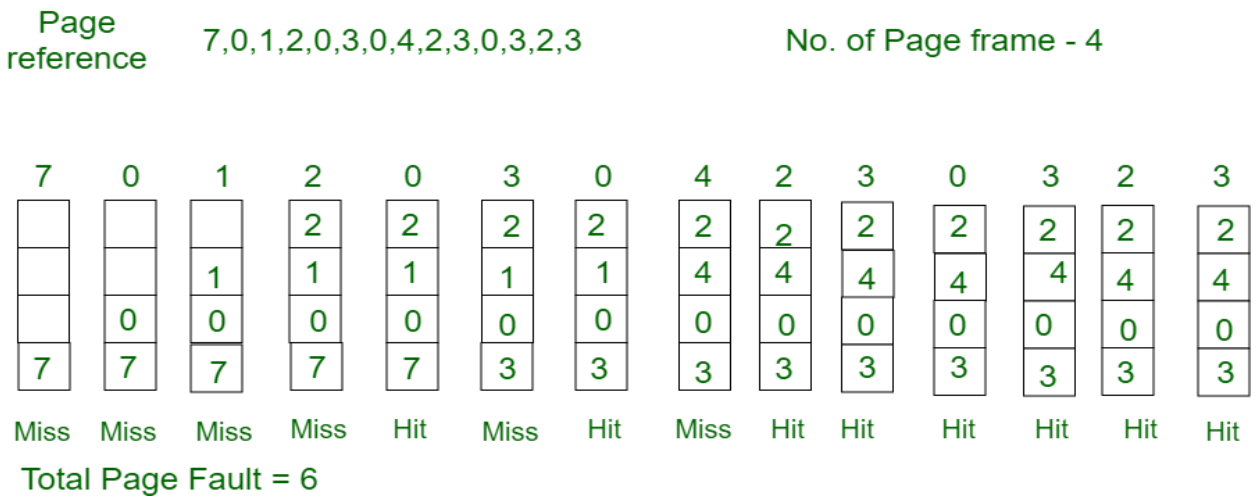
Advantages of OPR

- This algorithm is easy to use.
- This algorithm provides excellent efficiency and is less complex.
- For the best result, the implementation of data structures is very easy

Disadvantages of OPR

- In this algorithm future awareness of the program is needed.
- Practical Implementation is not possible because the operating system is unable to track the future request

Example-2: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frame. Find number of page fault.



Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults** 0 is already there so —> **0 Page fault.**

When 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.** 0 is already there so —> **0 Page fault.** 4 will take place of 1 —> **1 Page Fault.**

Now for the further page reference string —> **0 Page fault** because they are already available in the memory. Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

11.4.5. Least Recently Used:

In this algorithm, page will be replaced which is least recently used. The page that has not been used for the longest time in the main memory will be selected for replacement.

- This algorithm is easy to implement.
- This algorithm makes use of the counter along with the even-page.

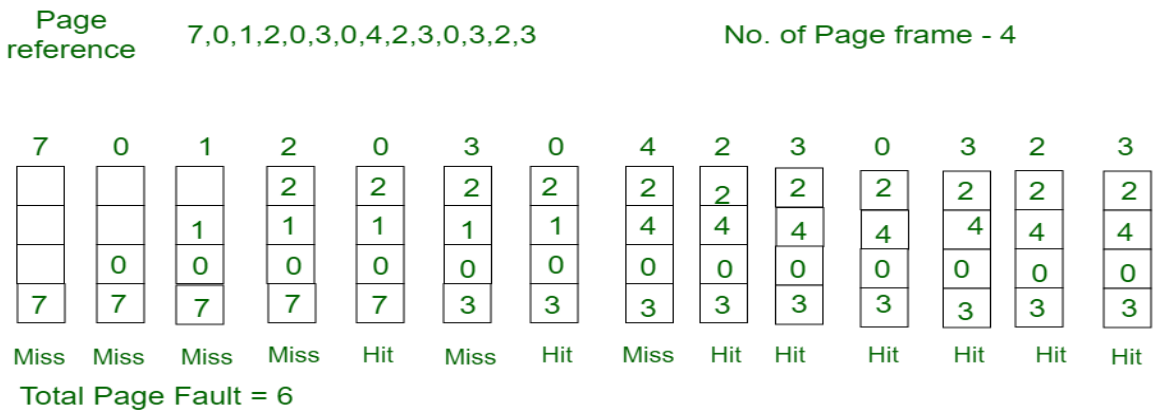
Advantages of LRU

- It is an efficient technique.
- With this algorithm, it becomes easy to identify the faulty pages that are not needed for a long time.
- It helps in Full analysis.

Disadvantages of LRU

- It is expensive and has more complexity.
- There is a need for an additional data structure.

Example-3: Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frames. Find number of page faults.



Here LRU has same number of page fault as optimal but it may differ according to question.

Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults** 0 is already their so —> **0 Page fault.** When 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**

0 is already in memory so —> **0 Page fault**.

4 will takes place of 1 —> **1 Page Fault**

Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

11.4.6. Random Page Replacement Algorithm

As indicated from the name this algorithm replaces the page randomly. This Algorithm can work like any other page replacement algorithm that is LIFO, FIFO, Optimal, and LRU.

11.5 Check Your Progress

1. Physical memory is broken into Łxed-sized blocks called _____

a) frames

b) pages

c) backing store

d) none of the mentioned

2. Every address generated by the CPU is divided into two parts. They are _____

a) frame bit & page number

b) page number & page oàset

c) page oàset & frame bit

d) frame oàset & page oàset

3. If the size of logical address space is 2 to the power of m, and a page size is 2 to the power of

a) m, n

b) n, m

c) m – n, m

d) m – n, n

4. The operating system maintains a _____ table that keeps track of how many frames have been allocated, how many are there, and how many are available.

a) page

b) mapping

c) frame

d) memory

5. For larger page tables, they are kept in main memory and a _____ points to the page table.

a) page table base register

b) page table base pointer

6. Each entry in a translation look aside buffer (TLB) consists of _____

a) key

b) value

c) bit value

d) constant

7. If a page number is not found in the TLB, then it is known as a _____

a) TLB miss

b) Buffer miss

c) TLB hit

d) All of the mentioned

8. When the valid – invalid bit is set to valid, it means that the associated page _____

a) is in the TLB

b) has data in it

c) is in the process's logical address space

d) is the system's physical address space

9. Illegal addresses are trapped using the _____ bit.

a) error

b) protection

c) valid – invalid

d) access

10. In paged memory systems, if the page size is increased, then the internal fragmentation generally _____

a) becomes less

b) becomes more

c) remains constant

d) none of the mentioned

11.6 Summary

The memory-management layer of an OS allocates & reclaims portions of main memory in response to requests from other users & from other OS modules, & in accordance with the resource-management objectives of a particular system. Processes are created & loaded into memory in response to scheduling decisions that are affected by, among other things, the amount of memory available for allocation at a given instant. Memory is normally freed when resident objects terminate. When it is necessary & cost-effective, the memory manager may increase the amount of available memory by moving inactive or low-priority objects to lower levels of the memory hierarchy (swapping).

Thus, the memory manager interacts with the scheduler in selecting the objects to be placed into or evicted from the main memory. The mechanics of memory management consists of allocating & reclaiming space, & of keeping track of the state of memory areas. The objective of memory

management is to provide efficient use of memory by minimizing the amount of wasted memory while imposing little storage, computational, & memory-access overhead. In addition, the memory manager should provide protection by isolating distinct address spaces, & facilitate inter-process cooperation by allowing access to shared data & code. Partitioned allocation of memory imposes relatively little overhead, but it restricts sharing & suffers from internal or external fragmentation. Segmentation reduces the impact of fragmentation & offers superior protection & sharing by dividing each process's address space into logically related entities that may be placed into dis-contiguous areas of physical memory. Contiguity of the virtual-address space is maintained by performing address translation at instruction execution time. Hardware assistance is usually provided for this operation in order to avoid a drastic reduction of the effective memory bandwidth. Paging simplifies allocation & de-allocation of memory by dividing address spaces into fixed-sized chunks. Execution-time translation of virtual to physical addresses, usually assisted by hardware, is used to bridge the gap between contiguous virtual addresses & dis-contiguous physical addresses where different pages may reside.

Virtual memory removes the restriction on the size of address spaces of individual processes that is imposed by the capacity of the physical memory installed in a given system. In addition, virtual memory provides for dynamic migration of portions of address spaces between primary & secondary memory fin accordance with the relative frequency of usage.

11.7 Keywords

Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory.

Segmentation is a memory management technique in which, the memory is divided into the variable size parts. Each part is known as segment which can be allocated to a process.

Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.

Physical address space in a system can be defined as the size of the main memory. It is really important to compare the process size with the physical address space.

11.8 Self-Assessment Test

1. What is the common drawback of all the real memory management techniques? How is it overcome in virtual memory management schemes?
2. What is the basic difference between paging & segmentation? Which one is better & why?
3. What extra hardware do we require for implementing demand paging & demand segmentation?
4. Differentiate between internal & external fragmentation.
5. What do you understand by thrashing? What are the factors causing it?

11.9 Answers to Check Your Progress

1. A
2. B
3. D
4. C
5. A
6. A
7. A
8. C
9. C
10. B

11.10 References/Suggested Readings

11. OS Concepts, 5th Edition, Silberschatz A., Galvin P.B., John Wiley & Sons.
12. Systems Programming & OSs, 2nd Revised Edition, Dhamdhere D.M., Tata McGraw Hill Publishing Company Ltd., New Delhi.
13. OSs, Madnick S.E., Donovan J.T., Tata McGraw Hill Publishing Company Ltd., New Delhi.
14. OSs-A Modern Perspective, Gary Nutt, Pearson Education Asia, 2000.
15. OSs, Harris J.A., Tata McGraw Hill Publishing Company Ltd., New Delhi, 2002.

Course: MCA-32	Writer:
Lesson: 12	Vetter:

Windows, UNIX and Linux System

Structure

12.0 Learning Objectives

In this unit you will get a detailed knowledge of the commonly used operating system viz. MS-Windows, UNIX and Linux. The commands and functions of each system have been dealt with in details to initiate you in using these operating systems.

12.1 Introduction

The oldest of all Microsoft's operating systems is MS-DOS (Microsoft Disk Operating System).

MS-DOS is a text-based operating system. Users have to type commands rather than use the friendlier graphical user interfaces (GUI's) available today. Despite its very basic appearance, MS-DOS is a very powerful operating system. There are many advanced applications and games available for MS-DOS. A version of MS-DOS underpins Windows. Many advanced administration tasks in Windows can only be performed using MS-DOS. Windows gives life to modern PCs; it's what sets new PCs apart from their dull typewriter like forebears. Windows gives you a graphical environment, so you can see what you're doing. Windows gives a consistent point and shoot user interface, so that allocation unit of all software works alike. And Windows gives you the ability to run multiple programs at once, so your computer can keep up with the most valuable resource around you.

12.2 Windows

An **Operating system (OS)** is software which acts as an interface between the end user and computer hardware. Every computer must have at least one OS to run other programs. An application like Chrome, MS Word, Games, etc needs some environment in which it will run and

perform its task. The OS helps you to communicate with the computer without knowing how to speak the computer's language. It is not possible for the user to use any computer or mobile device without having an operating system. Majority of home users use a Windows based machine. Most of today's applications and games are designed to run solely on Microsoft systems.

12.2.1 Advantage of Using WINDOWS vs DOS

Windows brings together features available in older style operating systems and adds to them user friendly features so that you can do more work quicker from your desktop or portable computer. Some of the main benefits include:

- Improved interface
- Easier file management, including support for networked connections and long filenames
- New Plug and Play feature automatically detects and uses additional devices you attach to your computer
- True 32-bit multitasking enables several programs to run simultaneously so you can get more work done
- Improved search facility
- Better multimedia support
- Extended communications capability, including E-mail, faxes, bulletin boards, Internet.

The basic foundation underlying Windows is its 'windowing' capability. A window (spelt with a lower-case w) is a rectangular area used to display information or to run a program. Several windows can be opened at the same time to work with multiple applications, so you should be able to dramatically increase your productivity when using your computer.

12.2.2 Functions of an Operating System

In an operating system software performs each of the function:

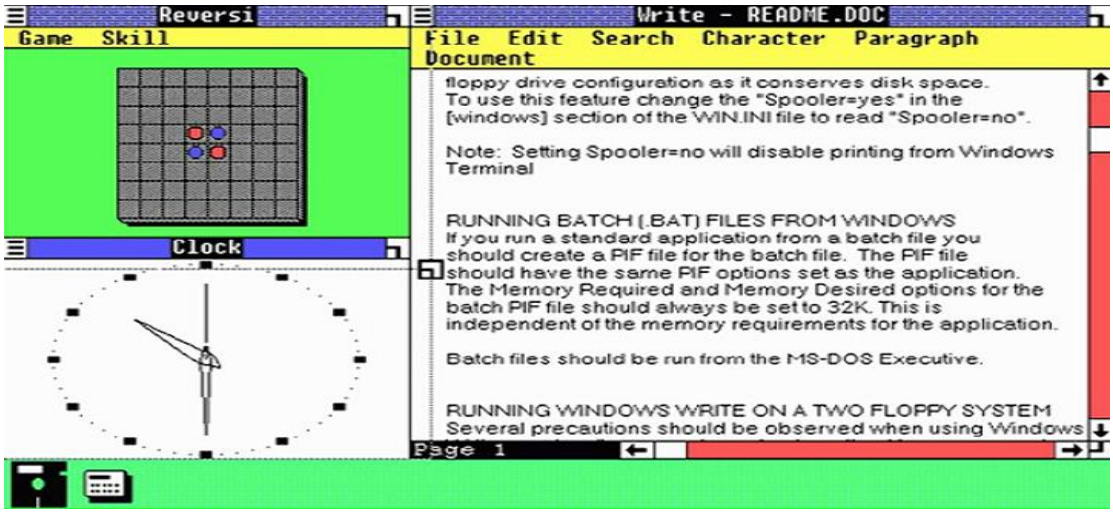
1. **Process management:-** Process management helps OS to create and delete processes. It also provides mechanisms for synchronization and communication among processes.
2. **Memory management:-** Memory management module performs the task of allocation and de-allocation of memory space to programs in need of this resources.
3. **File management:-** It manages all the file-related activities such as organization storage, retrieval, naming, sharing, and protection of files.
4. **Security:-** Security module protects the data and information of a computer system against malware threat and authorized access.

5. **Command interpretation:** This module is interpreting commands given by the user and acting system resources to process that commands.
6. **Job accounting:** Keeping track of time & resource used by various job and users.
7. **Communication management:** Coordination and assignment of compilers, interpreters, and another software resource of the various users of the computer systems.

12.3 Versions and History of MS WINDOWS

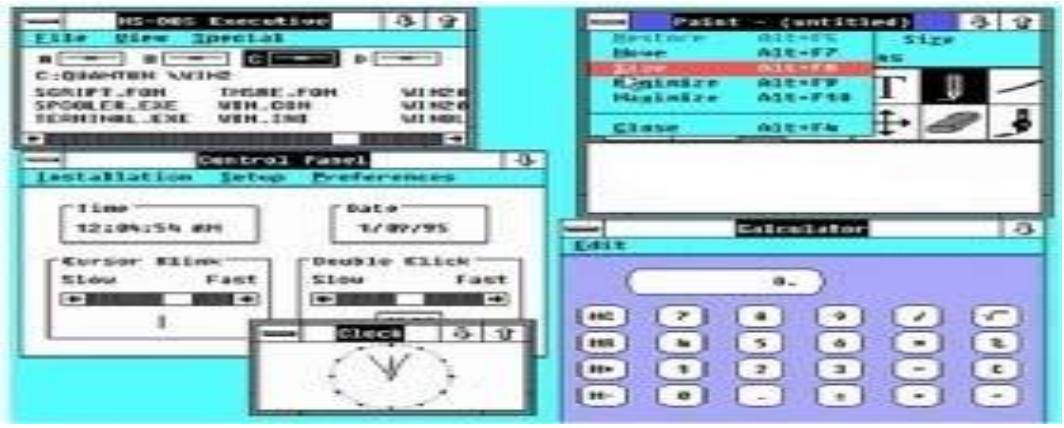
Windows versions through the years

12.3.1 1985: Windows 1.0



The history of Microsoft Windows dates back to 1985, when Microsoft released Microsoft Windows Version 1.01. Microsoft's aim was to provide a friendly user-interface known as a GUI (graphical user interface) which allowed for easier navigation of the system features. Windows 1.01 never really caught on. The release was a shaky start for the tech giant. Users found the software unstable. (The amazing thing about Windows 1.01 is that it fitted on a single floppy disk). However, the point-and-click interface made it easier for new users to operate a computer. Windows 1.0 offered many of the common components found in today's graphical user interface, such as scroll bars and "OK" buttons.

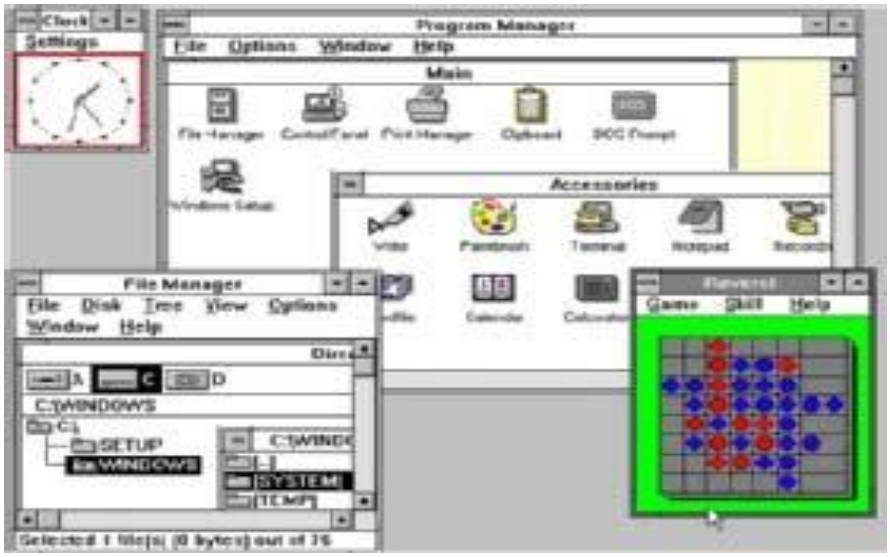
1987: Windows



Windows 2.0 was faster, more stable and had more GUI features. The GUI was very slightly improved but still looked too similar to Windows 1.01.

The system introduced the control panel and ran the first versions of Excel and Word. Windows 2.0 supported extended memory, and Microsoft updated it for compatibility with Intel's 80386 processor. It was during this time that Microsoft became the largest software vendor in the world, just as computers were becoming more commonplace. The fact that Windows systems were user-friendly and relatively affordable was a contributing factor to the growing PC market.

12.3.2 1990: Windows 3.0



Windows 3.0 supported 16 colors and included the casual games familiar to most Windows users: Solitaire, Minesweeper and Hearts. Games that required more processing power still ran directly on MS-DOS. Exiting to DOS gave games direct hardware access made more system resources available. Microsoft made an enormous impression with Windows 3.0 and 3.1. Graphics and functionality were drastically improved. The Windows 3 family provided multimedia capabilities as well as vastly improved graphics and application support. Building on the success of Windows 3.x, Microsoft released Microsoft Windows 3.11 for Workgroups. This gave Windows the ability to function on a network.

12.3.3 1993: Windows New Technology (NT)

Windows NT's release marked the completion of a side project to build a new, advanced OS. NT was 32-bit and had a hardware abstraction layer. DOS was available through the command prompt, but it did not run the Windows OS. Microsoft designed NT as a workstation OS for businesses rather than home users. The system introduced the Start button.

12.3.4 1995: Windows 95

In 1995 Windows went through a major revamp and Microsoft Windows 95 was released. This provided greatly improved multimedia and a much more polished user interface. The now familiar desktop and Start Menu appeared. Internet and networking support was built in. Although Windows 95 was a home user operating system, it proved to be very popular in schools and businesses. Windows 95 facilitated hardware installation with its Plug and Play feature. Microsoft also

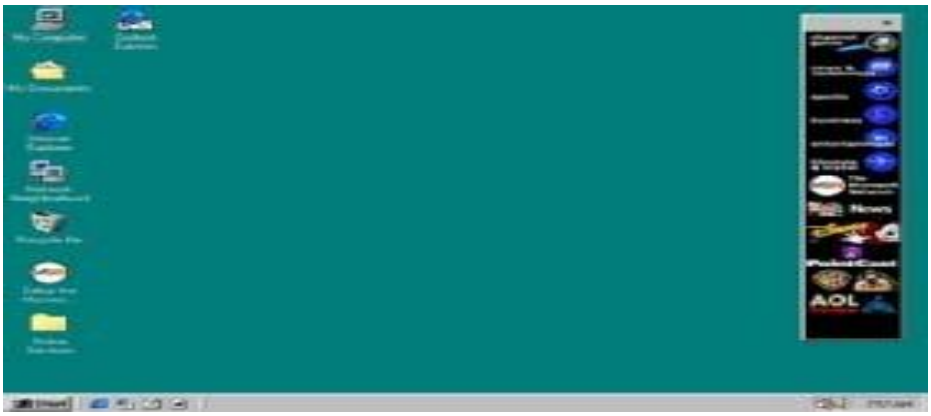
unveiled 32-bit color depth, enhanced multimedia capabilities and TCP/IP network support.

12.3.5 1998: Windows 98

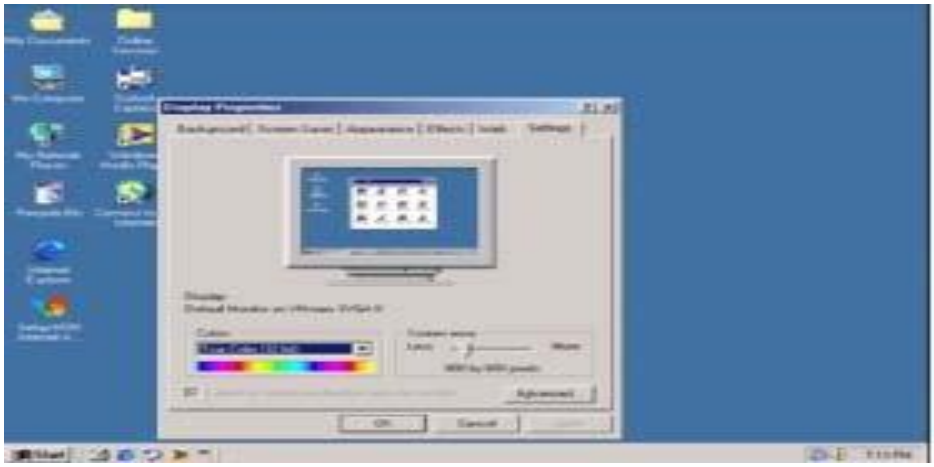
Windows 98 was officially released on June 25th 1998. Internet Explorer 4.0 is integrated as a component of Windows 98 just like Internet Explorer 3.0 to OSR2. There is also a built-in TV tuner, a better Dial-Up Network and a system shield called Dr.Watson which prevents crashes ensuring a better and smoother environment. Although Microsoft makes no guarantees that general system performance will increase after installing Windows 98, many users have seen an

improvement after upgrading from Windows 95 + IE4.0. 486DX / 66 MHz, 16 MB of memory Typical installation requires approximately 195 MB of free hard disk space, but may range between 120 MB to 295 MB, depending on your system configuration and the options you choose to install. CD-ROM or DVDROM drive (3.5" high-density disks available for additional charge). VGA or higher-resolution monitor. Microsoft Mouse or compatible pointing device. The

Windows 98 interface is more like a Web browser. So you can use the same navigation and viewing techniques whether you're searching your hard disk or the Internet. Most of these options can be adjusted or switched off to resemble Windows 95.



12.3.6 2000: Windows Millennium Edition (ME)



Windows ME (Millennium Edition) was the last use of the Windows 95 codebase. Its most notable new feature was System Restore. Many customers found this release to be unstable. Some critics said ME stood for "mistake edition."

Microsoft released the professional desktop OS Windows 2000 (initially called NT 5.0) in the same year for the business market. Improvements to the overall operating system allowed for easier configuration and installation. Microsoft based this OS on the more stable Windows NT code. Some home users installed Windows 2000 for its greater reliability. Microsoft updated Plug and Play support, which spurred home users to switch to this OS. One big advantage of Windows 2000 was that operating system settings could be modified easily without the need to restart the machine. Windows 2000 proved to be a very stable operating system that offered enhanced security and ease of

administration.

12.3.7 2001: Windows XP

Microsoft delivered Windows XP as the first NT-based system with a version aimed squarely at the home user. Home users and critics rated XP highly. The system improved Windows appearance with colorful themes and provided a more stable platform. Microsoft virtually ended gaming in DOS with this release. DirectX-enabled features in 3D gaming that OpenGL had difficulties with. XP offered the first Windows support for 64-bit computing, but it was not very well supported, lacking drivers and applications to run.

12.3.8 2006: Windows Vista

Microsoft hyped Windows Vista after the company spent a lot of resources to develop a more polished appearance. Vista had interesting visual effects but the OS was slow to start and run.Vista's flaws -- coupled with the fact that many older computers lacked the resources to run the system -- led to many home and business users staying with XP.

12.3.9 2009: Windows 7

Microsoft built Windows 7 on the Vista kernel. Windows 7 picked up Vista's visual capabilities but featured more stability. To many end users, the biggest changes between Vista and Windows 7 were faster boot times, new user interface and the addition of Internet Explorer 8. With true 64- bit support and more Direct X features, Windows 7 proved to be a popular release for Windows users.

12.3.10 2012: Windows 8

Microsoft released Windows 8 with a number of enhancements and distributed its tile- based Metro user interface. Windows 8 took better advantage of multicore processing, solid-state drives (SSD),

touch screens and other alternate input methods.

12.3.11 2015: Windows 10

Microsoft announced Windows 10 in September 2014, skipping Windows 9 and launched on July 2015. Version 10 includes the Start menu, which was absent from Windows 8. A responsive design feature called Continuum adapts the interface depending on whether the user works with a touch screen or a keyboard and mouse for input.

New features like an onscreen back button simplified touch input. Microsoft designed the OS to have a consistent interface across devices including PCs, laptops and tablets.

12.4 LINUX OS

Linux is an operating system or a kernel. It is distributed under an open source license. Its functionality list is quite like UNIX.

Who created Linux?

Linux is an operating system or a kernel which germinated as an idea in the mind of young and bright **Linus Torvalds** when he was a computer science student. He used to work on the **UNIX OS (proprietary software)** and thought that it needed improvements. However, when his suggestions were rejected by the designers of UNIX, he thought of launching an OS which will be **receptive to changes, modifications suggested by its users**.

As time passed by, he collaborated with other **programmers in places like MIT** and applications for Linux started to appear. So around 1991, a working Linux operating system with some applications

was officially launched, and this was the start of one of the **most loved and open-source OS options available today**. The earlier versions of Linux were not so user-friendly as they were in use by computer programmers and **Linus Torvalds never had it in mind to commercialize** his product. This definitely curbed the Linux's popularity as other commercially oriented Operating System Windows got famous. Nonetheless, the open-source aspect of the Linux operating system made it more robust.

The main advantage of Linux was that programmers were able to use the Linux Kernel to design their own custom operating systems. With time, a new range of user-friendly OS's stormed the computer world. Now, **Linux is one of the most popular and widely used Kernel,** and it is the backbone of popular operating systems like **Debian, Knoppix, Ubuntu, and Fedora**. Nevertheless, the list does not end here as there are thousands of OS's based on Linux which offer a variety of functions to the users. Linux Kernel is normally used in combination of GNU project by Dr. Richard Stallman. All modern distributions of Linux are actually distributions of Linux/GNU

12.4.1 Properties of Linux

Linux Pros

A lot of the advantages of Linux are a consequence of Linux' origins, deeply rooted in UNIX, except for the first advantage, of course:

1. Linux is free:

If you want to spend absolutely nothing, you don't even have to pay the price of a CD. Linux can be downloaded in its entirety from the Internet completely for free. No registration fees, no costs per user, free updates, and freely available source code in case you want to change the behavior of your system.

2. Most of all, Linux is free as in free speech:

The license commonly used is the GNU Public License (GPL). The license says that anybody who may want to do so has the right to change Linux and eventually to redistribute a changed version, on the one condition that the code is still available after redistribution.

3. Linux is portable to any hardware platform:

A vendor who wants to sell a new type of computer and who doesn't know what kind of OS his new

machine will run (say the CPU in your car or washing machine), can take a Linux kernel and make it work on his hardware.

4. Linux is secure and versatile:

The security model used in Linux is based on the UNIX idea of security, which is known to be robust and of proven quality. But Linux is not only fit for use as a fort against enemy attacks from the Internet: it will adapt equally to other situations, utilizing the same high standards for security. Your development machine or control station will be as secure as your firewall.

5. Linux is scalable:

From a Palmtop with 2 MB of memory to a peta byte storage cluster with hundreds of nodes: add or remove the appropriate packages and Linux fits all. You don't need a supercomputer anymore, because you can use Linux to do big things using the building blocks provided with the system. If you want to do little things, such as making an operating system for an embedded processor or just recycling your old 486, Linux will do that as well.

6. The Linux OS and most Linux applications have very short debug-times:

Because Linux has been developed and tested by thousands of people, both errors and people to fix them are usually found rather quickly. It sometimes happens that there are only a couple of hours between discovery and fixing of a bug.

12.4.2 Linux Cons

There are far too many different distributions:

"Quot capites, tot rationes", as the Romans already said: the more people, the more opinions. At first glance, the amount of Linux distributions can be frightening, or ridiculous, depending on point of view. But it also means that everyone will find what he or she needs. You don't need to be an expert to find a suitable release.

When asked, generally every Linux user will say that the best distribution is the specific version he is using. So which one should you choose? Don't worry too much about that: all releases contain more or less the same set of basic packages. On top of the basics, special third party software is added making, for example, Turbo Linux more suitable for the small and medium enterprise, Red Hat for servers and SuSE for workstations. However, the differences are likely to be very superficial. The

best strategy is to test a couple of distributions; unfortunately not everybody has the time for this. Luckily, there is plenty of advice on the subject of choosing your Linux. A quick search on Google, using the keywords "choosing your distribution" brings up tens of links to good advice. The Installation HOWTO also discusses choosing your distribution.

Linux is not very user friendly and confusing for beginners:

It must be said that Linux, at least the core system, is less user-friendly to use than MS Windows and certainly more difficult than MacOS, but... In light of its popularity, considerable effort has been made to make Linux even easier to use, especially for new users.

Is an Open Source product trustworthy?

How can something that is free also be reliable? Linux users have the choice whether to use Linux or not, which gives them an enormous advantage compared to users of proprietary software, who don't have that kind of freedom. After long periods of testing, most Linux users come to the conclusion that Linux is not only as good, but in many cases better and faster that the traditional solutions. If Linux were not trustworthy, it would have been long gone, never knowing the popularity it has now, with millions of users. Now users can influence their systems and share their remarks with the community, so the system gets better and better every day.

12.5 UNIX

After three decades of use, the UNIX computer operating system from Bell Labs is still regarded as one of the most powerful, versatile, and flexible operating systems (OS) in the computer world. Its popularity is due to many factors, including its ability to run a wide variety of machines, from micros to supercomputers, and its portability – all of which led to its adoption by many manufacturers.

12.5.1 History of UNIX

Like another legendary creature whose name also ends in ‘x,’ UNIX raised from the ashes of a multi-organizational effort in the early 1960s to develop a dependable timesharing operating system. The joint effort was not successful, but a few survivors from Bell Labs tried again, and what followed was a system that offers its users a work environment that has been described as “of unusual simplicity, power, and elegance....” The system also fostered a distinctive approach to software design – solving a problem by interconnecting simpler tools, rather than creating large monolithic application programs.

Its development and evolution led to a new philosophy of computing, and it has been a never-ending source of both challenges and joy to programmers around the world.

12.5.2 Main Features

Multi-user Capabilities

A multi-user system permits several users to use the same computer simultaneously. More than one terminal can be connected to one computer, and the users of the terminal can run allocation unit programs, access files, and print the documents at once. The O/S manages the requests made to the computer by managing users, sorts them from interfering with each other, and assigns priorities when two or more users want to access the same file at the same time. A computer system that can support multiple users is generally less expensive than the equivalent number of single user machines.

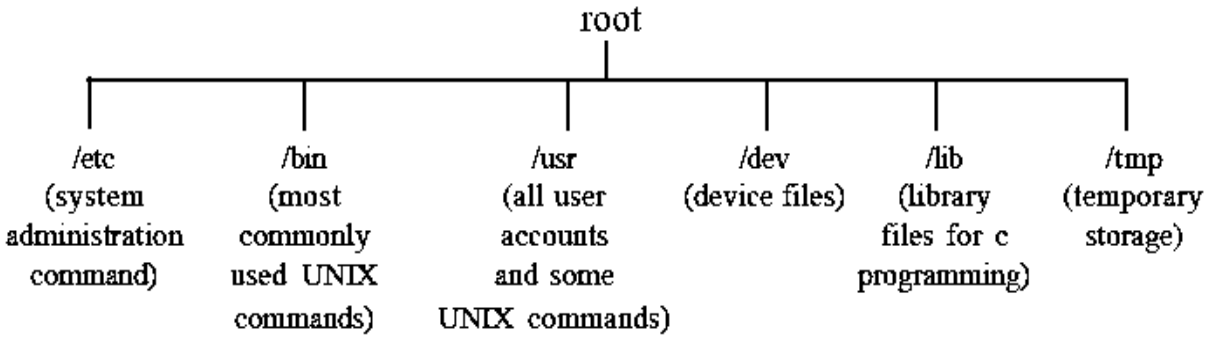
Multitasking Capabilities

Multitasking means that a given user can do more than one task at the same time. Multitasking with computer lets you simultaneously perform tasks formerly performed sequentially. This helps the set of tasks to be completed quickly. This is achieved by placing some tasks in the “background” and some of the tasks in the “foreground”.

Portability

Portability is the ability of the software that operates on one machine to operate on another, different machine. Hence, it is easier to modify the UNIX system code for installation on a new computer. The ability to transport the UNIX system from one brand of computer to another has been the major reason of acceptance of this system. The Unix System runs on more brands and types of computers.

The Hierarchical File System



At the top is the root or the first directory from which the second level of the directories descends. The initial level of the directories is standard to most UNIX systems and is described in the above figure.

12.5.3 Unix Operating System

The UNIX operating system is made up of three parts; the kernel, the shell and the programs.

The kernel

The kernel of UNIX is the hub of the operating system. It allocates time and memory to programs and handles the file store and communications in response to system calls. As an illustration of the way that the shell and the kernel work together, suppose a user types **rm myfile** (which has the effect of removing the file "**myfile**"). The shell searches the file store for the file containing the program **rm**, and then

requests the kernel, through system calls, to execute the program **rm** on **myfile**. When the process **rm myfile** has finished running, the shell then returns the UNIX prompt % to the user, indicating that it is waiting for further commands.

The shell

The shell acts as an interface between the user and the kernel. When a user logs in, the login program checks the username and password, and then starts another program called the shell. The shell is a command line interpreter (CLI). It interprets the commands the user types in and arranges for them to be carried out. The commands are themselves programs. When they terminate, the shell gives the user another prompt (% on our systems). The adept user can customize his/her own shell, and users can use different shells on the same machine. Staff and students in the school have the **tcsh** shell by default. The tcsh shell has certain features to help the user inputting commands.

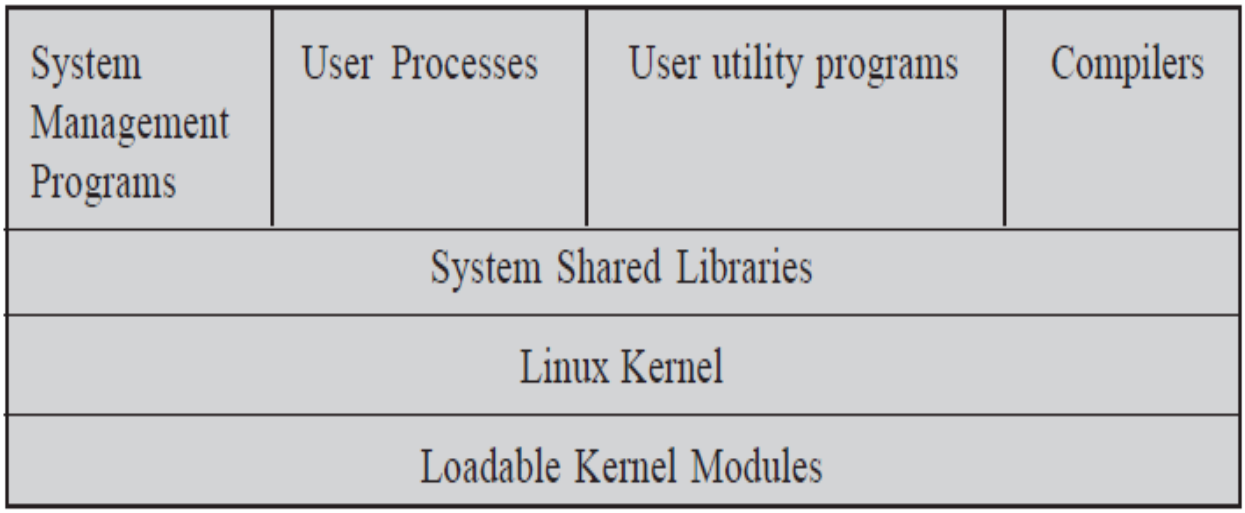
12.5.4 Components of a Linux System

The Linux system is composed of three main bodies of code, in line with most traditional UNIX implementations:

1) **Kernel:** the kernel is responsible for maintaining all the important abstractions of the operating system, including such things as virtual memory and processes.

2) **System Libraries:** the system libraries define a standard set of functions through which applications can interact with the kernel and that implementation much of the operating system functionally that does not need the full privileges of the kernel code.

3) **System utilities:** the system utilities are programs that perform individual, specialized management tasks. Some system utilities may be invoked just once to initialize and configure some aspect of the system, others known as deamons in UNIX terminology may run permanently, handling tasks such as responding to incoming network connections, accepting logon requests from terminals or updating log files.



The above figure illustrates the various components of the Linux system. The important distinction here is between the Kernel and everything else. All the kernel code executes in the processor's privileged mode with full access to all the physical resources of the computer. Linux refers to this privileged mode as **Kernel Mode**. Under Linux, no user-mode code is built into the kernel. Any operating-system-support code that does not need to run in kernel mode is placed into the system libraries.

Linux Kernel and its Modules

The kernel

The kernel is the heart of the system. It manages the communication between the underlying hardware and the peripherals. The kernel also makes sure that processes and server processes are started and stopped the kernel is the most important file on the system. The Linux kernel has the ability to load and

unload arbitrary sections of kernel code on demand. These loadable kernel modules run in privileged kernel mode and have full access to all the hardware capabilities of

the machine on which they run. Kernel modules are convenient for several reasons. Linux's source code is free, so anybody wanting to write kernel code is able to compile a modified kernel and to reboot to load that new functionality, however recompiling, relinking and reloading the entire kernel is cumbersome procedure, when you are developing a new driver. If you use kernel modules, you do not have to make a new kernel to test a new driver; the driver may be compiled on its own and loaded into the already-running kernel. Kernel Modules allows a Linux system to be set up with a standard, minimal kernel without any extra device drivers built in. Any device drivers that the user needs can be either loaded explicitly by the system at startup or loaded automatically by the system on demand and unloaded when not in use. For Example, a CD-ROM driver might be loaded when a CD is mounted and unloaded from memory when the CD is dismounted from the file system.

The Module support under Linux has three components:

1) The **Module management** allows module to be loaded into memory and to talk to the rest of the kernel.

2) The **driver registration** allows modules to tell the rest of the kernel that a new driver has become available.

3) A **conflict-resolution mechanism** allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another driver.

The shell

The shell is an advanced way of communicating with the system, because it allows for conversation and taking initiative. Both partners in the communication are equal, so new ideas can be tested. The shell allows the user to handle a system in a flexible way, and is also a means of stress reduction.

Shell types

There are different types of shells, which are:

- **sh or Bourne Shell:** The original shell still used on UNIX systems and in UNIX related environments. This is the basic shell, a small program with few features. **bash or Bourne Again shell:** the standard GNU shell, intuitive and flexible. Probably most advisable for beginning

users while being at the same time a powerful tool for the advanced and professional user. On Linux, **bash** is the standard shell for common users. This shell is a so-called superset of the Bourne shell, a set of add-ons and plug-ins. The Bourne Again shell is compatible with the Bourne shell.

- **csh or C shell:** the syntax of this shell resembles that of the C programming language. Sometimes used by programmers.
- **tcsh or Turbo C shell:** a superset of the common C shell, enhancing user friendliness and speed.
- **ksh or the Korn shell:** it is appreciated by people with a UNIX background.

This shell is the superset of the Bourne shell with standard configuration.

Drawbacks of Using Linux

The biggest disadvantage of using Linux is the fact that no single corporate entity is in charge of its development. There are several disadvantages discussed below:

1) **Lack of technical Support:**

2) **Hardware problems:** one disadvantage of Linux is that it can be hard to install and does not work on all hardware platforms. Unlike a commercial program development operation, where a cohesive group spends months building and testing a program against a variety of conditions and hardware, Linux developers are scattered around the globe. There are no formal quality-assurance programs. The hardware supported by Linux depends on the hardware each developer owns while writing that portion of the code.

3) **Inability to use current software:** Another disadvantage is that your current applications for such operating systems as DOS are more likely won't work under Linux. You cannot use both operating systems at the same time; you can leave Linux and boot the other operating system to use your applications there.

4) **Lack of Experience:** finally, unless you are already a UNIX user, you must learn how to manage a Linux system. Linux and UNIX need to be managed. The system administrator is responsible for maintaining the system and for performing such duties as adding and deleting user accounts, backing up

the system on a regular basis, installing new software, configuring the system and fixing things when they go wrong.

5) **Inconsistent User Interface:** the development of GNU/Linux reflects different interfaces, design goals, etc. It is not easy learning to use the vi editor, for example, or learning the command line syntax of find. Nor is there any consistency enforced among the various programs and utilities included in a standard Linux distribution. This leads to user confusion and frustration

12.6 Linux as compared to UNIX

Linux was developed considering UNIX as a reference model. This is why; the basic architecture and most of the features of Linux and UNIX are the same. Actually, Linux is also considered another version of UNIX. The main difference between Linux and UNIX is that Linux is FREE. Various distributors of Linux charges a price, which is quite low as compared to other operating systems. The Linux operating system requires at least 850 MB hard disk space; 64MB Memory for Red hat Linux 9.0.

Comparison between Linux and UNIX

Feature	Linux	Unix
Shell Available	bash, pdksh, tcsh zsh, ash	Bourne, korn, C
Variants	Red Hat, Caldera, Debain, LinuxPPC, SUSE	AT&T, MULTICS, BSD, SCO, HP-Ux, IRIX, Ultrix, Sun Solaris
Licensing	Freely Distributed	Expensive licensing

Linux as compared to Windows Most of the organizations having small, medium, or large networks require a few basic services such as file and print services, E-mail access, Internet access, and Intranet services. Most of the tasks that you are wanting to execute on a Windows machine can also be executed on a Linux machine. The following table compares some of the common applications used on the Linux and Windows operating systems. Comparison between Linux and Windows

Service	Linux	Windows
Web Server	Apache	Internet Information Server
Email Server	Send mail	Microsoft Exchange
Relational Database	Sybase / MYSQL	Microsoft SQL Server
Proxy Server	Squid Object Cache	Windows Pro Server
Backup Server	BRU	Included with Windows

The distributor of Linux A Linux distribution is collection software of (usually open source) software on top of a Linux kernel. A distribution (or short, distro) can bundle server software, system management tools, documentation and many desktop applications in a central secure software repository. A distro aims to provide a common look and feel, secure and easy software management and often a specific operational purpose. Let's take a look at some popular distributions. Red Hat Red Hat is a billion dollar commercial Linux Company that puts a lot of effort in developing Linux. They have hundreds of Linux specialists and are known for their excellent support. They give their products (Red Hat Enterprise Linux and Fedora) away for free. While Red Hat Enterprise Linux (RHEL) is well tested before release and supported for up to seven years after release, Fedora is a distro with faster updates but without support. Ubuntu Canonical started sending out free compact discs with Ubuntu Linux in 2004 and quickly became popular for home users (many switching from Microsoft Windows). Canonical wants Ubuntu to be an easy to use graphical Linux desktop without need to ever see a command line. Of course they also want to make a profit by selling support for Ubuntu. Debian There is no company behind Debian. Instead there are thousands of well organised developers that elect a Debian Project Leader every two years. Debian is seen as one of the most stable Linux distributions. It is also the basis of every release of Ubuntu. Debian comes in three versions: stable, testing and unstable. Every Debian release is named after a character in the movie Toy Story.

A comparison between Windows and Linux is as follows:

Parameters	Linux	Windows
Open Source	Linux is Open Source and is free to use.	Windows is not open source and is not free to use.
Case sensitivity	Linux file system is case sensitive.	Windows file system is case insensitive.
kernel type	Linux uses monolithic kernel.	Windows uses micro kernel.
Efficiency	Linux is more efficient in operations as compared to Windows.	Windows is less efficient in operations.
Kernel	Uses Monolithic Kernel which consumes more running space.	Uses Micro Kernel which takes less space but lowers the system running efficiency
Path Separator	Linux uses forward slash as path separator between directories.	Windows uses backward slash as a path separator.
Security	Linux is highly secure as compared to Windows.	Windows provides less security as compared to Linux.
User Account	1. Regular 2. Root 3. Service Account	1. Administrator 2. Standard 3. child 4. Guest

12.7 Check your progress

1. If there are multiple recycle bin for a hard disk

a. you can set the different size for each recycles bin

b. you can choose which recycle bin to use to store your deleted files

c. You can make any one of them default recycle bin.

d. None of the above

2. Identify false statement

a. You can find deleted files in recycle bin

b. You can restore any files in recycle bin if you ever need

c. You can increase free space on the disk by sending files in recycle bin.

d. You can right-click and choose Empty Recycle Bin to clean it at once

3. If the displayed system time and date are wrong, you can reset it using

a. Write

b. Calendar

c. Write file.

d. Control panel

4. You should save your computer from?

a. Viruses

b. Time bombs

c. Worms

d. All of the above

5. World Wide Web is being a standard by

a. Worldwide corporation

b. W3C

c. World Wide Consortium

d. World Wide Web Standard

6. A co-processor

a. Is relatively easy to support in software

b. Causes all processor to function equally

c. Works with any application

12.7 Key Words

Hierarchical File System Linux provides a standard file structure in which system files/ user files are arranged.

Multiprogramming Linux is a multiprogramming system means multiple applications can run at same time.

Time-Sharing Multi-programming is made possible on the Linux system by time-sharing feature.

Multi-Tasking A program in Linux is broken down into tasks; each task is something like reading from or writing to the disk, or waiting for input from a user. The ability of any as to handle the execution of multiple tasks is known as multi-tasking

Multi-User Linux is a multiuser system means multiple users can access system resources like memory, ram, and application programs at same time.

12.8 Self Check Exercises

1) Discuss the main features of UNIX OS?

2) Describe the ports of UNIX OS?

3) What are the basic components of Linux operating system?

4) What are the different types of files provided by Linux?

12.9 Answer to check your progress

1. A
2. C
3. D
4. D
5. B
6. A
7. A
8. A
9. A
10. A

12.9 References and Further Reading

1. Duncan, Ray (1994). Advanced MSDOS programming - New Delhi: BPB Publication.

2. Harris, J. Archer (2002). Schanm's Outlines Operating Systems. New Delhi: Tata Mcgraw Hill.

3. Kerninghan, Brian and Pike, Rob (1984). The Unix Programming Environment- New Jersey: Prentice-Hall.

4. Levine, John and Young, Margaret Levine (1988). The Complete Reference Windows 98 - New Delhi: Tata Mc-graw Hill.

5. Stroel, Stefan and Ulil, Thomas (1996). Linux Universe Installation and Configuration - 2nd ed. - Heil bronn : Springer.

6. Comparison Of Linux And Windows Smita Parale, 1institute Of Computer Science And Technology, B. K. Birla College Of Arts, Commerce And Science (Autonomous), Kalyan, Mumbai, India.

NOTES

NOTES

NOTES

NOTES